Energy-Quality-Time Fault Tolerant Task Mapping on Multicore Architectures

Minyu Cui

école
normale
supérieure
rennes

THÈSE DE DOCTORAT DE

L'ÉCOLE NORMALE
SUPERIEURE RENNES
COMUE UNIVERSITÉ BRETAGNE LOIRE

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Informatique

Par

Minyu CUI

Energy Quality Time Fault Tolerant Task Mapping on Multicore Architectures

Thèse présentée et soutenue à Rennes , le 24/06/2022
Unité de recherche : IRISA (UMR 6074), Institut de Recherche en Informatique et Systèmes Aléatoires

Rapporteurs avant soutenance :

Christophe JEGO	Professeur des Universités, Bordeaux INP
Smail NIAR	Professeur des Universités, Université Polytechnique Hauts-de-France

Composition du Jury :

Examinateurs :	Daniel CHILLET	Professeur des Universités, Université Rennes1
	Christophe JEGO	Professeur des Universités, Bordeaux INP
	Smail NIAR	Professeur des Universités, Université Polytechnique Hauts-de-France
	Claire PAGETTI	Ingénieur de recherche HDR, ONERA
Dir. de thèse :	Emmanuel CASSEAU	Professeur des Universités, Université de Rennes 1
Co-enc. de thèse :	Angeliki KRITIKAKOU	Maître de Conférences, Université de Rennes 1

Acknowledgement

During my PhD studies, I have met many people who affected me personally and scientifically. Their presence was essential and meaningful to support me to come here, I dedicate all my thanks to them.

I would firstly like to thank my supervisors Prof. Emmanuel Casseau and Prof. Angeliki Kritikakou, for the guidance, encouragement and advice during the past few years. This dissertation could not have been completed without their consistent help. I would also like to mention a special thank to Prof. Lei Mo for his guidance and advice in the discussion of research works.

Secondly, I would like to thank the members of the TARAN team in Inria Rennes for the interesting scientific discussions in reading group and relaxing coffee breaks.

Finally, I would like to thank all the people who love me and whom I love, those who accompany me on my life journey and give me their love, support and kindness.

Table of Contents

List of acronyms

SoA	State of Art
DVFS	Dynamic Voltage and Frequency Scaling
WCET	Worst Case Execution Time
WCEC	Worst Case Execution Cycle
SL	Schedule Length of DAG
EST	Earliest Start Time
LFT	Latest Finish Time
st	Actual start time
ft	Actual finish time of task
et	Execution time
pred	All immediate predecessors
succ	All immediate successors
SC	Selected Configuration
NC	New checked Configuration
proc	Task set that are allocated on processor
RAFTM	Proposed Reliability-aware Fault-tolerant Task Mapping approach
RAM	Reliability-Aware Mapping approach in SoA
TDM	Duplication Mapping approach in SoA
O_RAFTM	Proposed RAFTM approach with optimal solutions
H_RAFTM	Proposed RAFTM approach with heuristic solutions
O_RAM	RAM approach with optimal solutions
H_RAM	RAM approach with heuristic solutions
O_TDM	TDM approach with optimal solutions
H_TDM	TDM approach with heuristic solutions
EC	Energy consumption
RI	Reliability improvement
CT	Computation time

List of Figures

List of Tables

Résumé en français

Contexte

Les systèmes critiques pour la sécurité consistent généralement en des systèmes où des garanties doivent être fournies sur la sécurité et la fiabilité des applications critiques, ce qui implique à la fois une tolérance élevée aux pannes et des contraintes temporelles strictes (temps réel) [1]. Par exemple, les systèmes avioniques sont constitués d'applications avec des niveaux élevés d'assurance de conception [2] et fonctionnant à des altitudes élevées exposées aux rayonnements. Les systèmes spatiaux consistent en des applications de contrôle de la navigation fonctionnant dans l'espace extra-atmosphérique avec des particules et un rayonnement électromagnétique à haute énergie. Les systèmes automobiles ont des applications comme par exemple un capteur dans une roue pour à la fois le contrôle de la stabilité et la régulation de l'accélération et sont soumis à des particules alpha, des pics de température élevés et de l'interférences électromagnétiques [3]. Ces stimuli naturels sont à l'origine de défauts qui impactent le fonctionnement du système [4]. De plus, au cours des trente dernières années, la taille du code des applications avioniques, spatiales et automobiles a considérablement augmenté [5]. Ces systèmes font face à une croissance exponentielle des exigences de performances, et les futures applications automobiles et aérospatiales nécessiteront des ressources de calcul encore plus performantes [1].

Pour faire face aux demandes croissantes de performances, le marché grand public s'est tourné vers les architectures multicœurs, en raison de la consommation d'énergie et des limites de dissipation thermique des processeurs monocœurs [1]. D'une manière générale, les multicœurs offrent des réductions de taille, de poids, de puissance consommée et ont des capacités de calcul élevées par rapport aux processeurs monocœur, et peuvent donc intégrer plusieurs applications sur la même plate-forme [6]. Cependant,deux principaux défis scientifiques sont soulevés par l'utilisation des multicœurs.

Le premier défi est la consommation d'énergie élevée qui est devenue l'un des plus grands obstacles au développement des systèmes informatiques hautes performances, en particulier pour les systèmes à budget énergétique limité, tels que les objets connectés alimentés par batterie ou à récupération d'énergie. Les smartphones utilisent des architectures multicœurs hétérogènes, telles que big.LITTLE [7], qui se compose de gros cœurs optimisés en termes de performances et de petits cœurs optimisés en énergie avec une seule architecture de jeu d'instructions (ISA) [8]. Les cartes embarquées pour les objets connectés, telles Raspberry Pi, Odroid, Edison, Jetson et Artik, disposent également de plusieurs cœurs [9, 10, 11]. Par conséquent, les plates-formes

multicœurs ont été améliorées avec la possibilité de régler leur tension et leur fréquence (gestion dynamique de la tension et de la fréquence (DVFS) pendant l'exécution pour équilibrer les performances du système et les économies d'énergie.

Le deuxième défi est que le système multicœur lui-même est susceptible de subir des défauts en raison de la nature des systèmes électroniques. Conjugués à la réduction de la taille des transistors, les systèmes multicœurs deviennent de plus en plus sensibles aux conditions de fonctionnement et à l'impact environnemental [12]. Dans les systèmes électroniques, la variation de la tension de seuil dépend de la largeur du transistor, tandis que des trous ou de petites fissures dans les interconnexions entraînent des problèmes de source fermée ou ouverte. L'activité électrique et les points chauds sont inévitables et ils provoquent une électromigration, une instabilité de température de polarisation et une diaphonie, qui sont des sources de défauts. Pour améliorer la fiabilité du multicœur, soit des processeurs durcis aux radiations sont utilisés, soit le système est répliqué [13]. La première solution conduit à des systèmes avec des capacités de calcul limitées et nécessite une expertise de conception difficile à trouver. La seconde solution a un coût et une consommation d'énergie élevés. Pour réduire les coûts tout en assurant la fiabilité, la réplication des ressources et le surdimensionnement du système doivent être évités, dans la mesure du possible.

En tenant compte des deux défis présentés ci-dessus, afin d'exploiter pleinement les fonctionnalités des systèmes multicœurs tout en visant une exécution de l'application à la fois fiable, économe en énergie et satisfaisant la contrainte temps-réel, des méthodes sont nécessaires pour décider de l'exécution efficace des tâches. La manière dont les tâches sont exécutées sur une plateforme est déterminée par plusieurs facteurs. Le premier facteur est l'ordonnancement (à quel moment chaque tâche commence son exécution) et l'allocation (sur quel processeur chaque tâche est exécutée) des tâches. Le deuxième facteur est la décision de l'assignation de la tension et la fréquence du processeur lorsqu'il exécute une tâche spécifique, ce qui détermine le temps d'exécution de la tâche. Pour les applications critiques, les limites de pire temps d'exécution (Worst Case Execution Time (WCET)) sont utilisées car elles sont nécessaires pour garantir la fiabilité et le bon comportement fonctionnel.

Modèles systèmes et ordonnancement temps-réel de tâches

L'objectif de cette thèse est de trouver un compromis multicritères d'un ordonnancement de tâches sur des architectures multicœurs, tolérant aux fautes, efficace en énergie et temps-réel. Dans un premier temps, nous définissons l'architecture et les modèles systèmes utilisés dans cette thèse :

`Les architectures multicœurs`

Pour répondre à l'augmentation rapide des besoins de calcul, à une faible consommation du

système, ainsi qu'assurer un parallélisme élevé dans l'exécutions des applications, les systèmes multicœurs sont des plates-formes prometteuses pour les systèmes embarqués temps réel. Sur le marché des circuits, de nombreux fabricants de puces, par exemple AMD et Intel, ont lancé des puces multicœurs avec un nombre croissant de cœurs, comme par exemple la série Intel Xeon. Une architecture multicœur se compose de deux ou plusieurs unités de traitement séparées (cœurs) sur un seul circuit intégré. Chaque cœur exécute les instructions du processeur en même temps, ce qui augmente la vitesse globale du calcul parallèle. Les architectures multicœurs peuvent être caractérisées comme des systèmes homogènes ou hétérogènes. Les systèmes multicœurs homogènes incluent des cœurs identiques comme certains systèmes sur puce multiprocesseurs (MPSoC) couramment utilisés construits avec des cœurs ARM Cortex [14]. Les systèmes multicœurs hétérogènes combinent différents cœurs, comme par exemple l'architecture ARM big.LITTLE avec de « gros » processeurs avec des performances plus élevées mais gourmandes en énergie, comme l'A-15, et des « petits » processeurs avec des performances inférieures mais une meilleure efficacité énergétique, comme l'A-7 [15]. Dans cette thèse, pour faciliter l'approche (mais sans perte de généralité), nous nous concentrons sur des plateformes homogènes [16] comme par exemple le système Cortex-A53 quadricœur Arm.

`Le modèle de tâches`

Comme introduit dans [17], une tâche fait référence à un ensemble d'activités cohérentes qui sont exécutées afin d'atteindre un but dans un domaine donné. Une application est généralement représentée par un ensemble de tâches. Selon les relations entre les tâches, deux catégories de modèles de tâches sont considérées dans cette thèse, à savoir les tâches indépendantes et les tâches dépendantes. Lorsque, dans certaines applications, des activités de calcul peuvent être exécutées dans un ordre arbitraire, celles-ci peuvent être considérées comme des tâches *indépendantes*. Sinon, les tâches sont *dépendantes* : elles doivent respecter les relations de précédence (ou de dépendance), c'est-à-dire qu'une tâche s'appuie sur des entrées fournies par d'autres tâches. De telles relations de dépendance entre tâches sont utilisées pour construire des graphes acycliques dirigés (directed acyclic graphs : DAG) pour décrire l'application [18]. Les DAG sont utilisés pour la représentation du calcul, de la communication et des dépendances des tâches applicatives. Un graphe DAG G consiste en une paire $G = \{\boldsymbol{V}, \boldsymbol{E}\}$ où $\boldsymbol{V}$ est l'ensemble des sommets et $\boldsymbol{E}$ est l'ensemble des arêtes dirigées qui représentent la communication de données entre les tâches [19]. Un sommet cotrespond à une tâche et une arête représente une relation de dépendance. Pour deux tâches τ_i et τ_j, si $(\tau_i, \tau_j) \in \boldsymbol{E}$, la tâche τ_j dépend de la tâche τ_i et ne peut commencer son exécution qu'une fois τ_i ait terminé son exécution. Si une tâche τ_j dépend de la tâche τ_i (c'est-à-dire que la tâche τ_i et la tâche τ_j ont une relation de dépendance directe), alors la tâche τ_i est appelée un *prédécesseur* de la tâche τ_j et τ_j est appelée un *successeur* de la tâche τ_i. Si une tâche n'a pas de prédécesseur (successeur), elle est appelée tâche *d'entrée* (tâche *de sortie*).

Pour fournir des garanties de synchronisation pour les systèmes temps réel durs, le WCET

doit être pris en compte lors de l'analyse et la conception du système. Le WCET d'une tâche est une estimation du temps d'exécution le plus long parmi tous les cas possibles. Le WCET d'une tâche dépend de la fréquence du processeur et des interférences dues à l'exécution parallèle de tâches sur le système multicœurs. Étant donné que la technique DVFS affecte le temps d'exécution, le WCET est donné en cycles d'exécution dans le pire des cas (Worst-Case Execution Cycle : WCEC), c'est-à-dire le nombre total de cycles CPU nécessaires dans le pire des cas, comme étudié dans [20, 21]. De plus, la date limite d'exécution est le temps avant lequel une tâche doit se terminer afin de satisfaire la contrainte temps réel du système. Dans cette thèse, nous considérons des applications de type trames, où l'exécution d'une application fonctionne de manière cyclique et où toutes les tâches de l'application doivent se terminer dans une période appelée trame [22, 23]. Les tâches sont libérées en début de trame et doivent avoir terminé leur exécution lors de la trame, ce qui détermine la date limite de l'application. Dans cette thèse, la période est considérée comme égale à la date limite globale à l'ensemble des tâches.

`Les différents schémas de DVFS`

Avec les systèmes multicœurs, la consommation d'énergie est devenue un facteur crucial, en particulier pour les systèmes avec un budget énergétique limité tels que les objets communicants alimentés par batterie ou à récupération d'énergie. En conséquence, des techniques de gestion adaptative ont été établies pour maximiser l'efficacité énergétique [24]. La gestion dynamique de la tension et de la fréquence (DVFS) est un mécanisme bien connu qui gère la consommation d'énergie dynamique en réduisant simultanément la tension et la fréquence d'alimentation du processeur, pendant l'exécution de la tâche [25, 24, 26]. La gestion de la fréquence et de la tension [27] peut être implémentée de plusieurs manières dans les plates-formes matérielles multicœurs. La première approche est la gestion globale de la fréquence et de la tension où un seul contrôleur est utilisé pour gérer la tension et la fréquence pour tous les cœurs simultanément. Par conséquent, tous les cœurs fonctionnent à la même fréquence. La deuxième approche est la gestion individuelle de la fréquence et de la tension où chaque cœur a son propre contrôleur et peut fonctionner à différents niveaux de tension et de fréquence, comme par exemple les processeurs Haswell-EP [28] et AMD Ryzen [29]. Compte tenu du coût matériel des contrôleurs, il existe des approches hybrides qui combinent une gestion globale et individuelle de la tension et de la fréquence, par exemple Ryzen 1700x [27]. Par conséquent, trois catégories de schémas DVFS peuvent exister en ce qui concerne l'attribution des tensions/fréquences aux tâches :

— `DVFS au niveau tâche:` L'attribution de fréquence est effectuée par tâche, c'est-à-dire que chaque tâche peut être exécutée à son propre niveau de fréquence. Les fréquences attribuées à chaque tâche sont indépendantes. Ce mécanisme DVFS est envisagé dans de nombreux travaux récents tels que [30, 31, 32, 33].

— `DVFS au niveau processeur:` L'attribution de fréquence est effectuée par processeur; toutes les tâches affectées à un même processeur sont exécutées avec la même fréquence.

Les fréquences attribuées aux processeurs sont indépendantes.

— `DVFS au niveau système` La même fréquence est affectée à tous les processeurs de la plate-forme, et la fréquence est modifiée en même temps pour tous les processeurs. Un tel schéma est appliqué par exemple dans [34, 35]

`Modèle de consommation de puissance et d'énergie` Dans une plate-forme multicœur compatible DVFS, la puissance consommée se compose généralement de deux parties : la puissance dynamique qui est causée par l'activité lors de l'exécution et la puissance statique due au courant de fuite [25]. La puissance statique est constante et indépendante de la tension et de la fréquence du système. Étant donné que dans cette thèse nous utilisons la gestion dynamique de la tension et de la fréquence pour gérer la consommation d'énergie, nous nous concentrons donc sur la réduction de la puissance dynamique, comme in [36] [37]. Nous adoptons un modèle de puissance largement utilisé, comme dans [25, 26, 33] :

$$P(f) = P_s + P_d = P_s + \hbar(P_{ind} + P_{dep}) \tag{1}$$

P_s est la puissance statique qui est consommée pour maintenir le fonctionnement de base du circuit et qui peut être annulée lors de la mise hors tension du circuit. La puissance dynamique P_d comprend deux parties : 1) une composante de puissance consommée indépendante de la fréquence P_{ind} qui est causée par des modules périphériques comme la mémoire et des périphériques externes lorsque le système est en mode actif [26], qui peut être supprimé en mettant le système en mode veille, et 2) la puissance consommée dynamique du processeur et de tous les autres périphériques dépendant de la fréquence, P_{dep}. $\hbar$ est un facteur qui décrit les modes du système, c'est-à-dire que lorsque $\hbar = 1$, le système est en mode actif et la puissance dynamique est effective, sinon lorsque $\hbar = 0$, le système est en veille mode et aucune puissance dynamique n'apparait. P_{dep} peut être exprimé comme $P_{dep} = C_{eff} f^m$ où C_{eff} est la capacité de commutation effective et m est l'exposant de puissance dynamique, normalement non inférieur à 2. C_{eff} et m sont des constantes qui dépendent des caractéristiques du processeur/de la technologie. Dans cette thèse, nous nous concentrons principalement sur la puissance consommée dynamique dépendant de la fréquence, où la puissance consommée totale du système est dominée par la puissance consommée par les processeurs pour exécuter des tâches. Plusieurs schémas DVFS présentés ci-dessus sont utilisés pour ajuster la fréquence/tension afin de minimiser la puissance consommée totale du système.

`Modèles de tolérance aux fautes et de fiabilité` L'exécution correcte d'une application peut être menacée par plusieurs sources, telles que les rayonnements [38] et les interférences électromagnétiques. Une *faute* est un défaut physique ou une imperfection qui se produit dans un composant matériel ou logiciel [39]. Une faute peut entraîner un écart par rapport à l'exactitude ou à la précision du calcul, qui devient alors une *erreur*. Une *défaillance* est un écart par rap-

port à la valeur réelle et attendue. Un système est dit défaillant si le service qu'il fournit à l'utilisateur s'écarte de la conformité à la spécification pendant une période de temps donnée [40]. En général, les fautes sont les sources d'erreurs et les erreurs les sources de défaillance [39]. Les fautes matérielles peuvent généralement être classés en deux types : *fautes permanentes* et *fautes transitoires* en fonction de la durée de la faute. Pendant la durée de vie normale d'un système, les fautes transitoires se produisent plus fréquemment que les fautes permanentes, et sont donc considérées comme les principales menaces pour la bonne exécution des applications [32, 41]. En raison de la réduction de la taille des transistors, les systèmes sont devenus plus sensibles aux fautes transitoires [42]. Dans cette thèse, nous considérons les fautes transitoires. Une faute transitoire reste active pendant une courte période. Les causes des fautes transitoires sont principalement environnementales, telles que les particules (un impact de neutrons de rayons cosmiques ou de particules α), les décharges électrostatiques, les baisses de puissance électrique, la surchauffe ou les chocs mécaniques. Pour les systèmes compatibles DVFS, un niveau de tension/fréquence faible est plus susceptible de provoquer une faute transitoire. Le modèle de fautes transitoires suit une distribution de Poisson avec un taux de fautes moyen λ [36] où le taux de faute est le nombre de fautes attendu par unité de temps [39]. Pour les systèmes compatibles DVFS avec L paires de niveaux de tension/fréquence $\{(v_1, f_1), ..., (v_L, f_L)\}$, le taux de faute à la fréquence f_l suit une distribution exponentielle :

$$\lambda(f_l) = \lambda_0 \times 10^{d\frac{f_{max}-f_l}{f_{max}-f_{min}}} \tag{2}$$

où λ_0 est le taux de faute moyen à la fréquence maximale, d (appelé *facteur de sensibilité*) est une constante, utilisée pour mesurer la sensibilité du taux de faute à la gestion dynamique de la tension/fréquence. f_{max} et f_{min} sont respectivement la fréquence maximale et minimale dans les L niveaux de tension/fréquence. La *fiabilité* de l'exécution d'une tâche est la probabilité d'exécuter la tâche sans faute. Lors de l'exécution d'une application, selon la *loi de défaillance exponentielle* [39], la fiabilité d'une exécution varie de façon exponentielle en fonction de son temps d'exécution comme

$$R(f_l) = e^{-\lambda(f_l)\times t} \tag{3}$$

où t est la durée d'exécution et qui est inversement proportionnelle à la fréquence. L'exécution d'une application est d'autant plus fiable que la fréquence augmente.

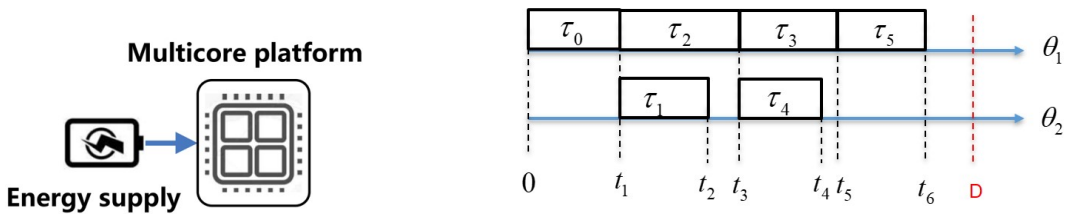
En pratique, il est impossible de construire un système parfait sans apparition de fautes, en particulier avec la diminution de la technologique [43]. Pour améliorer la fiabilité d'un système, plusieurs approches utilisent une fréquence élevée pour obtenir une grande fiabilité pour l'exécution de l'application. Mais avec l'augmentation de la complexité d'un système, la fiabilité du système diminue considérablement même en appliquant la fréquence la plus élevée pour

exécuter des tâches. Par exemple, en supposant que la fiabilité d'une tâche est très élevée, par exemple 99,999% à fréquence maximale, lorsque le système comporte 10 tâches, la fiabilité du système après exécution de ces 10 tâches est de 99,99% ; lorsque le système a 20 tâches, cette valeur diminue à 99,98%, et lorsque le système a 100 tâches, la valeur est de 99,9%. Pour un système avec des exigences de fiabilité élevées, l'exigence de fiabilité ne peut pas être satisfaite en utilisant uniquement la haute fréquence. La *tolérance aux fautes* est la capacité d'un système à continuer à exécuter ses fonctions prévues en présence de fautes.

Il existe différentes approches pour implémenter la tolérance aux fautes. Dans cette thèse, nous considérons une technique de réplication active pour implémenter la tolérance aux fautes. La réplication de tâches [24, 32, 37, 44, 45, 46, 47] est une technique largement adoptée pour tolérer les fautes transitoires. Elle est basée sur la redondance spatiale : plusieurs copies d'une tâche sont exécutées sur différents cœurs. Il existe deux approches principales pour assurer la réplication. Avec la *réplication passive*, chaque tâche est répliquée plusieurs fois et ces répliques sont exécutées sur différents processeurs [24, 32, 37, 45]. Ce faisant, il est peu probable que toutes les répliques d'une tâche échouent à l'exécution. Un vote majoritaire est opérée à la fin de l'exécution afin de déterminer la valeur du résultat. Avec la *réplication passive* [46, 47], chaque tâche a une copie principale et une copie de sauvegarde. La copie de sauvegarde n'est activée que lorsque la copie principale échoue dans son exécution. Un dispositif de détection d'erreur permet de savoir à la fin de l'exécution si la copie s'est exécutée correctement ou non.

`Ordonnancement temps-réel de tâches dans les systèmes multicœurs` Dans un système multicœur, étant donné une application composée de plusieurs tâches, l'ordonnancement temps-réel de tâches s'attache à résoudre en fait deux problèmes : 1) *l'allocation de tâches*, qui décide de l'allocation tâche-cœur (sur que cœur est exécuté quelle tâche) ; 2) *l'ordonnancement des tâches* proprement dit, qui est l'affectation temporelle des tâches (quand une tâche commence à s'exécuter). Dans cette thèse, nous effectuons l'ordonnancement des tâches au moment de la compilation, c'est ce qu'on appelle un ordonnancement statique. Les systèmes temps réel, tels que la robotique, les applications automobiles et les systèmes de contrôle de vol, sont des systèmes informatiques qui doivent réagir dans des délais précis aux événements de l'environnement [22]. Comme nous nous concentrons sur les systèmes temps réel durs, l'exactitude de la sortie du système dépend non seulement du résultat fonctionnel du calcul, mais également du moment auquel les résultats sont produits [32]. La figure 1 illustre un ordonnancement de tâches sur une plate-forme multicœur. En supposant qu'il y a deux cœurs, les six tâches et que la date limite globale est D, l'allocation tâche-cœur est obtenue en affectant les tâches 0, 2, 3, 5 au cœur 1 et les tâches 1,4 au cœur 2 . Les dates de début de chaque tâche sont $\{0, t_1, t_3, t_5\}$. La contrainte de temps est satisfaite dans cet ordonnancement car l'heure de fin d'exécution de la tâche la plus tardive ne dépasse pas D.

La plupart des cœurs modernes prennent en charge une large gamme de tensions et de



(a) Multicore platform (b) Task mapping with 6 tasks and 2 cores

Figure 1: Ordonnancement de tâches sur système multicœur.

fréquences [22] gérée par DVFS. La gestion dynamique de la tension/fréquence a un impact important sur la consommation d'énergie, la fiabilité et la vitesse d'exécution. Malheureusement, les effets/conséquences sont généralement contradictoires. Par exemple, lorsqu'une fréquence plus basse est utilisée, le coût énergétique peut être diminué, mais cela a un impact négatif sur la fiabilité et entraîne également des temps d'exécution plus longs. Généralement avec la diminution de la consommation d'énergie, la qualité d'exécution de l'application se dégrade. Avec l'augmentation de la taille des applications, l'allocation et l'ordonnancement d'un ensemble de tâches à un groupe de cœurs sous plusieurs contraintes, telles que le coût énergétique, les performances temporelles, la fiabilité, sont devenues un défi majeur dans les architectures temps réel multicœurs modernes. Par conséquent, il est intéressant d'étudier des algorithmes d'ordonnancement de tâches appropriés qui permettent d'optimiser le compromis entre fiabilité, vitesse d'exécution et efficacité énergétique.

Motivations et contributions

La fiabilité et la consommation d'énergie sont devenues deux préoccupations majeures dans les systèmes informatiques modernes. Bien que la tolérance aux fautes et la gestion de l'énergie aient été largement étudiées, la cogestion de la fiabilité du système et de l'efficacité énergétique n'a été abordée que récemment. Les approches existantes pour les plates-formes multicœurs considèrent généralement que l'allocation des tâches est donnée à l'avance ou qu'il est fixe, lors de l'exploration de la marge de temps disponible pour l'ordonnancement des tâches. Le couplage complexe entre les variables d'optimisation de l'ordonnancement des tâches et l'affectation de la tension et de la fréquence empêche les algorithmes d'atteindre la solution optimale. Par conséquent, des méthodes sous-optimales sont généralement proposées sur la base de 1) l'approximation/relaxation du problème et 2) l'utilisation d'heuristiques. Par rapport aux approches existantes, la thèse se concentre sur la conception de nouvelles méthodologies pour résoudre efficacement le problème de l'exécution de tâches sur des plates-formes multicœurs en abordant conjointement ces facteurs.

Dans cette thèse, nous nous intéressons à combiner le DVFS et des techniques de tolérance

aux fautes pour décider de l'exécution de l'application sur des architectures multicœurs et exploiter l'impact de trois schémas de DVFS représentatifs des schémas DVFS existant dans les plateformes multicœurs récentes. Tout d'abord, nous concevons des méthodologies pour l'ordonnancement des tâches sur des plates-formes multicœurs qui fournissent des solutions optimales pour les modèles de tâches indépendants et dépendants et pour les trois schémas de DVFS. Ensuite, pour faire face au temps de calcul élevé nécessaire pour obtenir des solutions optimales, nous proposons un ensemble d'heuristiques qui fournissent des solutions quasi optimales avec un temps de calcul réduit. Lors de l'analyse expérimentale, nous avons utilisé des graphes de tâches générés aléatoirement et des graphes de tâches d'applications réelles pour évaluer le comportement des approches proposées.

Pour mieux comprendre les contributions de cette thèse, nous posons les questions suivantes :

1. Que comprend l'ordonnancement des tâches dans les problèmes étudiés dans cette thèse ?

Nous avons étudié deux groupes de problèmes d'ordonnancement des tâches : le premier groupe concerne les tâches indépendantes. L'objectif des problèmes étudiés est de minimiser la consommation d'énergie sous des contraintes de temps réel et de fiabilité, en déterminant simultanément l'allocation des tâches, la duplication des tâches et l'affectation des fréquences. Le deuxième groupe concerne les tâches dépendantes ayant le même objectif sous des contraintes de temps réel, de fiabilité et de dépendance des tâches en déterminant simultanément l'allocation des tâches, l'ordonnancement des tâches (date de début d'exécution de chaque tâche), la duplication des tâches et l'affectation des fréquences.

2. Comment obtenir les solutions optimales pour les problèmes d'ordonnancement des tâches étudiés sachant qu'ils sont connus comme étant des problèmes NP-difficiles ?

En général, il est compliqué d'obtenir des solutions optimales pour les problèmes d'ordonnancement de tâches sur des plates-formes multicœurs car elles sont NP-difficiles. Les problèmes étudiés sont d'abord formulés sous forme de programmation non linéaire mixte en nombres entiers (MINLP), puis une méthode de remplacement de variables est utilisée pour transformer de manière équivalente les problèmes MINLP sous forme de problèmes en programmation linéaire mixte en nombres entiers (MILP) et qui peuvent être résolus avec des outils de type solveur, tels que Gurobi ou Cplex. Cette partie est présentée au Chapitre 3.

3. Quelles sont les idées centrales des approches heuristiques proposées pour les problèmes étudiés ?

Afin de faire face à des temps de calcul longs pour obtenir les solutions optimales, un ensemble d'heuristiques est proposé. Le principe des heuristiques proposées consiste en deux phases : une phase d'élagage et une phase d'ordonnancement. La phase d'élagage ne maintient que les config-

urations de tâches qui satisfont les contraintes de fiabilité. Ensuite, la phase d'ordonnancement minimise l'énergie consommée sous contraintes de temps réel pour les tâches indépendantes et sous contraintes de temps réel et de précédence pour les tâches dépendantes. La phase d'élagage exclut les solutions inutiles dans l'espace des solutions et la phase d'ordonnancement utilise les solutions restantes pour rechercher les solutions quasi optimales aux les problèmes étudiés.

Dans cette thèse, nous avons considéré et évalué trois niveaux DVFS comme expliqué dans la section 3.2 . Nous avons proposé une série de méthodologies d'ordonnancement des tâches qui peuvent être classées en deux groupes : 1) les algorithmes optimaux qui fournissent les solutions optimales, et 2) les algorithmes heuristiques qui fournissent des solutions quasi optimales, mais nécessitant beaucoup moins de calculs (voir figure 2). La figure 3 décrit l'idée générale de l'approche proposée RAFTM (Reliability-Aware Fault-Tolerant Task Mapping) basée sur la technique de duplication partielle pour répondre à la tolérance aux fautes. Dans l'approche proposée, nous fixons à deux le nombre maximal de répliques pour chaque tâche et sélectionnons une partie de l'ensemble de tâches pour effectuer la duplication.

Tout d'abord, les algorithmes optimaux pour les modèles de tâches indépendants et dépendants sous trois schémas de DVFS sont étudiés au chapitre 3, en utilisant une méthode *de remplacement de variables* pour transformer de manière équivalente les problèmes MINLP originaux dans les formes MILP, puisque les problèmes MILP peuvent être résolus à l'aide de solveurs d'optimisation. Cependant, le temps pour obtenir une solution avec de telles approches optimales devient rapidement trop important à moins que l'application ait très peu de tâches. Nous étendons donc l'approche proposée à des algorithmes heuristiques au Chapitre 4. En nous concentrant sur chaque schéma de DVFS, nous proposons les heuristiques correspondantes pour les modèles de tâches indépendantes et dépendantes. Enfin, nous menons un grand nombre d'expériences pour des graphes de tâches générés aléatoirement et pour des graphes de tâches d'applications réelles afin d'évaluer les approches proposées. Pour des solutions optimales, nous comparons nos approches avec deux autres approches de l'état d l'art. Les résultats expérimentaux montrent que les approches proposées permettent de gagner en énergie consommée et en capacité d'obtenir des solutions réalisables. Pour les approches basées sur des heuristiques, nous comparons d'abord les heuristiques proposées à des solutions optimales pour analyser l'écart de performance. De plus, des expériences sont réalisées pour évaluer les algorithmes heuristiques proposés par rapport à deux autres algorithmes heuristiques de l'état d l'art. Nos algorithmes heuristiques donnent des résultats très proches des algorithmes optimaux tout en ayant une complexité de calcul faible, et surpassent les heuristiques de l'état de l'art en matière d'énergie consommée et de capacité d'obtention de solutions réalisables.

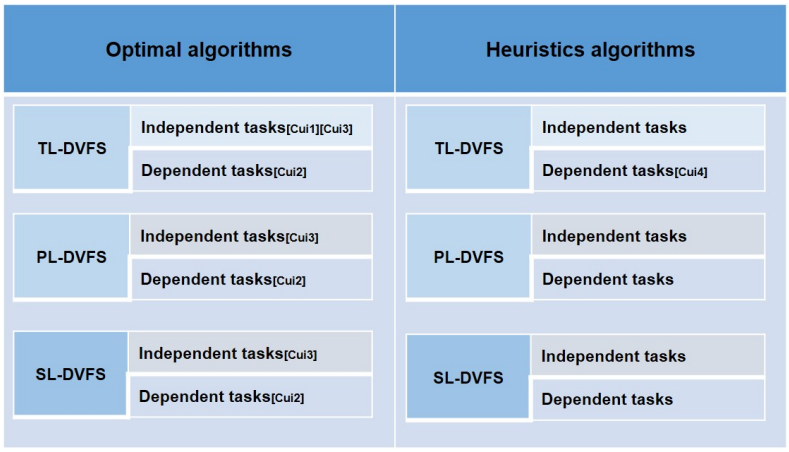


Figure 2: Approches proposées d'ordonnancement des tâches pour différents niveaux de DVFS et différents modèles de tâches.

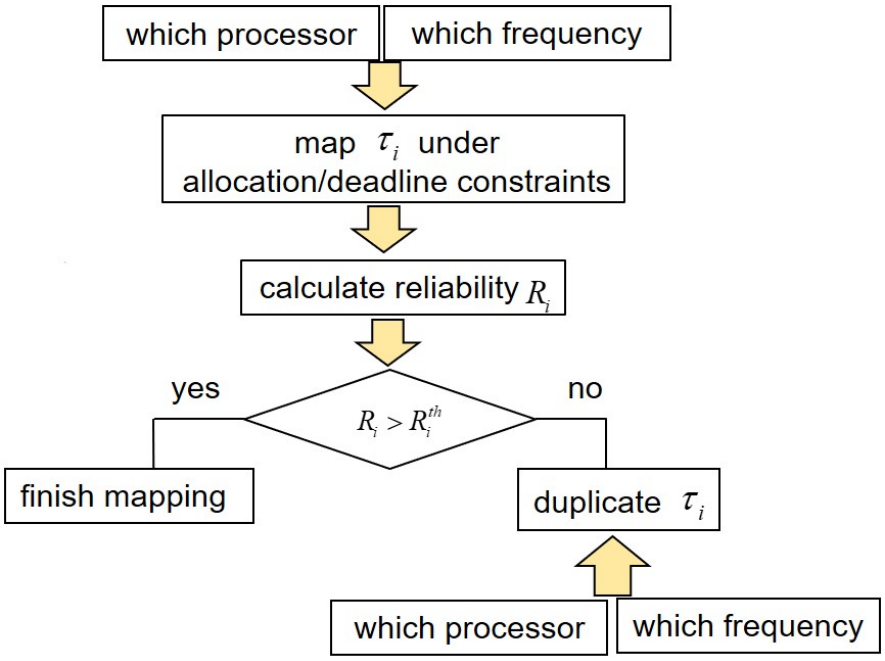


Figure 3: Présentation générale de l'approche proposée d'ordonnancement des tâches tolérant aux fautes et tenant compte de la fiabilité

Organisation du document

Le mémoire de thèse est organisé de la manière suivante:

— Dans le chapitre 1, nous introduisons brièvement les concepts de base qui seront utilisés dans le reste du document. Dans un premier temps, nous présentons le modèle de plates-formes multicœurs. Ensuite, nous décrivons brièvement les deux modèles de tâches qui sont utilisés dans les problèmes étudiés. Comme l'objectif de la thèse est de minimiser la consommation d'énergie, nous introduisons le modèle de puissance/énergie et les trois schémas de DVFS utilisés. De plus, la thèse portant sur la fiabilité, nous présentons les principales sources de fautes et les modèles de fautes. Enfin, nous introduisons plusieurs techniques de tolérance aux fautes en nous concentrant sur les approches de réplication de tâches.

— Dans le chapitre 2, nous présentons les travaux de l'état de l'art liés à notre sujet. Trois catégories de problèmes d'ordonnancement des tâches sont introduites en fonction de l'objectif du problème étudié. La première catégorie vise à minimiser la consommation d'énergie. Plusieurs approches d'ordonnancement des tâches sans et avec tolérance aux fautes sont brièvement présentées. La deuxième catégorie vise la maximisation de la fiabilité. Ensuite, la troisième catégorie se concentre sur la minimisation de la durée d'exécution de l'application. Enfin, nous concluons avec les limites des approches d'ordonnancement des tâches de l'état de l'art.

— Dans le chapitre 3, nous présentons d'abord le problème étudié dans cette thèse pour les tâches indépendantes et dépendantes sous trois schémas de DVFS en tant que problèmes MINLP. Ensuite, nous décrivons comment transformer de manière équivalente les formulations MINLP en formulations MILP. Pour l'évaluation expérimentale, les solutions optimales sont obtenues à l'aide du solveur Gurobi. Les résultats montrent que les approches optimales proposées permettent d'obtenir une consommation d'énergie plus faible et trouvent des solutions lorsque d'autres approches de l'état de l'art ne parviennent pas à obtenir des solutions pour le problème traité.

— Dans le chapitre 4, nous considérons les mêmes problèmes étudiés qu'au chapitre 3 et nous proposons des heuristiques pour obtenir des solutions quasi-optimales avec une complexité de calcul raisonnable. Des résultats expérimentaux utilisant divers graphes de tâches générées aléatoirement ainsi que provenant d'applications réelles sont présentés pour évaluer les heuristiques proposées en comparaison des solutions optimales et d'heuristiques de l'état de l'art.

— Dans le chapitre conclusion, nous concluons notre travail de thèse par un résumé des travaux effectués et nous proposons des perspectives à ces travaux.

INTRODUCTION

Context

The safety-critical domain industries usually consist of systems where guarantees must be provided on safety and reliability for the critical applications, implying both high fault tolerance and hard real-time constraints [1]. For example, avionics systems consist of applications with high Design Assurance Levels (DAL) [2] operating in high altitudes exposed to radiation. Space systems consist of navigation control applications operating in outer space with extreme particle and high-energy electromagnetic radiation. Automotive systems have applications, among others, in the same wheel sensor for stability control and for the acceleration regulation while they suffer from alpha particles, high temperature peaks and electromagnetic interferences [3]. These natural and technical stimuli are the source of faults that impact the system functionality [4]. Furthermore, within last thirty years, the code size of avionics, space and automotive applications has significantly increased [5]. These systems face exponential growth in performance requirements, whereas future automotive and aerospace applications will require higher performance computing resources [1].

To deal with the increasing performance demands, the consumer market has shifted towards multicore architectures, due to power consumption and heat dissipation limits of single processors [1]. Generally speaking, multicores provide a Space, Weight and Power reductions (SWaP) and massive computing capabilities compared with single core processors, while they can integrate several applications on the same platform [6]. However, two main scientific challenges raised by the use of multicores.

The first challenge is high energy consumption which has become one of the biggest obstacles to develop green and high performance computing systems, especially for systems with limited energy budget, such as battery powered or energy-harvesting Internet of Things (IoT) devices. Smartphones use heterogeneous multicore architectures, such as big.LITTLE [7], which consists of performance-optimized big cores and energy-optimized little cores with a single Instruction Set Architecture (ISA) [8]. Embedded boards for Internet of Thing (IoT), such as Raspberry Pi, Odroid, Edison, Jetson, and Artik also provide multiple cores [9, 10, 11]. The use of multiple cores supports efficiently the IoT services, but the increase in the number of cores puts pressure on the energy resource of the device, since the power and energy consumptions are increased [48]. Hence, multicore platforms have been enhanced with the capability of scaling their voltage and frequency (Dynamic Voltage and Frequency Scaling - DVFS) during execution to balance system

performance and energy savings.

The second challenge is that the multicore system itself is susceptible to faults due to the nature of electronic systems. Combined with the reduction of the transistor size and the technology, multicore systems are becoming more and more sensible to the operating conditions and to the environmental impact [12]. In electronic systems, the variation on the threshold voltage depends on the transistor width, whereas voids or small cracks in the wiring lead to close or open source problems. The current or voltage activity and hot spots are inevitable during the system operation, but they cause electromigration, Bias Temperature Instability (BTI) and crosstalk, which are sources of faults. To improve the multicore reliability, either radiation-hardened processors are used or the system is replicated [13]. The former solution develops systems with limited computation capabilities and it requires a difficult-to-find design expertise, which combines digital and analogue electronics with semiconductor physics. The latter solution has high cost and energy consumption. To reduce the cost while providing reliability, the resources replication and the system oversizing has to be avoided, whenever possible.

Taking the above two challenges into consideration, in order to fully exploit the features of multicore systems, while obtaining both reliable and energy efficient application execution meeting system specifications, methods are required to decide the efficient execution of the tasks on multicores with scalable operating features. The way that tasks are executed on a platform is decided by several factors. The first factor is the *task mapping*, which refers to both the task allocation (on which processor each task is executed) and the task scheduling (at which time each task starts its execution). The second factor is the decision of the *voltage and frequency assignment* of the processor when it runs a specific task, which determines the execution time of the task. For critical applications, Worst Case Execution Time (WCET) bounds are used since they are required for guarantees regarding reliability and correct functional behaviour.

Motivation and Goals

Reliability and energy consumption have become two major concerns in modern computing systems. Although both fault tolerance and energy management have been extensively (but often independently) studied, the co-management of system reliability and energy efficiency has been addressed only recently. The existing approaches on multicore platforms usually consider that the task allocation is upfront given or it is fixed, when exploring the available time slack for task scheduling. The complex coupling among optimization variables of task mapping and voltage and frequency assignment prohibits the algorithms to achieve the optimal solution. Therefore, sub-optimal methods are usually proposed based on 1) problem approximation/relaxation, and 2) heuristics. Compared with the existing approaches, the thesis focuses on designing novel methodologies to efficiently solve the problem of task execution on multicore platforms by jointly

addressing all aforementioned factors.

In this thesis, we are interested in combining DVFS and fault tolerance techniques to decide the execution of the application on multicore architectures and exploit the impact of three DVFS schemes, which are representative of DVFS schemes existing in recent platforms. First, we design methodologies for task mapping on multicore platforms that provide optimal solutions for both independent and dependent task models under three DVFS schemes. Then, to cope with high computation time required to obtain optimal solutions, we propose a set of heuristics that provide near-optimal solutions with reduced computation time, leading to scalable approaches. Overall, we used synthetic and real-world task graphs to evaluate the behavior of the proposed approaches during the experimental analysis.

To better understand the contributions of this thesis, we pose the questions:

1. What does the task mapping include in the studied problems in this thesis?

We studied two groups of task mapping problems: the first group is for independent tasks. The objective of studied problems is to minimize energy consumption under real-time, reliability requirement constraints by simultaneously determining task allocation, task duplication and frequency assignment. The second group is for dependent tasks with same objective under real-time, reliability requirement and task dependency constraints by simultaneously determining task allocation, task scheduling (execution start time of each task), task duplication and frequency assignment.

2. How to obtain the optimal solutions for studied task mapping problems since they are known as NP-hard problems?

In general, it is complicated to obtain optimal solutions for task mapping problems on multicore platforms since they are NP-hard. The studied problems are firstly formulated as Mixed-Integer-Nonlinear-Programming (MINLP) forms, then a variable replacement method is used to safely and equivalently transfer the MINLP problems into Mixed-Integer-Linear-Programming (MILP) forms which can be solved with solver tools, such as Gurobi, Cplex or Matlab. This part is presented in Chapter 3.

3. What are the core ideas for the proposed heuristics approaches for the studied problems?

To cope with long computation time to obtain the optimal solutions, a set of heuristics is proposed. The proposed heuristics consist of two phases: a pruning phase and a mapping phase. First, a pruning phase maintains only the task configurations that satisfy reliability constraints. Then, a mapping phase minimizes the total energy consumption under real-time constraints for independent tasks and under real-time and precedence constraints for dependent tasks. The pruning phase excludes the unnecessary solutions in the solution space and the mapping phase uses the remaining solutions of the pruning phase to search for the near-optimal solutions for the studied problems.

Thesis structure

This thesis is organized as follows:

— In Chapter 1, we briefly introduce the background information of the basic concepts, which will be used in the rest of this thesis. Firstly, we present the model for the multicore platforms. Then, we briefly describe the two task models that are used in the studied problems. As the goal of the thesis is to minimize energy consumption, we introduce the power/energy model and the three DVFS schemes used as energy management method. Furthermore, since the thesis focuses on reliability, we present the main sources of faults and the fault models. Last, we summarise several fault tolerance techniques and we focus on task replication approaches.

— In Chapter 2, we present the State-of-Art (SoA) works related to our topic. Three categories of task mapping problems are introduced based on the objective of the studied problem. The first category aims at minimizing energy consumption. Several task mapping approaches without and with fault tolerance are briefly presented. The second category aims at reliability maximization. Then, the third category focuses on minimizing the schedule length of the application execution. Finally, we conclude with the limitations of SoA task mapping approaches.

— In Chapter 3, we firstly present the problem studied in this thesis for both independent and dependent tasks under three DVFS schemes as MINLP problems. Then, we describe how to safely and equivalently transfer the MINLP forms into MILP forms. For the experimental evaluation, the optimal solutions are obtained using Gurobi solver tool. Results show that the proposed optimal approaches achieve better energy consumption and find solutions, when other SoA approaches fail to obtain solutions for the studied problem.

— In Chapter 4, we consider the same studied problems as in Chapter 3 and we propose heuristics to obtain near-optimal solutions with a reasonable computational complexity. Experimental results using various task graphs from both synthetic and real-world applications are presented to evaluate the proposed heuristics with optimal solutions and SoA heuristics.

— In Chapter 5, we conclude our thesis with an overview of the presented work and summarize future perspectives of our work.

Chapter 1

Background

We start this thesis by providing the required background regarding the main concepts of energy-quality-time fault tolerant task mapping on multicore architectures, which is the topic of this thesis. Initially, we define the architecture and task models. The architecture is multicore platforms described in Section 1.1. Section 1.2 presents the notations of two main types of task models which are commonly studied in task mapping problems. Dynamic-Voltage-Frequency-Scaling (DVFS) scheme is an important technique to jointly manage energy consumption, timeliness and reliability of task execution. Three DVFS schemes are introduced in Section 1.3. Then, we provide the power/energy consumption model in Section 1.4. Reliability is one metric to measure the quality of task execution, as far as reliable execution is necessary. We present the main origins of faults and how to use mathematical methods to build reliability models. Then we introduce main fault tolerance techniques in Section 1.5. Finally, in Section 1.6, we focus on the real-time task mapping systems studied in this thesis.

1.1 Multicore Architecture

To meet the rapidly increasing computation needs, low resource consumption, as well as ensuring the high parallelism of multiple application executions, multicore systems are becoming a promising platform for real-time embedded systems. In recent chip market, many chip manufactures, e.g. AMD and Intel, have been releasing multicore chips with increasing number of cores, e.g Intel Xeon Series. A multicore architecture consists of two or more separate processing units (cores) on a single integrated circuit. Each core executes CPU instructions at the same time, increasing overall speed for parallel computing.

Multicore architectures can be characterised as homogeneous and heterogeneous systems. Homogeneous multicore systems include identical cores like some commonly used Multiprocessor system-on-chip (MPSoC) built with ARM Cortex cores [14]. Heterogeneous multicore systems combine different cores, e.g. ARM big.LITTLE architecture with big processors with higher performance but power energy hungry, such as A-15, and LITTLE processors with lower performance but better energy efficiency, such as A-7 [15]. In this thesis, to ease the approach but without loss of generality, we focus on homogeneous platforms where all processors share a set of frequencies [16] e.g. the Arm quad-core Cortex-A53 system.

1.2 Task Model

As introduced in [17], a task refers to a set of coherent activities that are performed in order to achieve a goal in a given domain. An application is generally represented by a set of tasks. According to the relations between tasks, two categories of task models are considered in this thesis, i.e., independent tasks and dependent tasks. When, in certain applications, computational activities can be executed in arbitrary order, these can be considered as *independent* tasks. Otherwise, tasks are *dependent*: they have to respect the precedence (or dependency) relations, i.e., whether a task relies on inputs provided by other tasks. Such dependency relations between tasks are used to build directed acyclic graphs (DAG) to describe the application [18]. DAGs are employed for the representation of the computation, communication and dependencies of the application tasks. Figure 1.1 illustrates a simple DAG graph with 6 tasks.

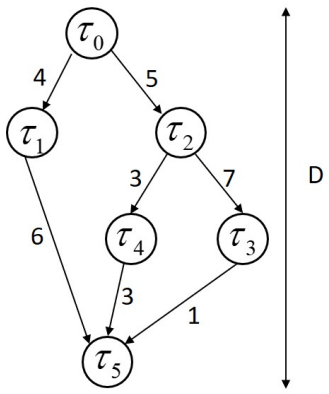


Figure 1.1: A DAG task graph with a global deadline D.

A DAG graph G consists of a pair $G = \{\boldsymbol{V}, \boldsymbol{E}\}$ where $\boldsymbol{V}$ is the set of vertices and $\boldsymbol{E}$ is the set of direct edges which denotes the data communication among tasks [19]. A vertex denotes a task and an edge presents a dependency relationship. For two tasks τ_i and τ_j, if $(\tau_i, \tau_j) \in \boldsymbol{E}$, task τ_j depends on task τ_i and can start its execution only after τ_i finishes its execution. Each vertex is characterised by the Worst-Case-Execution Time (WCET) of the corresponding task. The weights of edges depict the communication cost (in amount of data or in time), when tasks are mapped on different cores. In this thesis, we assume that the communication cost (in time) is included in the WCET of the tasks. If a task τ_j is dependent on task τ_i (i.e., task τ_i and task τ_j have direct dependency relation), then task τ_i is called a *predecessor* of task τ_j and τ_j is called a *successor* of task τ_i. If a task has no predecessor (successor), it is called *entry* task (*exit* task). Note that, independent task sets do not have dependency relations.

To provide timing guarantees for hard real-time systems, the WCET must be considered during system analysis and design. The WCET of a job of a task is an estimation of the longest

execution time among all possible cases. The WCET of a task depends on the processor frequency and the interferences occuring due to the parallel execution of tasks in multicores. Since DVFS affects execution time, the WCET is given in Worst-Case Execution Cycles (WCEC), i.e., the total number of CPU cycles needed in the worst-case, as studied in [20, 21].

Furthermore, deadline is the time before which a task must finish in order to guarantee timing results for real-time applications. For instance, all six tasks should finish before the global deadline D in Figure 1.1. Depending on the consequences when the deadline is not met [22], the deadline is characterised as: 1) *hard* if the results produced after the deadline can cause catastrophic consequences; 2) *firm* if the results produced after the deadline are useless, but do not cause catastrophic consequences; and 3) *soft* if the results produced after the deadline can be used, but with a degradation on performance. In this thesis, hard deadlines are considered.

In many embedded real time systems, the execution of application tasks operates on a cyclic basis. Therefore, a task may release multiple jobs in a regular or irregular way. Period is the minimum time interval between two successive jobs of a task [19]. Therefore, a task set can be characterised by:

1. `Periodic tasks` where jobs are released in a regular way. A periodic task τ_i is denoted by following tuples $\tau_i = \{\phi_i, p_i, et_i, d_i\}$, where ϕ_i is the phase of the task; p_i is the period of the task, i.e., the time interval between the release time of two consecutive jobs, et_i is the execution time of the task and d_i is is the relative deadline of the task. Fig 1.2 illustrates a periodic task with $et_i = 2$, $d_i = 3$ and $p_i = 5$. The phase is considered zero. The first job j_0 of the task is released at time zero, it executes for 2 time units, then the next job j_1 is released at time 5, etc. Jobs are released at $t = 5k$ where $k = 0, 1, ..., n$. The hyper period of the task set is the time after which the pattern of job release times is repeated. The hyper period (H) of a set of periodic tasks is defined as the least common multiple (lcm) of periods of all n tasks in that set [49], i.e., $H = lcm(p_1, ..., p_n)$.

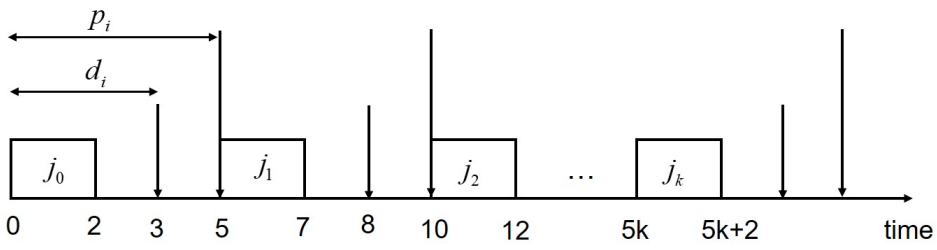


Figure 1.2: Time parameters for a periodic task.

2. `Aperiodic tasks` In aperiodic tasks, jobs are released at randomly time intervals. Aperiodic tasks usually have soft deadlines or no deadlines. When tasks are finished after their deadlines or without deadlines, the results of their execution may be used but with a performance

degradation [22]. If a task only generate one job, the period is considered as infinite.

3. `Sporadic tasks` Sporadic tasks behave similarly to periodic tasks for the repetition and similarly to aperiodic tasks for the randomness of job releases. For sporadic tasks, the jobs are generated separately by at least "period" time units.

In this thesis, we consider the frame-based applications, where the execution of an application operates on a cyclic basis and all application tasks should finish within a period called a frame [22, 23]. Frame-based applications are used in systems that use timeline scheduling [22] and pipelined scheduling [50]. All tasks are released in the beginning of the frame and must complete the execution within the frame, which determines the deadline of the application. In this thesis, the period is considered as equal to the global deadline for frame-based tasks.

1.3 DVFS Schemes

On multicore systems, the energy consumption has become a crucial factor, especially for systems with a limited energy budget such as battery powered or energy-harvesting Internet of Things(IoT) devices. As a result, adaptive management techniques have been established to maximize energy efficiency [24]. Dynamic Voltage and Frequency Scaling (DVFS) is a well-known mechanism that manages dynamic energy consumption by simultaneously scaling down the processor supply voltage and frequency, during task execution [25, 24, 26]. Frequency and voltage scaling [27] can be implemented in several ways in hardware multicore platforms. The first approach is global frequency and voltage scaling where a single voltage controller is used to scale voltage and frequency for all cores simultaneously. Therefore all cores run at a same frequency. The second approach is individual frequency and voltage scaling where each core has its own voltage controller and can run at different voltage and frequency level, e.g. in Haswell-EP [28] and AMD Ryzen [29] processors. Considering the hardware cost of the controllers, hybrid approaches exist that combine global and individual voltage and frequency scaling, e.g. Ryzen 1700x [27]. Therefore, three categories of DVFS schemes can exist regarding the frequency assignment to tasks:

1. `Task-level(TL) DVFS:` The frequency assignment is performed per task, i.e., each task can be executed at its own frequency level. The frequencies assigned to each task are independent. This DVFS mechanism is considered for platforms in many recent works such as [30, 31, 32, 33].

2. `Processor-level(PL) DVFS:` The frequency assignment is performed per processor, thus all tasks, mapped on the same processor, are executed with the same frequency. The frequencies assigned to processors are independent. Variants of PL-DVFS scheme are applied in existing hardware platforms, e.g., clock frequency is controlled per core in Intel-Xeon E5620, and used in the literature, e.g., PL-DVFS is applied in [51] for example.

3. `System-level(SL) DVFS:` The same frequency is assigned to all processors of the platform, and the frequency is modified at the same time for all processors. Such a SL-DVFS is applied for example in the dependent platform with runtime adjusting of [34, 35]

1.4 Power and Energy Consumption Model

Similar to [24, 26, 31], we assume that the relationship of voltage and frequency is almost linear. Therefore, in the rest of this thesis, we will use the term frequency scaling to express the simultaneous change of voltage and frequency. In a DVFS-capable multicore platform, power consumption generally consists of two parts: dynamic power consumption which is caused by execution activities and static power consumption due to the leakage current [25]. Static power is constant and independent of system voltage and frequency. Since in this thesis we use voltage and frequency scaling to manage energy consumption, therefore we focus on dynamic power reduction, as in [36] [37]. We adopt a system-level power model that is widely used, as in [25, 26, 33]:

$$P(f) = P_s + P_d = P_s + \hbar(P_{ind} + P_{dep}) \tag{1.1}$$

P_s is the static power which is consumed to maintain the basic running of circuits and can be removed when switching off the circuit. Dynamic power consumption P_d includes two parts: 1) a frequency-independent power consumption component P_{ind} which is caused by peripheral modules like memory and external devices when system is in active mode [26], which can be removed by putting the system into sleep mode, and 2) the dynamic power consumption of CPU and all other frequency-dependent devices, P_{dep}. $\hbar$ is a factor that describes the system modes, i.e., when $\hbar = 1$, the system is in active mode, and dynamic power is consumed, otherwise when $\hbar = 0$, the system is in sleep mode and no dynamic power is consumed. P_{dep} can be expressed as $P_{dep} = C_{eff} f^m$ where C_{eff} is effective switching capacitance and m is dynamic power exponent, normally no smaller than 2. Both C_{eff} and m are constants that depend on processor/technology characteristics.

In this thesis, we mainly focus on the frequency-dependent dynamic power consumption, where total energy consumption of the system is dominated by the energy consumed by processors to execute tasks. Multiple DVFS schemes introduced above are used to adjust the frequency/voltage scaling in order to minimize total energy consumption of system.

1.5 Fault Tolerance and Reliability Model

1.5.1 Fault Origin

The correct execution of an application can be threatened by several sources, such as radiation [38] and electromagnetic interference. A *fault* is a physical defect, imperfection, or flaw that occurs in a hardware or software component in [39]. A fault can cause a deviation from correctness or accuracy in computation, which becomes an *error*. A *failure* is a deviation from actual and expected value. A system is said to have a failure if the service it delivers to the user deviates from compliance with the system specification for a given period of time [40]. In general, faults are the sources of errors and errors the sources of failures [39]. Considering the origin of hardware faults, there are two main sources of faults [39]:

1. `Component Defects:` Component defects can cause many hardware faults. These include manufacturing imperfections, random device defects, and components wear-outs. Component failure caused by physical component defects is when a component is not functioning or performing as expected. It eventually damages the product. These defects may result in a complete breakdown or degradation in the performance of the device.
2. `External Factors:` Faults caused by external factors come from the environment, the user or the operator. External factors include temperature, vibration, electrostatic discharge, nuclear or electromagnetic radiation that affect the system. For instance, a fault caused by an external factor includes a cell in a memory to flip to an opposite value due to radiation.

Hardware faults generally can be classed into two types: *permanent faults* and *transient faults* according to the fault duration.

1. `Permanent Faults:` A fault is permanent if its impact cannot be removed from the system without external action. These faults usually occur due to physical defects in the hardware, such as shorts in a circuit, core aging, broken interconnections or stuck bits in the memory [39]. Permanent faults can be detected by online test routines that work concurrently with the normal system operation.
2. `Transient Faults:` A transient fault (also called soft error) stays active for a short time period. The causes of transient faults are mostly environmental, such as particles (the hit of cosmic ray neutrons or α-particles), electrostatic discharge, electrical power drops, overheating or mechanical shock. For example, in Fig 1.3 energetic particles such as α-particles can create minority carriers when they cross through the silicon bulk, this may be collected by the source/drain diffusions, altering the voltage value of these nodes causing a logic error [52].

During the normal lifetime period of the system, transient faults occur more frequently than permanent faults, and thus it is considered as the main threats for the correct execution of

applications [32, 41]. Due to the technology size reduction and the increasing scaling of CMOS technology, systems have become more susceptible to transient faults [42]. For DVFS-enable systems, a lower voltage/frequency scaling is more probable to cause a transient fault. In this thesis, we consider transient faults.

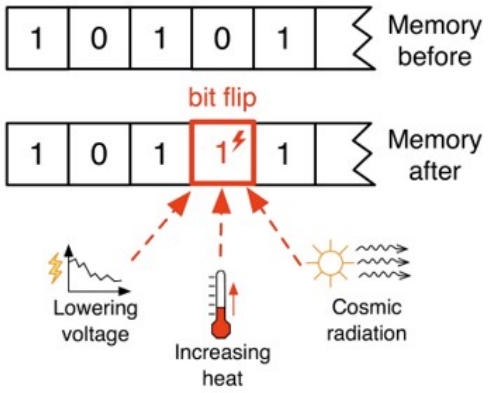


Figure 1.3: Cause to transient faults.

1.5.2 Reliability Model

The model of transient faults follows a Poisson distribution with an average fault rate λ [36] where fault rate is the expected number of faults per time unit [39]. For DVFS-enable systems with L pairs of voltage/frequency levels $\{(v_1, f_1), ..., (v_L, f_L)\}$, the fault rate at frequency f_l follows an exponential distribution:

$$\lambda(f_l) = \lambda_0 \times 10^{d\frac{f_{max}-f_l}{f_{max}-f_{min}}} \tag{1.2}$$

where λ_0 is the average fault rate at maximum frequency, d (called *sensitivity factor*) is a constant, used to measure the sensitivity of fault rate to voltage/frequency scaling. f_{max} and f_{min} are the maximum and minimal frequency in the L voltage/frequency levels respectively.

The *reliability* of a task execution is the probability of executing the task without any fault. During application execution, according to *exponential failure law* [39], the reliability of an execution varies exponentially as a function of its execution time as

$$R(f_l) = e^{-\lambda(f_l)\times t} \tag{1.3}$$

where t is the time duration of the execution which is inversely proportional to the frequency. As shown in Fig 1.4, the execution of an application has increasing reliability with the frequency increasing.

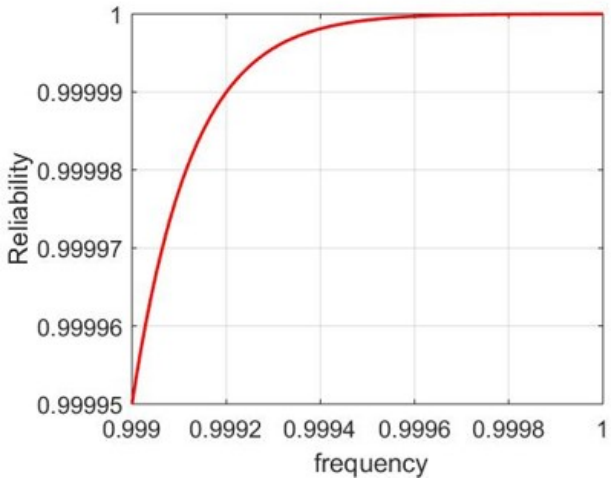


Figure 1.4: Reliability as a function of frequency.

1.5.3 Main Fault Tolerance Techniques

In practice, it is impossible to build a perfect system without fault occurrence especially with technology node decreasing [43]. To improve system reliability, several approaches use a high frequency to obtain a high reliability for the application execution. But with the complexity of a system increasing, the reliability of system drastically decrease even applying the highest frequency to execute tasks. For example, assuming the reliability of an individual task is very high, e.g. 99.999% at maximum frequency, when the system has 10 tasks, the reliability of the system after executing these 10 tasks is 99.99%; when the system has 20 tasks, this value decreases to 99.98%, and when the system has 100 tasks, the value is 99.9%. For system with high reliability requirements, the reliability requirement cannot be met by only using high frequency. *Fault tolerance* is the ability of a system to continue performing its intended functions in presence of faults while providing required reliability need.

There are various approaches to achieve fault tolerance. *Redundancy* is a commonly used fault tolerance technique. Redundancy is the provision of the system with functional capabilities that would be unnecessary in a fault-free environment [40], such as a replicated hardware component, an additional check bit attached to a flow of digital data, or additional lines of program code to verify the correctness of the program's results [40]. There are two categories of redundancy:

1. `Space Redundancy:` Space Redundancy (also called spatial redundancy) provides additional components, functions, or data items that are unnecessary for fault-free operation. Space redundancy is further classified into hardware, software, and information redundancy. Hardware-based techniques change the original architecture of the system or its components by adding extra hardware modules [53]. Such techniques are implemented during the design of the sys-

tem. Techniques based on hardware redundancy include 1) Duplication With Comparison (DWC) [54] in which it duplicates all components and a comparator module is added to detect a mismatch between both results and 2)Triple/N- Modular Redundancy (T/N-MR) in which processing units triplicate or replicate a task to produce the output [53].

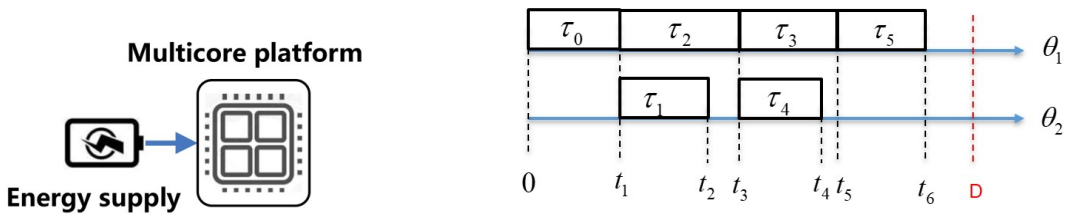
2. `Time Redundancy:` With time redundancy (also called temporal redundancy), the computation is repeated onto the same computing hardware component multiple times and the results are then compared to a stored copy of the previous result [54, 53]. Techniques based on time redundancy need a high time overhead because they require multiple time slots to perform the same operation.

To provide fault tolerance for task execution, two commonly used fault tolerance techniques based on the aforementioned redundancy:

1. `Task Recovery:`Task recovery [41, 55, 56, 26] is applied based on temporal redundancy, i.e., by exploring the available time slack during application execution and schedule recovery task(s) in the form of re-executing faulty task(s) usually at maximum frequency to recuperate system reliability.
2. `Task Replication:`Task replication [24, 32, 37, 44, 45, 46, 47] is another widely adopted technique to tolerate transient faults. It is based on space redundancy and multiple copies of every task are executed on different cores. There are two main approaches to provide replication. In *passive replication* [46, 47] (primary-backup) each task has a primary copy and back-up copy. The back-up copy is activated only when the primary copy fails its execution. In back-up fault tolerance technique, a fault detection mechanism is assumed to detect if there is a fault after task execution. In *active replication* (N-Module redundancy), each task is replicated multiple times (replicas) and these replicas are executed on different processors [24, 32, 37, 45]. By doing so, it is unlikely that all replicas of a task fail the execution. Error detection is provided at the end of execution so that we know whether the copy executed correctly or not. In this thesis, we consider an active replication technique to provide fault tolerance.

1.6 Real-Time Task mapping in Multicore Systems

In a multicore system, given an application consisting of multiple tasks, task mapping solves two problems: 1) *task allocation*, which is the spatial assignment that decides the task-to-core allocation; 2) *task scheduling*, which is the temporal assignment that provides the start time for each task. In this thesis, we perform task mapping at compile time, so it is called static mapping. Real-time systems, such as robotics, automotive applications and flight control systems, are computing systems that must react within precise time constraints to events in the environment [22]. As we focus on hard real-time systems, the correctness of the system output



(a) Multicore platform

(b) Task mapping with 6 tasks and 2 cores

Figure 1.5: Task mapping on multicore architecture.

depends not only on the logical result of the computation but also on the time at which the results are produced [32]. Fig. 1.5 depicts an example of task mapping on multicore platform. Assuming there are two cores, the six tasks in Fig 1.1, and the global deadline is D, the task-to-core allocation is achieved by mapping tasks 0, 2, 3, 5 to core 1, and tasks 1,4 to core 2. The start times of each task are $\{0, t_1, t_3, t_5\}$. The timing constraint is satisfied in this task mapping as the longest finish time of all tasks (called schedule length or makespan) does not exceed D.

Most modern cores support a wide range of voltages and frequencies [22] which is managed by DVFS. Voltage/frequency scaling has an important impact on energy consumption, reliability achievement and timeliness. Unfortunately, consequences are usually contradictory. For instance, when a lower frequency is used, the energy cost can be decreased, but this impacts negatively reliability and also causes longer execution times, which may lead to infeasible solution especially in real-time systems. Generally with energy cost decreasing, the quality of application execution is degraded. With application size increasing, allocating and scheduling a set of tasks to a group of cores under several constraints, such as energy cost, performance, reliability has become a major challenge in modern multicore real-time architectures. Therefore, it is worthy to study proper task mapping algorithms which can optimize the tri-criteria trade-off between reliability, timeliness and energy efficiency for task execution on multicore platforms.

Chapter 2

Energy-Reliability-Time Multi-criteria Task Mapping Mechanisms in SoA

In this chapter, we review the recent works that focus on task mapping on multicore platforms when considering *energy consumption*, *reliability achievement* and *timeliness*. High energy consumption has become one of the biggest obstacles to the rapid development of computing systems, and reducing energy consumption is an important research area in past decade and is also necessary for sustainable computing systems. To meet the increasing demand of high performance in safety-critical systems, reliability requirement has been considered in some industrial safety standards like ISO 26262 for automotive systems, DO-178B for avionics systems and IEC 61508 for industrial software systems. Energy saving and reliability enhancement are two irrelevant and normally conflicting issues when designing modern multicore platforms. Minimizing the schedule length (also called makespan) of a parallel application in order to obtain an effective execution is another fundamental issue which has attracted much attention, especially for data-intensive application. According to the above different goals, the studied task mapping problems in state-of-art (SoA) works can be summarized into three categories: energy minimization, reliability maximization and schedule-length minimization. In Section 2.1, we describe task mapping algorithms with energy minimization goal. In section 2.2, we present task mapping algorithms to maximize the reliability of the tasks in the system. Then, Section 2.3 describes representative scheduling algorithms proposed to reduce the overall schedule length required to execute the application.

2.1 Task Mapping Targeting Energy Minimization

Table 2.1 summarises representative task mapping approaches with DVFS technique. Abbreviation terms are: Energy Budget (EB), Real-Time (RT), and Reliability (R) for the constraints, maximizing system Reliability (mR), minimizing schedule length (mS) and minimizing Energy consumption (mE) for the objective of the approach. Tasks are Independent (I) or Dependent

(D). The platform has Homogeneous (HO), Heterogeneous (HE) or Single (S) processor (s). Based on the problem formulation and solving method, solutions are Feasible (F) or Optimal (O). Fault tolerance is provided by task Recovery (Rec) or task Replicas (Rep). Next sections describe these approaches.

2.1.1 Task Mapping Without Reliability Guarantee

Regarding minimizing energy consumption, which is also the objective in this thesis, approaches exist without considering reliability guarantee. First, we introduce some works without considering reliable execution, i.e., the reliability of task execution is not taken into account. The authors studied the problem of minimizing the power consumption of multiple-processor-core systems using multiple variable supply voltages and proposed a method to simultaneously do task allocation, task scheduling and voltage assignment for multiple processor-core systems in [57] and [58]. Originally, the problem is formulated using Mixed-Integer Non-Linear Programming (MINLP) model. It is known that task mapping problem is NP-hard problems. Optimal solution is given by introducing two modifications to enhance the efficiency of solving the origi-

Table 2.1: Representative State-of-the-Art targeting energy minimization.

Ref.	Goal	Task		Platform			Fault tol.		Constraints			Sol.	
	mE	I	D	HO	HE	S	Rec	Rep	RT	EB	R	F	O
[57]	√	√		√					√				√
[58]	√	√		√					√			√	
[34, 35]	√	√			√				√			√	
[59]	√		√	√					√			√	
[60]	√		√		√				√			√	√
[25]	√		√	√					√				√
[61]	√		√		√							√	√
[62]	√		√		√						√	√	
[63]	√		√		√				√			√	
[51]	√		√		√				√		√	√	
[41]	√	√				√	√		√		√	(√)	
[55, 56]	√	√				√	√		√		√	√	
[26]	√		√			√	√		√		√		√
[64]	√	√				√	√		√		√	(√)	
[65]	√	√		√			√		√		√		(√)
[66]	√		√		√		√		√		√	√	
[30]	√		√		√		√		√		√		√
[67]	√		√		√		√		√		√	√	
[68]	√	√		√				√	√				√
[69]	√	√		√	√			√	√		√	√	
[32, 45]	√	√		√				√	√		√	√	
[70]	√	√			√				√		√	√	
[24]	√		√		√			√			√	√	
[37]	√	√			√			√	(√)		√	(√)	√
[44]	√		√	√				√	√			√	
[71]	√	√		√				√	√		√		√
[72]	√		√	√				√	√		√		√
[73]	√	√		√				√	√		√		√
Prop.	√	√	√	√				√	√		√	√	

nal MINLP problem, given that the complexity is large for solving the MINLP problem in [57]. However, for large scale task-sets, the complexity is very high so that the solution cannot be obtained in reasonable time. The authors proposed a divide-and-conquer heuristic algorithm to solve the large task-sets efficiently. The allocated and scheduled Task Flow Graph (AS-TFG) is divided into several small partitions where the MINLP problems are easier to solve in [58]. After the optimal solution of each partition is found by solving the MINLP formulation, all the local optimal solutions of all the partitions are integrated together and combined as nonlinear programming (NLP) problem which tries to further optimize the total energy. In this way, a very good approximate global optimum solution can then be obtained. The authors proposed a novel Relaxation-based Iterative Rounding Algorithm (RIRA) to achieve minimum total energy consumption for all tasks without violating the deadline constraint under three types of platforms, i.e., dependent platform without runtime adjusting where all of the processors must operate at a common frequency and the shared frequency cannot be adjusted during runtime after setting the initial frequency, dependent platform with runtime adjusting where the initial frequency can be adjusted during runtime, and independent platform where processors can operate at different frequencies at any time and can adjust their execution frequencies independently in [34, 35, 59]. The initial task mapping problem is formulated as a binary integer problem, then relaxed to be a convex optimization problem by relaxing the binary variables (such as $x_{i,j}$, which denotes task i is allocated to processor j) as continuous in $[0, 1]$. Iteratively, the current-to-be-scheduled task is one by one allocated to the most possible processor which achieves highest $x_{i,j}$ obtained by solving the convex optimization problem. By doing this, the proposed algorithm achieved near-optimal scheduling under most cases. The studied problem is extended to dependent tasks in [59]. After the task-to-processor allocation is achieved in a same way like above, the authors developed a genetic algorithm to do task scheduling by searching voltage/frequency assignment for both tasks and communications in the application.

Instead of assuming Worst-Case-Execution-Time (WCET) as used in most works, the task execution time is modeled as a probabilistic random variable in [60]. Using a probabilistic approach can be applicable because the worst case assumption may not be practical in reality and thus increases unnecessary cost. The authors studied heterogeneous assignment with probability problem which is useful for both hard and soft real-time systems. Given the number of different type of processors and an application modelled as a probabilistic data flow graph, the goal is to find the proper type of processor for each task so that the total cost is minimized and deadline constraint is met with a guaranteed confidence probability. The authors provided both optimal solutions for the simple application case like a tree or simple path, and near-optimal solutions for more general problems.

As said previously, when talking about energy (power) consumption, two parts are generally considered: dynamic energy (power) consumption and static energy (power) consumption. As

said previously, Dynamic Voltage and Frequency Scaling (DVFS) is a technique to reduce dynamic power consumption by reducing voltage and frequency of a processor. Static power (such as leakage power) consumption cannot be ignored especially as the increasing of chip density leads to dramatic increase of static power. Dynamic Power management (DPM) [25, 30] is applied to explore the idle time interval to reduce static power by switching the processor to a lower power consuming mode like sleep mode. An energy-minimization problem is formulated under deadline constraints by integrating DVFS and DPM in [25]. The authors proposed a technique to directly model the idle time intervals of processors so that DPM can be integrated into the problem formulation. The optimal solution is obtained by solving a Mixed-Integer-Linear-Programming (MILP) formulation problem with CPLEX solver tool.

In cloud computing, cost-minimization is a critical issue. The authors studied the problem to decide the number of allocated processors, the type of each processor and task scheduling on processors in order to minimize total costing with a given usage cost per time unit [61]. The optimal solution is proposed by iterating through all possible configurations which caused a large time complexity. A heuristic is proposed to reduce time complexity while keeping approximately optimal solution by separately deciding processor type selection and task scheduling.

2.1.2 Task Mapping With Reliability Guarantee

For safety-critical applications, task reliable executions must be taken into account. Reliability of an application is defined as the probability of executing an application without meeting failure. Energy management and reliability enhancement have been jointly studied with the advent of DVFS technique. The authors investigated how frequency and voltage scaling effects on the fault rate in [36]. Focusing on transient faults, two fault rate models are proposed based on how voltage and frequency scaling changes the fault rate. The exponential fault rate model is largely used in recent works when on transient faults and also used in this thesis. Based on the exponential fault and an occurrence considering Poisson distribution, the reliability is given as introduced in Section 1.5.2.

As depicted in equations 1.2 and 1.3 in Section 1.5.2, a higher frequency provides a higher reliability for a task execution, while this will cause higher energy consumption. Energy saving and reliability enhancement are two conflicting objectives as studied in [62, 63, 51]. The authors in [62] studied the problem of energy consumption minimization of a reliable application on heterogeneous systems without using fault tolerance. The problem is decomposed into two sub-problems, i.e., satisfying reliability goal by transferring the reliability goal of the application to the reliability of each task, and minimizing energy consumption by selecting the processor with minimum energy consumption while satisfying its reliability requirement for each task. The authors in [63] presented a workflow for the scheduling problem by jointly minimizing energy consumption and maximizing system reliability which is a bi-objective problem under deadline

constraint. A genetic algorithm is developed to obtain a fine pareto front. The authors studied the problem of power consumption and reliability trade-off optimization on heterogeneous multiprocessor systems with DVFS under schedule length and reliability constraints in [51]. A heuristic-modified whale optimization algorithm [74] is proposed to search for solution of task-to-processor mapping, and determine the task execution order on each processor by using a downward-ranking heuristic. Tasks on critical path are rescheduled to reduce makespan without increasing energy consumption and adjust the frequency to maintain high system reliability.

Applying high frequency can provide a high reliability for task executions while this also lead to large energy consumption. To cope with this issue, fault tolerance is an effective way to provide reliable execution which has been studied in recent researches. We briefly introduce some typical approaches which provide fault-tolerance techniques introduced in Section 1.5.3.

Task Recovery Fault Tolerance

Executing the tasks at maximum platform frequency may lead to some time slack. Using DVFS technique, the available time slack can be reserved to execute a recovery task (individual [41, 55] or shared [56, 26]) with the maximum frequency, to preserve reliability and remaining time slack can be used to scaled down voltage and frequency for other tasks in order to save energy. If an error is detected, the recovery task is called up. Task recovery is proposed to guarantee reliability requirement by exploring the slack time in application execution combined with DVFS technique in [41, 55, 56, 26] with the goal to minimize energy consumption. Taking a single task model (a set of aperiodic tasks) into account in [41], the authors studied the problem how to use the given dynamic time slack to schedule an additional recovery task in order to preserve its reliability for each task. Except the time slack used to schedule a recovery task, the remaining slack can be used to save energy by reducing voltage and frequency assignment to task execution. Instead of only considering one task at a time in [41] when allocating slack to execute tasks, the authors considered all tasks are considered at same time when allocating slack to individual tasks to guarantee the reliability target for each task in [55]. In these two works, an individual recovery task for each task is scheduled which is quite conservative. Shared recovery technique is applied where the recovery task(s) can be used for any task when it fails its execution in [56, 26]. Generalized shared recovery mechanism is proposed in [56] where a few number of recovery tasks are shared among all tasks to achieve system reliability target. The authors solved the energy minimization problem for independent tasks by determining how many recovery tasks are reserved and frequency-to-task assignment under reliability and deadline constraints. Dependent tasks are considered in [26], the authors addressed similar problem as in [56] to find execution order of dependent tasks and their frequency assignment under task deadline and precedence constraints so that the total energy consumption is minimized while preserving system reliability requirement. Dependent tasks with common deadline in heteroge-

neous systems is considered in the studied energy minimization problem under reliability and global deadline constraints in [66]. The authors addressed the scheduling of energy-reliability trade-offs for hard real-time applications on heterogeneous embedded systems in [67]. Given a fixed number of transient faults and task-to-processor allocation, the proposed approach determined voltage scaling and start time of each task to minimize total energy consumption, while satisfying the real-time and reliability constraints.

Among other works applying recovery technique to provide fault tolerance, the authors in [64] presented an energy efficient quasi-static scheduling algorithm which consists of an offline feasibility analysis and an online voltage scaling. The static slack in offline task scheduling and dynamic slack due to variations in actual task execution time are both utilized to schedule recovery tasks. Neither scheduling a separate recovery for each task nor shared recovery for all tasks, the authors in [65] selected a subset of tasks to share the reserved resource to schedule recovery tasks. Given a task set, it is partitioned into fault-unprotected task set which is executed with highest frequency, and fault-protected task set with a scaled down frequency. Tasks in fault-protected set share a reserved time slack to schedule a recovery task if a fault occurs.

Task Replication Fault Tolerance

Another effective fault tolerance technique is applying task replication. We introduce some representative approaches, aiming at energy savings, which apply replication schemes to provide fault-tolerance. Primary-backup is a passive replication to provide fault tolerance [69, 68, 46]. Traditionally primary-backup technique is used for promising fault tolerance in dual-processor systems [69, 68]. Regarding minimizing energy consumption, the authors studied the energy efficiency of dual-processor system in [68]. Two copies (primary and backup copies) of each task are executed in different processors. One processor executes the primary copy and the other one executes the backup copy. When the primary copy of the task is executed without fault, the other processor stops executing the backup copy. Two power management schemes are applied to reduce energy consumption: the first one is static power management which takes WCET into account and schedules tasks in the offline fashion, the second one is dynamic power management which utilizes the time slack between tasks' actual execution time for further energy reduction at runtime. Considering both homogeneous and heterogeneous dual-processor systems, the authors applied a modified primary-backup technique to maintain reliability when DVFS is used to reduce energy consumption in [69]. Rather than the pessimistic WCET assumption, the actual execution time in practice can be less, so there can exist some available time slack which can be used to save energy by scaling down voltage and frequency when executing tasks. A side effect of reducing voltage is the increasing rate of transient faults, so extra copies are scheduled to account for the loss of reliability due to frequency scaling.

In recent literature, active replication has attracted much attention. Replication has several

advantages as an energy efficient fault tolerant technique. First, by applying multiple replicas for a task, the reliability requirement can be guaranteed especially when a task has very high reliability threshold. In this case, replication is the only way to achieve the high reliability. Second, it can explore the frequency space to execute some replicas at a low frequency to save energy without sacrificing the reliability requirement using DVFS. Aiming at minimizing total energy consumption, some researches compute the required number of replicas (also called replication degree) to always meet reliability constraints [32, 45, 24, 70]. The authors in [32] studied the problem of obtaining a given task level reliability for a set of independent tasks under deadline constraints by deciding the number of replicas and frequency assignment for each task. Heuristics are typically proposed, due to high computation complexity or NP-hard problems. A first-fit decreasing heuristic is used to find the processor for each replicas in [32] and Earliest-Deadline-First scheduling heuristic decides if the processor is fit for the replicas. The replicas of a task are allocated to different processors at same frequency. Starting with initial configuration where all replicas are executed at maximum frequency to get the initial static mapping and scheduling for all tasks, if there exists available resource, an iterative procedure is used by searching all possible configurations (i.e. the degree of replication and frequency assignment) for each task. A dynamic scheduling and mapping is developed to further reduce energy consumption by cancelling the other replicas when one replica is executed correctly. In [45], the authors took [32] as the reference paper and improved the scheduling algorithm by 1) using a layered worst-fit decreasing heuristic which selects the least-loaded processor as a fit for current scheduled task replicas, and 2) instead of allocating all replicas of a task before moving to next task, the first replica of each task is allocated to the fit processor, and then the second replica of each task (if it exists), and so on. Same problem as [32, 45] is studied except the platform is heterogeneous in [70]. Due to the processor heterogeneity, it cannot be known how many replicas are needed for a task to meet the reliability requirement before knowing the task-to-processor allocation, because different processors in heterogeneous platforms have different fault rate and frequency levels. So the way to calculate how many replicas needed like in [32, 45] cannot be used. To tackle this, the authors in [70] assumed full replication where all processors are added to schedule a replica for each task. Starting from computing the reliability by only considering the first replica on first processor, if it does not meet reliability threshold, then a replica is added. This is done iteratively still either the reliability threshold is met (replication setting and allocation for this task is finished) or the added total processor number exceeds the number of processors in the given platform (not feasible problem).

Dependent tasks are considered to be executed on heterogeneous systems with a system reliability goal in [24]. The authors first proposed an energy efficient scheduling with reliability goal without fault tolerance by deciding processor and frequency combinations of the tasks to minimize total energy consumption under a system reliability constraint, which consists of three

steps: prioritizing tasks, transferring system reliability goal to each task and reducing energy consumption. Then an energy efficient fault tolerant scheduling is proposed by applying active replication. It also include three steps where prioritizing tasks and transferring system reliability goal to each task are same, while in the third step, the later scheduling algorithm selects multiple processor and frequency combinations for the replicas of each task. This implies the replicas of a same task are not necessary to be executed at same frequency.

However, the increased number of replicas leads to large energy consumption, combined with a negative impact on execution time. When the real-time constraints are strict, solutions may not exist. To reduce this negative impact, the number of replicas must be restricted. Duplication is one kind of replication where at most two replicas of a task are executed (original copy and duplication copy). In [44] a linear chain workflow is considered and the goal is to minimize total energy consumption under timing constraints. Even though there is no reliability constraint, two ideas are applied to guarantee reliable execution: 1) if an error strikes a task's execution, this task is re-executed at maximum frequency, and 2) some tasks are selected to be duplicated on a different processor at same frequency to mitigate the effect of failures. A heuristic is proposed to tackle the studied problem by determining which tasks to be duplicated and which frequency to execute the tasks.

Optimization problems with multiple objectives are studied in [37]. Considering the objective of minimizing total energy consumption, an Integer Linear Programming (ILP) approach maps independent tasks on a heterogeneous platform to satisfy a given percentage target of duplicated tasks, under a cost constraint where the number of processors are fixed. Then a heuristic algorithm based on Earliest Deadline First (EDF) is proposed for the energy minimization problem under cost, duplication percentage and deadline constraints.

In our published works [71, 72, 73] and this thesis (Prop.), duplication is applied as the fault tolerance technique to provide reliable execution. Different from most SoA works where replicas of a task are executed at the same frequency, we let the different replicas (original copy and duplication copy) to be potentially executed at different frequencies which is more efficient to manage energy consumption.

2.2 Task Mapping Targeting Reliability Maximization

As discussed in section 1.5.2, a high frequency can provide a high reliability for the task execution, and indeed this approach is employed a lot especially in safety-critical domains. Nowadays replication is also utilized a lot as an effective reliability management technique. Let assume that the reliability of the execution of a task 0.9 and the task is executed with three replicas. This task will fail its execution only if all three replicas fail their execution, which has a probability of $(1-0.9)^3$. The reliability considering the three replicas is $1-(1-0.9)^3 = 0.999$

which is largely increased. Regarding reliability enhancement, except the works introduced in Section 2.1, this section introduces approaches aim at maximizing reliability under timing, energy budget and/or reliability requirements, summarized in Table 2.2.

2.2.1 Task Mapping Without Fault Tolerance

Some approaches without fault tolerance map only original tasks [75, 76, 77]. In [75] the authors studied the problem of deciding frequency assignment to a set of tasks on uni-processor with the goal of maximizing overall reliability under given energy budget and global deadline constraints. First an optimal static solution is provided under the assumption that all tasks are executed with worst-case workload. Then by detecting if there exists early completion in actual execution at runtime, the frequencies can be adjusted to improve all reliability by making best use of the total energy budget. A bi-objective scheduling algorithm is proposed to optimize both reliability and schedule length on heterogeneous systems in [76]. To maximize the reliability, the problem is built with independent unitary tasks under a given makespan. DVFS technique is not supported in this work. The optimal solution is provided by deciding task-to-processor allocation. For the bi-objective optimization, the authors proposed an algorithm that approximated the Pareto-curve. Similarly, in [77], the reliability maximization is studied under global deadline and energy budget, including communication energy cost. The authors proposed a three-step reliability management algorithm, namely, prioritizing tasks, allocating communication edges and reclaiming time slack.

2.2.2 Task Mapping With Fault Tolerance

Some approaches with fault tolerance aiming at maximizing reliability are studied in [78, 79] where replication is applied and [80, 14] where both permanent and transient faults are considered. In [78] the authors presented a replication-based scheduling for maximizing system reliability while meeting reliability threshold for each task. The reliability model includes communication reliability. A two-step heuristic is proposed by deciding the number of replicas for each task. Redundancy Multithreading (RMT) is a prominent technique to mitigate transient faults. In [79] a combination use of Simultaneous Redundant Threading (SRT) in which running

Table 2.2: Representative State-of-the-Art approaches targeting reliability maximization.

Ref.	Goal	Task		Platform			Fault tol.		Constraints			Sol.	
	mR	I	D	HO	HE	S	Rec	Rep	RT	EB	R	F	O
[75]	√		√			√			√	√		√	√
[76]	√	(√)	√		√				√			√	(√)
[77]	√		√		√				√	√		√	
[78]	√		√		√			√			√	√	
[79]	√		√	√				√	√				
[80, 14]	√		√	√				√	√	√		√	

all replicas on the same processor and Chip-level Redundant Multithreading (CRT) in which replicas are executed on different processors is utilized to provide fault tolerance, which also provides additional choices for balancing the usage of cores and for optimizing reliability.

Other approaches jointly considered transient faults and permanent faults in reliability optimization [80, 14]: Soft-Error Reliability (SER) related to transient faults and lifetime reliability (measured in form of mean time to failure, MTTF) related to permanent faults. Given energy budget and global deadline constraints, the objective in [80] is to jointly maximize lifetime (MTTF) and soft-error reliability by determining what tasks to be replicated, task-to-processor allocation and when to start time. Task replication is adopted to tolerate transient faults as well as to limit the execution of too many tasks which has a negative effect on processor aging. Similar problem is studied in [14] in which the number of replicas of each task is given while satisfying energy budget, global deadline and task dependency constraints. An evolutionary algorithm is proposed to find solutions to the multi-objective optimization problem.

2.3 Task Mapping Targeting Schedule-Length Minimization

Schedule length is an important performance metric in computationally intensive computing systems which can support execution of several applications. These systems need effective utilization of the limited resources such that applications can be completed as early as possible. Table 2.3 summarizes these approaches.

2.3.1 Task Mapping Without Fault Tolerance

Regarding minimizing schedule length without taking fault tolerance into account, some works have proposed several heuristics for similar problems: 1) heterogeneous earliest finish time algorithm and the critical path on a processor algorithm when there is a bounded number of heterogeneous processors [81], 2) cluster-based task scheduling algorithm where tasks are clustered to minimize the worst schedule length when there is a large number of heterogeneous processors [82], and 3) partitioning and scheduling algorithm which focuses on maximizing the

Table 2.3: Representative State-of-the-Art approaches targeting schedule-length minimization.

Ref.	Goal	Task		Platform			Fault tol.		Constraints			Sol.	
	mS	I	D	HO	HE	S	Rec	Rep	RT	EB	R	F	O
[81]	√		√		√							√	
[82]	√		√		√							√	
[83]	√		√		√							√	
[18]	√		√	√								√	
[84]	√		√		√				√	√		√	
[85, 86]	√		√		√							√	
[47]	√		√		√			√			(√)	√	
[87]	√		√		√			√		√	√	√	√
[88]	√		√		√			√			√	√	

overall completion time of the critical path [83]. In [18] the authors built a rule-based model to maximize the parallelism of DAG task graph first, then a scheduling method is proposed by ordering tasks in three sequences 1) critical path, 2) early predecessors path of the critical path, and 3) longer paths to reduce the schedule length. In [84], the authors addressed the problem of finding a proper processor and frequency for each task to generate minimum schedule length under energy budget and global deadline constraints. The proposed algorithm suggested a weighted-based mechanism to pre-assign energy consumption for unscheduled tasks and then transfer all processor and frequency combinations to select the best combination for each task.

We have introduced task duplication is an effective technique to tolerate fault occurrence as well as to provide reliability guarantee in Section 2.1. Task replication may also be an efficient way to improve performance when scheduling parallel tasks with dependency constraints normally presented in the form of Directed Acyclic Graph (DAG). We focus on such approaches here. In [47], granularity is defined as the ratio of the sum of the slowest commutation times of each task to the sum of the slowest communication times along each edge. It is called coarse grain, if a task graph has a granularity equal or larger than 1, otherwise it is called fine grain. For fine grain DAG tasks, unnecessary communication delays occur when a task has its predecessors on different processors. By applying task duplication, the unnecessary communication delays can be eliminated, thereby reducing the overall completion time of the application. Regarding minimizing schedule length, task duplication based technique is considered in [86, 85]. The objective is to assign a set of dependent tasks to different processors such that all tasks can finish execution as soon as possible. Task duplication-based clustering algorithm is proposed in [85] to generate initial task clusters. Task duplication is adopted when a task's critical predecessor is not on the same processor. A duplication copy of this critical predecessor is added on the same processor of the current task. New clusters are merged to further shorten the schedule length. In [86] the goal is to minimize both schedule length and energy consumption. The authors proposed two duplication-based algorithms. According to the decision to select which tasks to be duplicate, one algorithm selects the tasks to be duplicated when the schedule length can be shortened while the energy increase is within a given threshold. The other algorithm defines a ration as the division of energy saving and schedule length reduction. A task is selected to be duplicated if its ration value is within a given threshold.

2.3.2 Task Mapping With Fault Tolerance

Regarding minimizing schedule length with fault tolerance, task replication is applied in [47, 87, 88]. A task mapping and scheduling algorithm which adopts full replication is proposed in [47] under a given number ϵ of failures when dependent tasks and heterogeneous platform are considered. Each task is executed with $\epsilon + 1$ copies and each copy is allocated to different processor. After determining the task priority, the first $\epsilon + 1$ processors with minimum finish

time are selected to executed these $\epsilon + 1$ copies for the current scheduled task. Then a variant of the proposed algorithm is designed to reduce the communication cost induced by the replication mechanism. The authors in [87] addressed the problem of mapping a set of DAG tasks onto multicore heterogeneous platforms such that total execution time is minimized under power consumption, failure rate and temperature constraints. Based on list scheduling where the list of tasks is ordered in a fixed order, which can be determined e.g. by the priority of executing the tasks, a heuristic is proposed by applying active replication and DVFS to trade off failure rate and the number of replicas. Besides, the optimal scheduling is obtained using Integer Linear Programming (ILP) and solved by CPLEX solver. In [88] the authors studied the problem of makespan optimization jointly considering reliability, temperature and stochastic characteristics of precedence-constrained tasks. An affinity (probability)-driven task allocation and scheduling approach is proposed and a heuristic is designed by assigning a task to the processor with highest affinity (i.e., most possible processor to execute the task).

2.4 Limitations of SoA Task Mapping Approaches

Although both reliability and energy management have been extensively (but often independently) studied, their co-management has been addressed only recently as in some SoA works presented in Section 2.1 and Section 2.2. Without particularly considering multi-objective approaches, the main limitations can be summarized as below:

1. `Task mapping without fault tolerance:` Without providing fault tolerance, one category of such approaches focuses on meeting the reliability requirements, considering only original tasks (and thus, no duplication) [62, 51]. However, these methods usually assign high frequencies to tasks, which it does not only cause large energy consumption, but it also may not always satisfy the reliability constraints, even with the highest processor frequency.
2. `Recovery task executed at maximum frequency:` Many approaches have applied task recovery to provide fault tolerance [41, 55, 56, 26, 66, 67, 64], where the recovery task(s) are normally executed at maximum frequency. First, the negative effect is energy consumption at maximum frequency. Furthermore, these works assume there is a fixed number of faults. An energy and reliability trade-off to tolerate the given number of faults is then searched [67] which is less practical. Furthermore, some works considered quite simple systems like uni-processor platforms [64] rather than multi-processors platforms.
3. `Task replication:` Task replication, which is also the technique we have considered in this PhD thesis, can be applied to provide fault tolerance. Full replication where each task of the application is replicated on multiple different processors, such as in [32, 47], leads to large energy consumption, combined with a negative impact on the execution time because the end-times of tasks are delayed, due to the execution of task replicas. One way to cope with

this issue is to do partial task replication. In this case, not all the tasks in the task set but only a part of the task set is selected to do replication on different processors. In [78, 24], partial replication is performed with heuristics without considering timing constraints. In [32, 45], it is assumed all replicas of a task are executed at the same frequency, leading once again to a high energy consumption.

Considering the above limitations, in this thesis we apply task replication as the fault tolerance technique to decide the execution behavior of the application on multicore architectures. Partial replication is studied where we decide which tasks from the task set to be duplicated in order to not cause large energy consumption due to the execution of task replicas. The proposed approach is expected to be a good trade-off between reliability, energy consumption, and real time constraint, compared to no duplication with high frequency to meet reliability, thus leading to a large energy consumption or duplication of every task where real time constraint is difficult to satisfy with strict deadlines. Furthermore, we target three DVFS schemes implemented in recent multicore platforms or proposed in recent researches to manage energy consumption. To evaluate the performance of our proposed techniques, we have done a large set of experiments to compare the results of the proposed approach with the task mapping approach when fault tolerance technique is not applied and with the approach when full replication is used.

Chapter 3

Energy Efficient Fault Tolerant Task Mapping with Optimal Solutions

In this chapter, we introduce the task mapping problem of minimizing total energy consumption under real-time and reliability requirement constraints and provide the means to obtain optimal solutions. Firstly, a motivation example is given in section 3.1 to explain how the duplication technique can provide a trade-off between energy savings and reliability enhancement. In section 3.2, we describe the studied problem for independent tasks, while section 3.3 extends the problem to dependent tasks, under three DVFS schemes providing optimal solutions. Experimental results are presented in sections 3.2.4 and 3.3.4, showing that the proposed approach outperforms two SoA approaches used to solve the studied problems with optimal solutions.

3.1 Motivation Example

Initially, we show the benefits of the proposed Reliability-aware Fault-tolerant Task Mapping (RAFTM) approach through a motivational example. We consider a simple case with a single task only, under the TL-DVFS scheme, having Worst Case execution Cycles equal to $W = 4 \times 10^8$, a reliability threshold equal to 0.9995 and the first five voltage/frequency levels of the platform used in the experiment section (Table 3.3). Table 3.1 depicts the possible solutions (Sol.), with the selected voltage/frequency level (f) [1] [2], reliability (R), execution time (t), and energy consumption $(E = P_l \times t)$. Columns f_1-f_5 corresponds to the execution of only the original task at frequency f_i (i.e., no duplication), while columns f_1/f_1-f_5/f_5 corresponds to the execution of both the original and its replica at frequencies f_i and f_j respectively.

Table 3.1.a enumerates all feasible solutions found by a particular approach and that satisfy only the reliability constraint, order with increasing execution time, for i) the proposed

1. To ease the presentation, in the rest of the document we will talk about frequency levels or more generally about frequency adjusting. However voltage and/or frequency adjusting are both concerned.

2. In Table 3.1, f_i are ordered in increasing voltage/frequency values (f_1 is the level with lowest voltage and frequency, f_5 is the level with highest voltage and frequency)

Table 3.1: Motivational Example. (*Optimal solutions are highlighted in bold).

Sol.	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
f	f1	f2	f3	f4	f5	f1/f1	f1/f2	f1/f3	f1/f4	f1/f5
R	0.9753	0.9964	0.9994	0.9999	∼1	0.9994	0.9999	∼1	∼1	∼1
t	0.4994	0.4825	0.4677	0.4547	0.4431	0.9988	0.9818	0.9671	0.9541	0.9425
E	2.1169	2.7905	3.6959	4.926	6.6141	4.2338	4.9074	5.8128	7.0429	8.731
Sol.	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20
f	f2/f2	f2/f3	f2/f4	f2/f5	f3/f3	f3/f4	f3/f5	f4/f4	f4/f5	f5/f5
R	∼1	∼1	∼1	∼1	∼1	∼1	∼1	∼1	∼1	∼1
t	0.9649	0.9501	0.9372	0.9256	0.9353	0.9224	0.9108	0.9094	0.8978	0.8862
E	5.581	6.4864	7.7165	9.4046	7.3918	8.6219	10.31	9.852	11.5401	13.2282

a) Feasible solutions of three approaches under reliability constraint only

RAFTM	S5,S4,S20,S19,S18,S17,S16,S14,S15,S13,S10,S12,S9,S11,S8,S7
RAM	S5,S4
TDM	S20,S19,S18,S17,S16,S14,S15,S13,S10,S12,S9,S11,S8,S7

b) Feasible and **optimal*** solutions under reliability and deadline constraints

D	[0,0.4431)	[0.4431, 0.4547)	[0.4547, 0.8862)	[0.8862, 0.8978)	.....	[0.9818, -)
RAFTM	-	***S5***	S5,***S4***	***S4***-S5,S20	.....	S4-S5,***S7***-S20
RAM	**-**	***S5***	S5,***S4***	S5,***S4***	.....	S5,S4
TDM	**-**	**-**	**-**	S20	.....	***S7***-S20

approach (RAFTM), ii) the Reliability-Aware Mapping (RAM) approach, which satisfies the reliability constraint without any task duplication, and iii) the Task Duplication Mapping approach (TDM), always performing task duplication (more details about RAFTM and TDM are given in the experimental section).

Table 3.1.b explores how the deadline constraint (D) affects the feasible and optimal solutions. For instance, when $0.8862 \leq D < 0.8978$, S4 is the optimal solution. The proposed approach can find the solutions {S4-S5,S20} including the optimal solution. RAM obtains the solutions {S4,S5}, also including S4. However, for TDM, only one solution is feasible, i.e., S20, while for more strict deadlines no solution is found. Therefore, in this case, full replication based approaches, such as TDM, are not able to find optimal solutions, due to too strict deadline constraints, even for only a single task. For more relaxed deadlines, when $0.9818 \leq D$, the proposed approach can find the solutions {S4-S5,S7-S20}, where S7 is the optimal solution. TDM obtains the solutions {S7-S20}, also including the optimal solution. However, RAM obtains the solutions {S5,S4}, i.e., without the optimal solution.

Therefore, reliable original execution based approaches, such as RAM, cannot exploit energy efficient solutions due to no task duplication, leading to higher energy consumption. Full duplication approaches, such as TDM, do duplication for every task which may also lead to larger energy consumption and fail to find solutions especially at strict deadlines. In our work, the proposed RAFTM approach exploits the benefits of the aforementioned methods by applying partial task duplication where some of the tasks are selected to do duplication, whenever it is advantageous.

3.2 Task Mapping Problem for Independent Tasks

To simplify the problem, we first target independent tasks. The goal is to minimize the total energy consumption of the system, subject to a set of reliability and real-time constraints. To achieve that, we decide: 1) frequency assignment of original and duplicated tasks ($\boldsymbol{s}$); 2) duplication of original tasks ($\boldsymbol{\sigma}$); 3) allocation of original tasks and duplicated tasks ($\boldsymbol{q}, \boldsymbol{d}$). The following paragraphs describe the constraints and objective function in the studied problem.

3.2.1 System Model

Based on Chapter 1, we first briefly introduce the task model, power consumption model and reliability model. Table 3.2 summarizes the main notations and their definitions. For the sake of thesis presentation, when original and duplicated tasks must be distinguished in mathematical formulations, the subscript $k \in \{o, d\}$ indicates the original task (^{o}) or the duplicated task (^{d}). If no subscript exists, the mathematical formulation is valid for both.

Task Model: We consider a set of N independent tasks, i.e., $\{\tau_1, \ldots, \tau_N\}$. Each task τ_i is measured in Worst Case Execution Cycles (WCEC) W_i. All tasks must be executed before a common deadline D, which is the scheduling period. No preemption occurs between different tasks executed on the same processor. Without loss of generality, in the rest of the paper, the release times of all tasks are considered at the start of the scheduling period. R_i^{th} denotes the reliability threshold of task τ_i. Each task has its own reliability constraint, since functions of an application exhibit distinct significance and/or vulnerabilities, due to variations in the spatial and temporal vulnerabilities of different instructions [89].

Platform and Power Model: A multicore platform is considered with M homogeneous processors, i.e., $\{\theta_1, \ldots, \theta_M\}$. The multicore platform can support three DVFS schemes introduced in Section 1.3, i.e., i) task-level DVFS (TL-DVFS), ii) processor-level DVFS (PL-DVFS), and iii) system level DVFS (SL-DVFS). For each core, there are L different Voltage/Frequency (V/F) pairs $\{(v_1, f_1), \ldots, (v_L, f_L)\}$. When task τ_i is assigned with frequency f_l, its execution time is calculated as $et_i = \frac{W_i}{f_l}$. For each processor θ_m, the power consumption introduced in Section 1.4 is modeled as the sum of static power P_l^{sta} and dynamic power P_l^{dyn}, i.e., $P_l = P_l^{sta} + P_l^{dyn}$. The dynamic power consumption with V/F level (v_l, f_l) is given by $P_l^{dyn} = C_{eff} v_l^2 f_l$, which is a common used model assuming that frequency and voltage scaling have a linear relation.

Fault Model and Reliability: We focus on soft errors that follow a Poisson Distribution with fault rate $\lambda(f)$ at frequency f introduced in Section 1.5.2, modeled as $\lambda(f) = \lambda_0 \times 10^{d\frac{f_{\max}-f}{f_{\max}-f_{\min}}}$, and the reliability of an original task at frequency f_l is given by $R_i^o(f_l) = e^{-\varphi_i(f_l)}$, where $\varphi_i(f_l) = \lambda(f_l) \times et_i^o$. If the reliability of original task τ_i is larger than its reliability constraint, the execution is considered as reliable, and, thus, the reliability of τ_i (donated as R_i) is not modified, i.e., $R_i = R_i^o$. Otherwise, the task τ_i is duplicated and the duplication task (also called replica)

Parameters	**Definitions**
$\boldsymbol{M}$	$\{1, \ldots, M\}$, with M number of processors
$\boldsymbol{N}$	$\{1, \ldots, N\}$, with N number of tasks
$\boldsymbol{L}$	$\{1, \ldots, L\}$, with L number of voltage/frequency levels
W_i	WCEC of task τ_i
D	global deadline
τ_i^o/τ_i^d	the original/duplication copy of task τ_i
(v_l, f_l)	the l^{th} voltage/frequency level
R_i^{th}	reliability threshold of task τ_i
R_i^o	reliability of original copy of task τ_i
R_i^d	reliability of duplication copy of task τ_i
et_i^o	execution time of original copy of task τ_i
et_i^d	execution time of duplication copy of task τ_i
Binary Variables	**Definitions**
$\sigma_i = 1$	if task τ_i is duplicated, else $\sigma_i = 0$
$q_{im} = 1$	if τ_i^o executes on processor θ_m, else $q_{im} = 0$
$d_{im} = 1$	if τ_i^d executes on processor θ_m, else $d_{im} = 0$
TL-DVFS:	$s_{il} = 1$, if original task of τ_i executes with f_l, else $s_{il} = 0$
	$c_{il} = 1$, if duplication task of τ_i executes with f_l, else $c_{il} = 0$
PL-DVFS:	$s_{ml} = 1$, if processor θ_m executes with f_l, else $s_{ml} = 0$
SL-DVFS:	$s_l = 1$, if system executes with f_l, else $s_l = 0$

Table 3.2: Main Notations

is executed on a different processor and the reliability of τ_i becomes $R_i = 1 - [1 - R_i^o][1 - R_i^d]$, where R_i^d is the reliability of duplication task of τ_i.

3.2.2 Problem Constraints

Frequency Assignment

We consider the three DVFS schemes introduced in Section 1.3. The frequency assignment constraints for TL-DVFS, PL-DVFS and SL-DVFS schemes are given below:

1. `Frequency assignment under TL-DVFS:` Under TL-DVFS, the platform applies DVFS per task, and each task can only be assigned with one frequency level:

$$\sum\nolimits_{l \in \boldsymbol{L}} s_{il} = 1, \ \forall i \in \boldsymbol{N}, \tag{3.1}$$

$$\sum\nolimits_{l \in \boldsymbol{L}} c_{il} = \sigma_i, \ \forall i \in \boldsymbol{N}. \tag{3.2}$$

2. `Frequency assignment under PL-DVFS:` When the platform supports PL-DVFS, each processor can have a single frequency level and the tasks assigned to the processor are

executed with the same frequency:

$$\sum_{l \in \boldsymbol{L}} s_{ml} = 1, \ \forall m \in \boldsymbol{M}, \tag{3.3}$$

3. `Frequency assignment under SL-DVFS:` When the platform supports SL-DVFS, all processors are assigned with the same frequency:

$$\sum_{l \in \boldsymbol{L}} s_{l} = 1. \tag{3.4}$$

Task Duplication Decision

We assume a task is executed successfully if at least one replica is executed without faults [24, 32, 44, 70]. Different replicas of a task are executed on different processors, having a higher probability of correct execution. If all replicas are executed on the same processor and the processor faulty, no execution manages to be correct.

Since all tasks need to be executed with original copy, then $\sigma_i = 1$ when $1 \leq i \leq N$. If $0 < R_i^o \leq R_i^{th}$, the task needs to be duplicated, $\sigma_{N+i} = 1$, else (i.e., $r_i > R_i^{th}$), only the original task is executed, thus, $\sigma_{N+i} = 0$. In order to describe this behaviour, the following Lemma is introduced.

Lemma 1. *Let x and y denote two discrete variables where $0 < x_{\min} \leq x \leq x_{\max} \leq 1$ and $0 < y_{\min} \leq y \leq y_{\max} \leq 1$. Let c denote a binary variable. Given the determination i) if $0 < x \leq y$, $c = 1$, and ii) if $x > y$, $c = 0$, we have $\delta - (1 + \delta)c \leq x - y \leq 1 - c$, where δ is positive small value.*

Proof. Let $C_1 : \delta - (1 + \delta)c \leq x - y$ and $C_2 : x - y \leq 1 - c$. i) If $x < y$, then $x - y < 0$. For C_1, c must be 1. For C_2, c can be either 0 or 1. To satisfy C_1 and C_2 at the same time, we have $c = 1$. If $x = y$, for C_1, c must be 1 due to $x - y = 0$ and $\delta > 0$. For C_2, c can be either 0 or 1. Similarly, we obtain $c = 1$. ii) If $x > y$, for C_1, c can be either 0 or 1. However, c must be 0 in C_2 due to $x - y > 0$. Hence, c must be 0 if $x > y$. □

for instance, considering TL-DVFS scheme, since there are L pairs of voltage/frequency, for the values of task R_i^o, potentially we have $R_i^o \in \{e^{-\varphi_i(f_1)}, \ldots, e^{-\varphi_i(f_L)}\}$. According to Lemma 1, the relationship between $R_i(f_l)$, R_i^{th} and σ_i is linearized as follows:

$$\delta_i - (1 + \delta_i)\sigma_{N+i} \leq \sum_{l \in \boldsymbol{L}} s_{il} e^{-\varphi_i(f_l)} - R_i^{th} \leq 1 - \sigma_{N+i}, \ \forall i \in \boldsymbol{N}. \tag{3.5}$$

Similarly, for PL-DVFS and SL-DVFS schemes, the duplication decisions are formulated as:

$$\delta_i - (1 + \delta_i)\sigma_{N+i} \leq \sum_{m \in \boldsymbol{M}} q_{im} \left(\sum_{l \in \boldsymbol{L}} s_{ml} \right) e^{-\varphi_i(f_l)} - R_i^{th} \leq 1 - \sigma_{N+i}, \ \forall i \in \boldsymbol{N}. \tag{3.6}$$

$$\delta_i - (1+\delta_i)\sigma_{N+i} \leq \sum_{l\in \boldsymbol{L}} s_l e^{-\varphi_i(f_l)} - R_i^{th} \leq 1 - \sigma_{N+i},\ \forall i \in \boldsymbol{N}. \tag{3.7}$$

Task Allocation

We do not consider task migration in this work. For each task τ_i, it is executed on one processor:

$$\sum_{m\in \boldsymbol{M}} q_{im} = \sigma_i,\ \forall i \in \boldsymbol{N}. \tag{3.8}$$

If a task is duplicated, the original and its replica are allocated on different processors [44, 24]:

$$\sum_{m\in \boldsymbol{M}} d_{im} = \sigma_i,\ \forall i \in \boldsymbol{N}, \tag{3.9}$$

$$q_{im} + d_{im} \leq 1,\ \forall i \in \boldsymbol{N},\ \forall m \in \boldsymbol{M}. \tag{3.10}$$

Real-Time Requirement

Tasks (original and duplicated) assigned on processor θ_m should be executed within a common deadline D, thus

$$\sum_{i\in \boldsymbol{N}} q_{im} et_i^o + \sum_{i\in \boldsymbol{N}} d_{im} et_i^d \leq D,\ \forall m \in \boldsymbol{M}, \tag{3.11}$$

3.2.3 Objective Function and Problem Formulation

In this work, the objective function is related to the energy consumption of tasks' execution. We first present in details the problem forumaltion for the TL-DVFS scheme, and then extend to the PL-DVFS and SL-DVFS schemes.

TL-DVFS scheme

Considering TL-DVFS scheme, the energy consumption of task τ_i (original and duplicated) is $E_i = \sum_{l\in \boldsymbol{L}} s_{il} P_l \frac{W_i}{f_l}$, so the total system energy consumption is

$$E_s = \sum_{i\in \boldsymbol{N}} \left(\sum_{l\in \boldsymbol{L}} s_{il} \frac{W_i}{f_l} P_l \right), \tag{3.12}$$

Based on the objective function and the aforementioned problem constraints, the Primal

Problem ($\mathbf{PP-TL}$) considering TL-DVFS is formulated as

$$\mathbf{PP\text{-}TL}: \min_{\boldsymbol{s},\boldsymbol{c},\boldsymbol{q},\boldsymbol{\sigma},\boldsymbol{d}} \sum\nolimits_{i\in\boldsymbol{N}} \left(\sum\nolimits_{l\in\boldsymbol{L}} s_{il}\frac{W_i}{f_l}P_l\right) \tag{3.13}$$
$$\text{s.t.} \begin{cases} (3.1),(3.2),(3.5),(3.8),(3.9),(3.10),(3.11) \\ s_{il}, c_{il}, q_{im}, \sigma_i, d_{im} \in \{0,1\},\ \forall i \in \boldsymbol{N},\ \forall m \in \boldsymbol{M},\ \forall l \in \boldsymbol{L}. \end{cases}$$

Since the nonlinear items exist in Equation (3.11) (i.e., $q_{im}et_i^o$ and $d_{im}et_i^d$), $\mathbf{PP-TL}$ is an Integer Non-linear Programming (INLP) problem, which is difficult to solve optimally. In order to find the optimal solution, as well as to simplify the structure of the problem, we equivalently transform $\mathbf{PP-TL}$ to an MILP problem. By applying **variable replacement** method, the nonlinear variable combinations are replaced equivalently by an MILP formulation. First, we observe that $s_{il} \in \{0,1\}$ and W_i is large enough, one cycle has a negligible impact on the solution. Thus, et_i^o and et_i^d can be relaxed to the continuous variables: $0 \le et_i^o = \sum_{l\in\boldsymbol{L}} s_{il}\frac{W_i}{f_l} \le \overline{T_i}$ and $0 \le et_i^d = \sum_{l\in\boldsymbol{L}} c_{il}\frac{W_i}{f_l} \le \overline{T_i}$, where $\overline{T_i} = \frac{W_i}{f_{min}}$. To linearize $q_{im}et_i^o$ and $d_{im}et_i^d$, we introduce the following lemma:

Lemma 2. *Given two positive constants s_1 and s_2, there are two constraint spaces $P_1 = \{[t,b,x]|t = bx, -s_1 \le x \le s_2, b \in \{0,1\}\}$ and $P_2 = \{[t,b,x]| - bs_1 \le t \le bs_2, t + bs_1 - x - s_1 \le 0, t - bs_2 - x + s_2 \ge 0, b \in \{0,1\}\}$, then $P_1 \rightleftharpoons P_2$.*

Proof. i) $P_1 \rightharpoonup P_2$: According to $t = bx$ and $-s_1 \le x \le s_2$, then $-bs_1 \le t \le bs_2$. Based on $-s_1 \le x \le s_2$ and $b \in \{0,1\}$, then $(b-1)(x-s_2) \ge 0$ and $(b-1)(x+s_1) \le 0$. Therefore, $t - bs_2 - x + s_2 \ge 0$ and $t + bs_1 - x - s_1 \le 0$ hold. ii) $P_1 \leftharpoonup P_2$: When $b = 0$, $t = 0$ and $-s_1 \le x \le s_2$ according to P_2 space definition. When $b = 1$, then $-s_1 \le t = x \le s_2$ from P_2. Thus, $P_1 \rightleftharpoons P_2$. □

Based on Lemma 2, the continuous variables $\alpha_{im} = q_{im}et_i^o$ and $\beta_{im} = d_{im}et_i^d$ are introduced, and Equation (3.11) can be replaced by:

$$\sum\nolimits_{i\in\boldsymbol{N}} \alpha_{im} + \sum\nolimits_{i\in\boldsymbol{N}} \beta_{im} \le D,\ \forall m \in \boldsymbol{M}, \tag{3.14a}$$

$$-\overline{T_i}q_{im} + \alpha_{im} \le 0,\ -et_i^o + \alpha_{im} \le 0,\ \overline{T_i}q_{im} + et_i^o - \alpha_{im} \le \overline{T_i},$$
$$\forall i \in \boldsymbol{N},\ \forall m \in \boldsymbol{M}, \tag{3.14b}$$

$$-\overline{T_i}d_{im} + \beta_{im} \le 0,\ -et_i^d + \beta_{im} \le 0,\ \overline{T_i}d_{im} + et_i^d - \beta_{im} \le \overline{T_i},$$
$$\forall i \in \boldsymbol{N},\ \forall m \in \boldsymbol{M}. \tag{3.14c}$$

Therefore, the primal problem (3.13) is equally reformulated as follows:

$$\textbf{RAFTM-TL}: \min_{\substack{\boldsymbol{s},\boldsymbol{c},\boldsymbol{q},\boldsymbol{d},\boldsymbol{\sigma},\boldsymbol{\beta},\\ \boldsymbol{et^o},\boldsymbol{et^d}}} \sum\nolimits_{i\in\boldsymbol{N}} \sum\nolimits_{l\in\boldsymbol{L}} P_l \frac{W_i}{f_l}(s_{il}+c_{il}) \tag{3.15}$$

$$\text{s.t.} \begin{cases} (3.1),(3.2),(3.5),(3.8),(3.9),(3.10),(3.14a),(3.14b),(3.14c) \\ -\sum\nolimits_{l\in\boldsymbol{L}} \frac{W_i}{f_l} s_{il} + et_i^o = 0,\ \forall i \in \boldsymbol{N}, \\ -\sum\nolimits_{l\in\boldsymbol{L}} \frac{W_i}{f_l} c_{il} + et_i^d = 0,\ \forall i \in \boldsymbol{N}, \\ s_{il}, c_{il}, q_{im}, d_{im}, \sigma_{im}, \beta_{im} \in \{0,1\}. \\ 0 \le et_i^o, et_i^d, \alpha_{im}, \beta_{im} \le \overline{T_i},\ \forall i \in \boldsymbol{N},\ \forall m \in \boldsymbol{M},\ \forall l \in \boldsymbol{L}. \end{cases}$$

Since all the variables (binary and continuous) are coupled linearly with each other, $\textbf{RAFTM}-\textbf{TL}$ is an MILP problem.

Extension to PL-DVFS and SL-DVFS schemes

When supporting PL-DVFS and SL-DVFS schemes, the total energy consumption is given by

$$E_s = \sum\nolimits_{i\in\boldsymbol{N}} \left[\sum\nolimits_{m\in\boldsymbol{M}} q_{im} \left(\sum\nolimits_{l\in\boldsymbol{L}} s_{ml} \frac{W_i}{f_l} P_l \right), \right. \tag{3.16}$$

$$E_s = \sum\nolimits_{i\in\boldsymbol{N}} \left(\sum\nolimits_{l\in\boldsymbol{L}} s_l \frac{W_i}{f_l} P_l \right) + \sum\nolimits_{i\in\boldsymbol{N}} \left(\sum\nolimits_{l\in\boldsymbol{L}} \sigma_i s_l \frac{W_i}{f_l} P_l \right), \tag{3.17}$$

Then, the studied problems under PL-DVFS and SL-DVFS are formulated as:

$$\begin{aligned} \textbf{PP-PL}: \min_{\boldsymbol{s},\boldsymbol{q},\boldsymbol{d},\boldsymbol{\sigma}} & \sum\nolimits_{i\in\boldsymbol{N}} \left[\sum\nolimits_{m\in\boldsymbol{M}} q_{im} \left(\sum\nolimits_{l\in\boldsymbol{L}} s_{ml} \frac{W_i}{f_l} P_l \right) \right. \\ & \left. + \sum\nolimits_{m\in\boldsymbol{M}} d_{im} \left(\sum\nolimits_{l\in\boldsymbol{L}} s_{ml} \frac{W_i}{f_l} P_l \right) \right] \end{aligned} \tag{3.18}$$

$$\text{s.t.} \begin{cases} (3.3),(3.6),(3.8),(3.9),(3.10),(3.11) \\ s_{ml}, q_{im}, d_{im}, \sigma_i \in \{0,1\},\ \forall i \in \boldsymbol{N},\ \forall m \in \boldsymbol{M},\ \forall l \in \boldsymbol{L}. \end{cases}$$

$$\textbf{PP-SL}: \min_{\boldsymbol{s},\boldsymbol{q},\boldsymbol{d},\boldsymbol{\sigma}} \sum\nolimits_{i\in\boldsymbol{N}} \left(\sum\nolimits_{l\in\boldsymbol{L}} s_l \frac{W_i}{f_l} P_l \right) + \sum\nolimits_{i\in\boldsymbol{N}} \left(\sum\nolimits_{l\in\boldsymbol{L}} \sigma_i s_l \frac{W_i}{f_l} P_l \right) \tag{3.19}$$

$$\text{s.t.} \begin{cases} (3.4),(3.7),(3.8),(3.9),(3.10),(3.11) \\ s_l, q_{im}, d_{im}, \sigma_i \in \{0,1\},\ \forall i \in \boldsymbol{N},\ \forall m \in \boldsymbol{M},\ \forall l \in \boldsymbol{L}. \end{cases}$$

PP − PL and **PP − SL** are INLP problems. Similarly we can apply the **variable replacement** method explained above to safely and equivalently transfer into MILP forms. We do not repeat the process here. Except the notations introduced in Table 3.2, the extra notations used in this part are summarized in Table 3.10.

3.2.4 Evaluation

The aim of this section is to compare the proposed optimal task mapping approach (O_RAFTM) with existing task mapping approaches and explore the impact that different DVFS schemes may have in the task mapping decisions. The experiments are based on simulations, where at each experimental set-up the same platform and power model are used, in order to compare the quality of the obtained results among TL-DVFS, PL-DVFS and SL-DVFS schemes. Since the studied problems are formulated as MILP problems, they can be solved using optimization solver tools like CPLEX and Gurobi.

Experimental set-up

Regarding the DVFS, $L = 6$ voltage/frequency levels are used, based on the work of [90] considering 64 nm technology, as depicted in Table 3.3. To obtain realistic inputs for our experiments regarding the WCEC of the tasks, we count the execution cycles and Memory Accesses (MA) of common benchmarks from MiBench suite [91], using Comet simulator, which is based on a a high-level C++ model with 32-bit RISC-V ISA and standard 5-stage pipeline [92]. The sources of timing variability are eliminated to obtain safe and context-independent measurement [93] without interferences (WCEC_{iso}). Then, the WCEC_{inf}, considering worst case interferences from the other processors, is computed. As the contribution of this paper is not WCET estimation, a trivial pessimistic approach is applied: all processors may conflict during a memory access. Thus, the interference cost is given by (M-1)*MA*Main_Memory_Access_Delay (Table 3.3).

The proposed approach (O_RAFTM) is compared to two SoA approaches which also provide optimal solutions: i) the Reliability-Aware Mapping (O_RAM) approach, similar to [62] and "ESRG" algorithm in [24], and ii) the Duplication Mapping approach (O_TDM), always performing task duplication, similar to [32, 24], when the number of replicas is two, or to [37], with 100% task duplication. O_RAM is the typical way to meet the required reliability without replication by adjusting frequency and/or voltage to meet reliability. From a fault tolerant point of view, O_TDM is a very good approach since it always duplicate the tasks.

A large and diverse set of experiments is performed, by tuning:

1. Number of processors ($M = 2, 4$).
2. Size of task set ($N = 10, 20$).

Table 3.3: Platform and benchmark characteristics

l	f_1	f_2	f_3	f_4	f_5	f_6
$\boldsymbol{f_l}$ (GHz)	0.801	0.8291	0.8553	0.8797	0.9027	1.0
$\boldsymbol{v_l}$ (V)	0.85	0.90	0.95	1.00	1.05	1.1
C_{eff}	7.3249	8.6126	10.238	12.315	14.998	18.497
Main Memory Access Delay					200 cycles	

Benchmark	**MA**	$\boldsymbol{WCEC_{iso}}$	$\boldsymbol{WCEC_{inf}}$	
			$M = 2$	$M = 4$
`matmul (int)`	371,957	3,313,958	77,705,358	226,488,158
`matmul (int64)`	507,133	4,055,289	78,446,689	308,335,089
`qsort (int)`	184,089	875,616	75,267,016	111,329,016
`qsort (int64)`	259,553	1,219,854	75,611,254	156,951,654
`qsort (float)`	185,437	1,745,122	76,136,522	113,007,322
`dijkstra`	117,151	766,369	75,157,769	71,056,969
`blowfish`	110,330	3,058,991	77,450,391	69,256,991
`stringsearch`	597,608	13,093,544	87,484,944	371,658,344

WCEC W	$[\mathbf{1}\times10^8, 4\times10^8]$
R^{th}	[0.999, 0.9995]
λ_0, d	$5\times10^{-5}, 3$

3. Platform DVFS scheme (from flexible TL-DVFS to more restricted PL-DVFS and SL-DVFS).
4. Average failure rate ($\lambda_0 = 5 \times 10^{-5}$ faults/sec) with a failure rate constant ($d_0 = 3$) of processor fault model [36].
5. For each experiment, the characteristics of a task are summarized in Table 3.3:
   — WCEC: Based on WCEC_{inf} (Table 3.3), the WCEC of each task is selected within the range $[1 \times 10^8, 4 \times 10^8]$, incorporating the time overhead for frequency (and supply voltage) changes (e.g., 10–150 μs [66, 94]) and the sanity checks at the end of a task [32].
   — Reliability threshold R_i^{th}: Selected within the range $[0.9990, 0.9995]$, considering a typical magnitude 10^{-3} for reliability target [32]. Such a reliability target for a task is inline with safety standards, such as ISO 26262 for automotive systems, DO-178B for avionics systems and IEC 61508 for industrial software systems [62, 24].
   — Global deadline D (from strict D to more relaxed ones, using $D = k \times \frac{N}{M} \times \frac{1}{2}(\frac{C_{max}}{f_{min}} + \frac{C_{max}}{f_{max}})$, with a step of 0.1 for $N = 10$ and 0.2 for $N = 20$ by adjusting k).

Note that the above numbers provide only specific values to problem parameters for experiments, without affecting the problem structure. The approaches are implemented and solved with Gurobi 9.0.2 (MILP solver) on several servers, as hundreds of experiments took place.

To evaluate the behavior of the proposed approach, we compute:

1. Feasibility, i.e., the number of experiments where a solution is found out of the total number of experiments (NE), in each set-up.

2. Energy Consumption (EC) in mJ of the solutions provided by each approach.
3. Reliability Improvement (RI) ($RI = R_i - R_i^{th}$), i.e., the task reliability *above* the task reliability threshold in each set-up, for all approaches.
4. Task duplication, i.e., the average percentage of tasks that O_RAFTM approach decided to duplicate out of the total number of experiments (NE), in each set-up.
5. Computation time (CT), i.e., the average time required for each approach to find a solution out of the total number of experiments (NE), in each set-up.

Note that an approach may fail to find a solution, especially in strict deadlines. In order to fairly compare the energy consumption, reliability improvement and the computation time, we present the average values of the experiments where both compared approaches (i.e., O_RAFTM vs O_RAM, or O_RAFTM vs O_TDM) were able to find a solution. We apply the same approach in all the experiments of this work.

Experimental Results

Feasibility: Fig. 3.1 depicts the feasibility of TL-DVFS scheme for all approaches. Note that as we observed no real difference in the feasibility behavior among TL-DVFS, PL-DVFS and SL-DVFS schemes in the performed experiments, we present only the TL-DVFS feasibility.

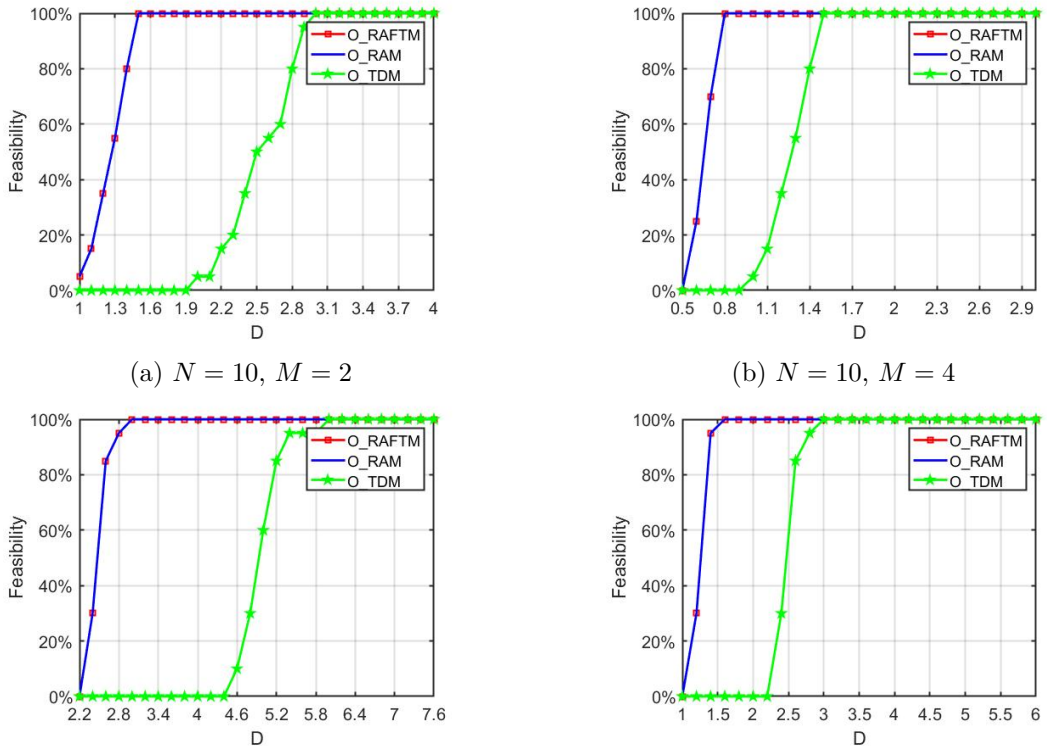
Comparing O_RAFTM and O_TDM feasibilities, O_RAFTM can find solutions in significantly more experiments, because O_RAFTM is not obliged to duplicate every task, whereas O_TDM does. More precisely, when feasibility has not reached 100%, for both approaches, O_RAFTM finds a solution, on average, in 63.5% (Fig. 3.1a), 60.5% (Fig. 3.1b), 65.26% (Fig. 3.1c) and 61.5% (Fig. 3.1d) more experiments than O_TDM. We also observe that TDM finds solutions only after the deadline $D = 1.9$ (Fig. 3.1a), $D = 0.9$ (Fig. 3.1b), $D = 4.4$ (Fig. 3.1c) and $D = 2.2$ (Fig. 3.1d). Moreover, O_RAFTM achieves 100% feasibility in earlier deadlines than O_TDM, i.e., $D = 1.5$ (Fig. 3.1a), $D = 0.8$ (Fig. 3.1b), $D = 3$ (Fig. 3.1c) and $D = 1.6$ (Fig. 3.1d). With increasing number of processors (comparing left and right parts in Fig. 3.1), the capability of O_TDM to find solutions improves, as more processors are available to schedule original and duplication tasks. Furthermore, all approaches achieve 100% feasibility at later deadlines.

Comparing O_RAFTM with O_RAM feasibilities, they have the same feasibility, due to the values of reliability thresholds, i.e., the reliability thresholds can be achieved by executing the original task with a high processor frequency.

Energy consumption: Energy consumption (EC) in mJ achieved by the proposed approach compared to O_RAM and O_TDM, is depicted in Fig. 3.2 and Fig. 3.3 Notice that, when points do not appear in the figures, the corresponding approach found no solutions. Table 3.5

depicts the achieved minimum, average and maximum energy gains compared to O_RAM and O_TDM approaches. The minimum gain is zero (or close to) for strict deadlines for RAM, since O_RAFTM behaves as RAM in this case, executing reliably only the original tasks, i.e., with no duplication. For relaxed deadlines, O_RAFTM behaves as O_TDM, duplicating all tasks. The next paragraphs describe in more details the energy consumption results based on the different tuned parameters.

Number of processors: When the task set size is the same and the number of processors increases, we observe that O_RAFTM, compared to O_RAM, provides solutions with lower energy consumption and the average energy savings are increasing. For instance, on average, gains change from 29.8% to 40.8% (TL-DVFS), 17.2% to 37.0% (PL-DVFS) and 29.5% to 54.9% (SL-DVFS), when M increases from 2 to 4 with $N = 10$. In fact, with processor number increasing, the proposed approach has more available resources, thus it can duplicate more tasks and execute them with low frequency and guarantee the reliability requirements. Energy consumption is thus reduced. On the contrary, RAM should always meet the reliability constraint. Although it uses more processors to execute in parallel the tasks, it cannot reduce their energy consumption, as O_RAFTM can do, due to the fact that the reliability constraints have to be satisfied using



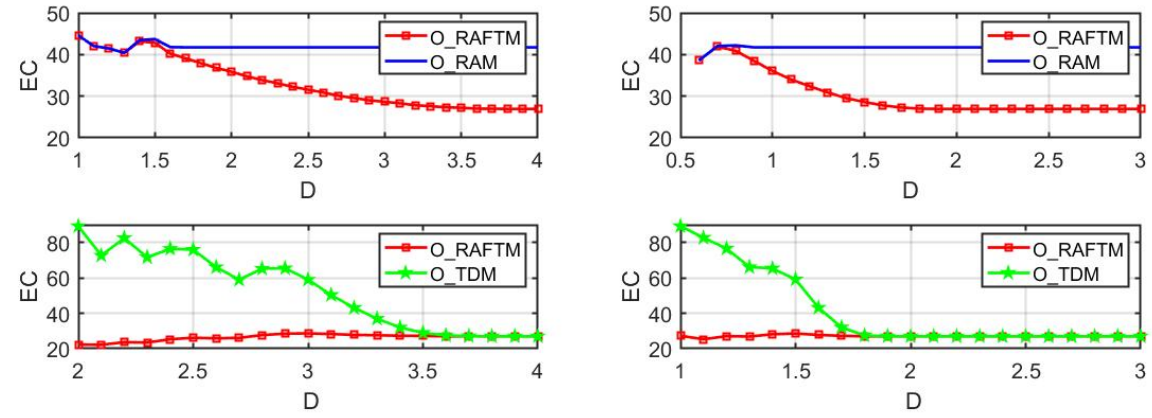
(a) $N = 10$, $M = 2$

(b) $N = 10$, $M = 4$

(c) $N = 20$, $M = 2$

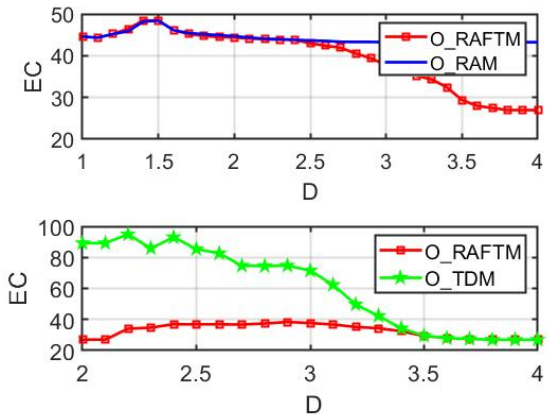
(d) $N = 20$, $M = 4$

Figure 3.1: Feasibility for independent tasks under all DVFS schemes.

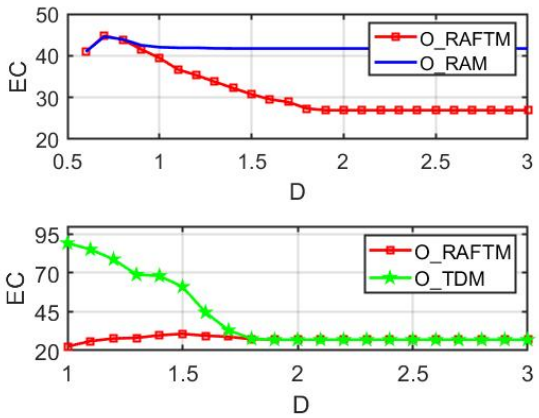


(a) TL-DVFS, $M = 2$

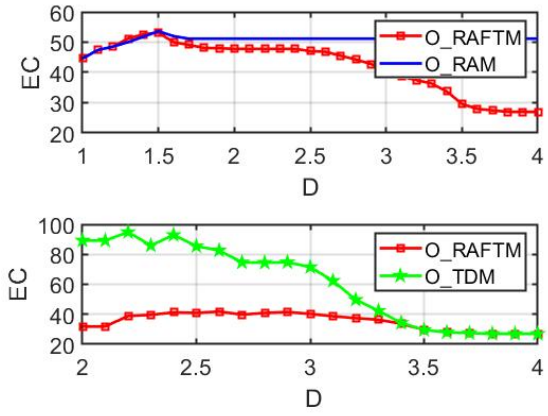
(b) TL-DVFS, $M = 4$



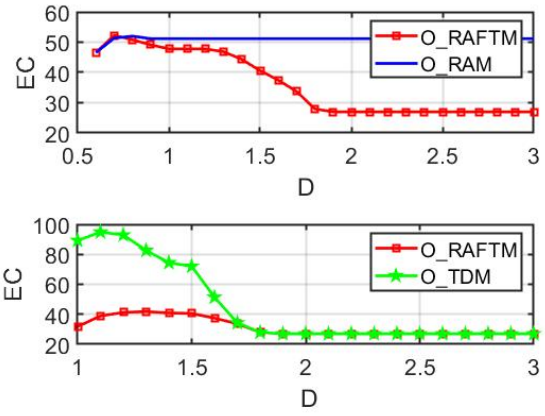
(c) PL-DVFS, $M = 2$



(d) PL-DVFS, $M = 4$

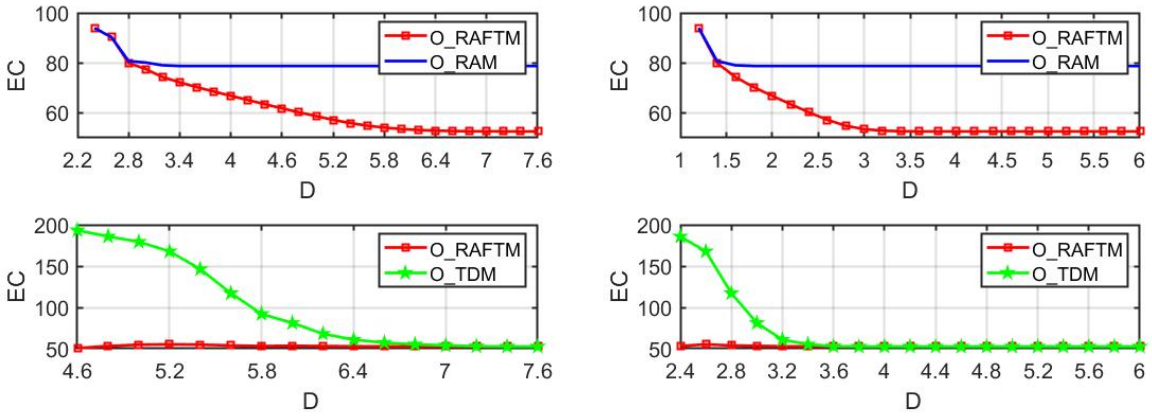


(e) SL-DVFS, $M = 2$



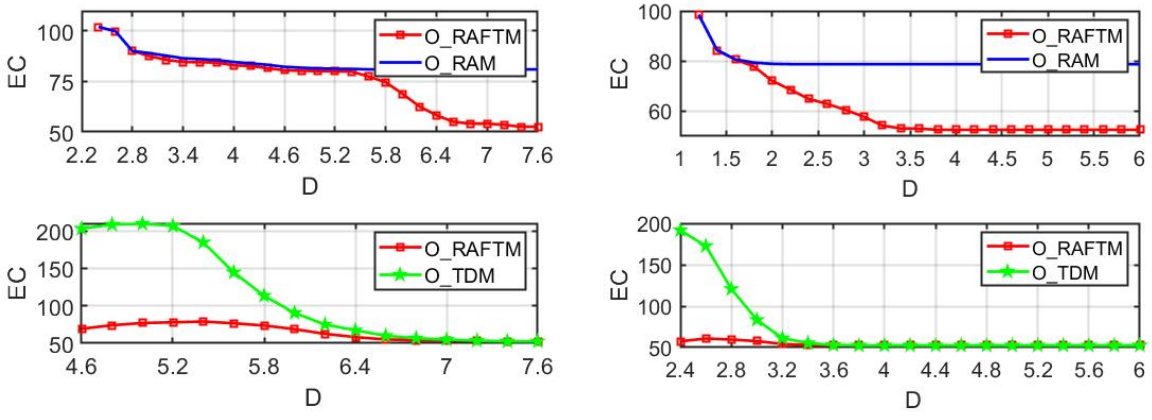
(f) SL-DVFS, $M = 4$

Figure 3.2: Energy consumption (mJ) for independent tasks ($N = 10$) under all DVFS schemes.



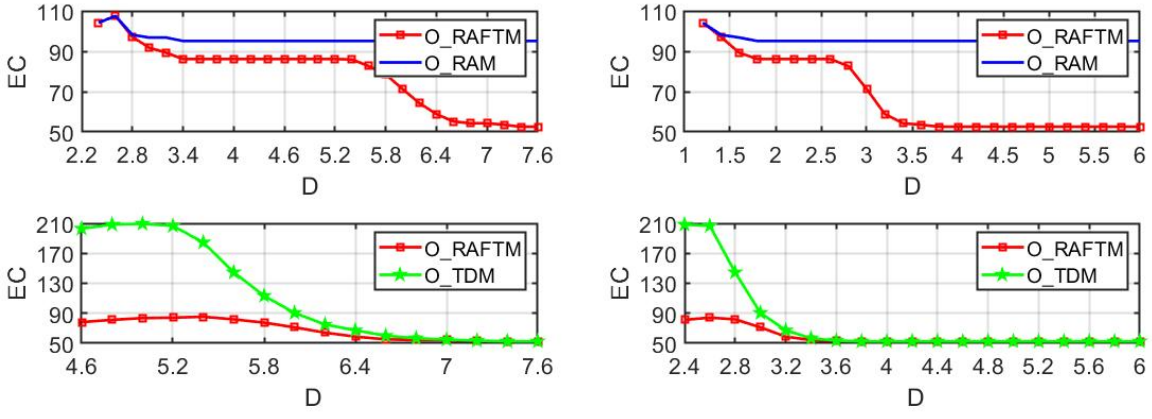
(a) TL-DVFS, $M = 2$

(b) TL-DVFS, $M = 4$



(c) PL-DVFS, $M = 2$

(d) PL-DVFS, $M = 4$



(e) SL-DVFS, $M = 2$

(f) SL-DVFS, $M = 4$

Figure 3.3: Energy consumption (mJ) for independent tasks ($N = 20$) under all DVFS schemes.

only original tasks (no duplication).

Compared to O_TDM, the energy savings of the proposed approach are reduced when the processor number increases. For instance, on average, gains change from 103.9% to 69.8% (TL-DVFS), 77.1% to 51.8% (PL-DVFS) and 61.8% to 34.1% (SL-DVFS), when M increases from 2 to 4 with $N = 10$. Actually, with more processors or with more relaxed deadlines, the proposed approach and O_TDM tend to behave similarly, i.e., duplicating all tasks. However, with less processors, O_RAFTM can execute more tasks using only the original copy, while O_TDM has to duplicate all tasks, thus increasing the energy consumption.

Number of tasks: As expected, with task number increasing, more energy is consumed in all approaches, under all DVFS schemes. When the number of tasks is increased from $N = 10$ to $N = 20$, with the same number of processors, the energy savings remain high for the proposed approach, under all three DVFS schemes. With task number increasing and a given number of processors, it takes a wider deadline region for the energy saving gains to become stable. For instance, for $N = 10$ and TL-DVFS, during the deadline region $D = [1, 3.7]$ (top Fig. 3.2a), the energy gain of O_RAFTM, compared to O_RAM, increases with the deadline becoming more relaxed. After $D = 3.7$, the energy gain becomes stable. For $N = 20$ with same number of processors (Fig. 3.3a), the energy gain increases with increasing deadline during a wider region $D = [2.6, 6.6]$, and after $D = 6.6$, it stays fixed. A similar behavior is observed for O_RAFTM, compared to O_TDM, but with energy saving gain decreasing. Actually, with more tasks, a larger time is required for their execution, until more relaxed deadlines where energy saving gains become stable.

DVFS scheme: When considering only the proposed approach O_RAFTM, the energy consumption is depicted in Table 3.4. The minimal (min.), average (aver.) and maximum (max.) of energy consumption of each group of experiments are obtained among all deadlines. a general observation is that TL-DVFS scheme achieves promising energy savings for O_RAFTM. Actually the frequency assignment in TL-DVFS is the most flexible one, since it is performed per task. PL-DVFS comes next, as it performs frequency assignment per processor. SL-DVFS is the least flexible DVFS scheme, since all tasks are assigned the same frequency. For example, taking $N = 20$ and $M = 4$ into account, the min., aver., and max. of energy consumption is 62.7, 59.2, and 95.2 (mJ) under TL-DVFS, 53.2, 63.3 and 101.4 (mJ) under PL-DVFS, and 55.2, 68.8, and 103.5 (mJ) under SL-DVFS. When comparing the proposed approach O_RAFTM with O_RAM and O_TDM, the energy gains are given in Table 3.5. Comparing the energy gains of TL-DVFS and PL-DVFS schemes, the average energy gains, between O_RAFTM and O_RAM, are decreased, e.g., from 40.8% to 37.0% and 38.8% to 35.4%, for $N = 10$ and $N = 20$, when $M = 4$. Comparing to SL-DVFS, the average energy savings are increased, especially when the number of processors in the platform becomes larger, e.g., from 40.8% to 54.9% and from 38.8% to 52.3%, for $N = 10$ and $N = 20$, when $M = 4$. With the most flexible DVFS scheme,

i.e., TL-DVFS, O_RAM performs a more fine-grained frequency assignment, achieving a lower energy consumption, compared to SL-DVFS. With SL-DVFS, O_RAM is obliged to select a high frequency, in order to meet the highest reliability threshold among the tasks, and thus, all tasks must be executed with this high frequency, leading to large energy consumption. On the contrary, O_RAFTM is able to better exploit frequency assignment, even for the less flexible SL-DVFS scheme. Regarding O_TDM, comparing the average energy gains between TL-DVFS and PL-DVFS and SL-DVFS schemes, we observe a decrease, e.g., from 103.9% (TL-DVFS) to 77.1% (PL-DVFS) and to 61.8% (SL-DVFS), for $N = 10$ and $M = 2$, as the flexibility in deciding frequencies is reduced.

Table 3.4: Min., avg. and max. energy consumption (mJ) of O_RAFTM under all DVFS schemes.

N	M	**TL-DVFS**			**PL-DVFS**			**SL-DVFS**		
		Min.	Aver.	Max.	Min.	Aver.	Max.	Min.	Aver.	Max.
10	**2**	27.31	33.63	43.27	28.69	40.83	47.40	30.41	43.18	47.40
10	**4**	27.31	31.04	39.19	27.85	33.59	45.05	30.41	37.97	46.55
20	**2**	52.67	63.67	93.18	53.87	76.55	103.56	55.17	79.00	103.56
20	**4**	52.67	59.24	95.24	53.24	63.29	101.45	55.17	68.82	103.56

Deadline restriction: From Fig. 3.2 and Fig. 3.3, we observe that the energy gains, between the proposed approach and O_RAM, remain small at strict deadlines. For instance, for $N = 20$ and $M = 2$, the energy gain is smaller than 10% for $D = 2.2$ to $D = 3.4$ in TL-DVFS (Fig. 3.3a) and smaller than 5% for $D = 2.2$ to $D = 5.8$ in PL-DVFS (Fig. 3.3c), and for $D = 2.2$ and $D = 5.6$ in SL-DVFS (Fig. 3.3e). When the deadline is strict, the proposed approach behaves as O_RAM: there is no available time slack, and thus, O_RAFTM assigns high frequencies without applying task duplication. At less strict deadlines, the proposed approach explores any available time slack to duplicate tasks. For instance, for $N = 20$ and $M = 2$, the energy gain reaches its maximum for $D = 6.2$ to $D = 7.6$ in TL-DVFS (Fig. 3.3a), $D = 6.6$ to $D = 7.6$ in PL-DVFS (Fig. 3.3c), and $D = 6.2$ to $D = 7.6$ in SL-DVFS (Fig. 3.3e). The trend is inverted between the proposed approach and O_TDM. When O_TDM can find solutions, the energy gain reaches its maximum at strict deadlines, since the proposed approach can execute original tasks at a high frequency, while O_TDM requires duplication of all tasks. The minimum (0% for most cases) is observed at quite relaxed deadlines, since in this case O_RAFTM and O_TDM behave similarly.

Reliability Improvement: Fig. 3.4 and Fig. 3.5 show the reliability achievements of all approaches and DVFS schemes.

Regarding O_RAFTM, it achieves higher reliability than O_RAM, except in very strict deadlines. Compared to O_TDM, O_RAFTM provides lower reliability for tight deadlines, as

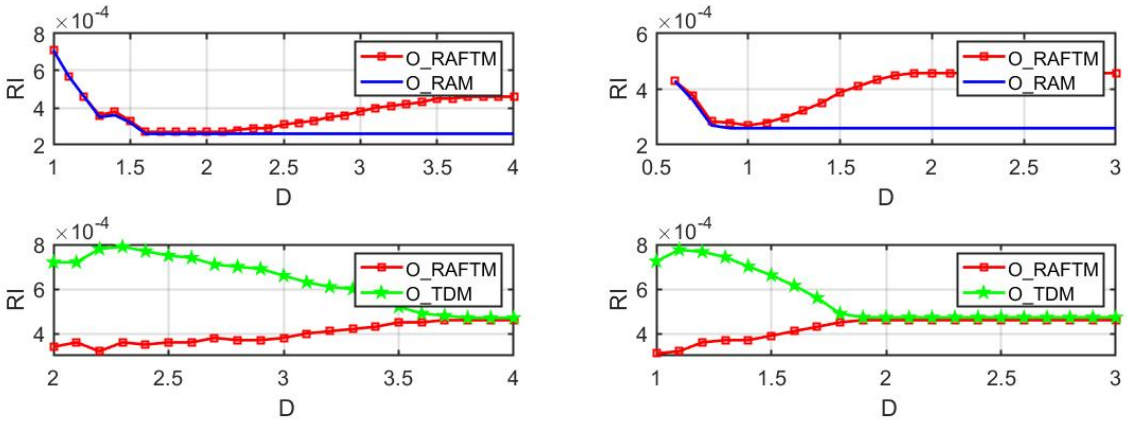
Table 3.5: Min., avg. and max. energy saving gains (%) under all DVFS schemes.

		TL-DVFS			PL-DVFS			SL-DVFS		
N	**M**	**Min.**	**Avg.**	**Max**	**Min.**	**Avg.**	**Max**	**Min.**	**Avg.**	**Max**
O_RAFTM vs O_RAM										
10	2	0	29.8	53.9	0	17.2	59.6	0	29.5	89.4
10	4	0	40.8	53.9	0	37.0	53.9	0	54.9	89.4
20	2	0	31.2	49.7	0	16.9	54.1	0	29.9	81.1
20	4	0	38.8	49.7	0	35.4	49.7	0	52.3	81.1
O_RAFTM vs O_TDM										
10	2	0.1	103.9	298.7	0	77.1	229.8	0	61.8	180.0
10	4	0.1	69.8	296.6	0.1	51.8	295.6	0	34.1	180.0
20	2	0.01	87.3	280.9	0	69.7	258.4	0	55.5	159.8
20	4	0.01	32.9	248.1	0.1	29.6	232.7	0	21.4	157.1

it partially duplicates the task-set. However, as discussed in next section, O_RAFTM can find solutions when O_TDM cannot. When the deadline is not so strict, e.g., for $D = 1.9$ to $D = 3$ in TL-DVFS (Fig. 3.4b) and PL-DVFS (Fig. 3.4d), O_RAFTM achieves the same reliability as O_TDM, since they behave in a similar way.

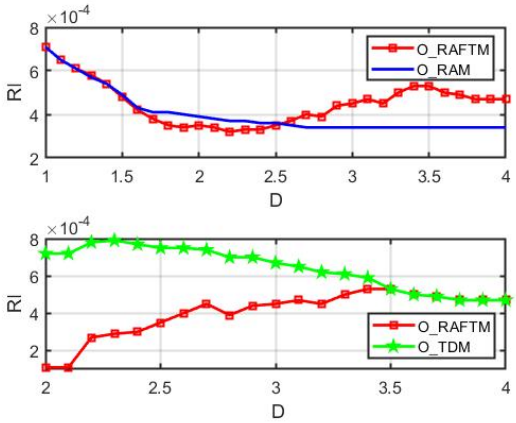
Regarding O_RAM, it has the lowest reliability in TL-DVFS, without violating the reliability constraints. This is because O_RAM has as requirement to meet the reliability threshold. However, for the less flexible in frequency assignment PL-DVFS and SL-DVFS schemes, O_RAM is obliged to select a higher frequency, even for tasks with lower reliability threshold. As a result, the achieved reliability of O_RAM is increased, especially in SL-DVFS scheme where all tasks are executed with same frequency. This can be observed that in third row (SL-DVFS) in Fig. 3.4 and Fig. 3.5, O_RAM generally achieves higher reliability than in first row (TL-DVFS) and second row (PL-DVFS) in Fig. 3.4 and Fig. 3.5. Another observation is that the proposed approach and O_RAM obtain same reliability at strict deadlines, e.g., when $D = 1$ to $D = 1.5$ in TL-DVFS (Fig. 3.4a) and $D = 1$ to $D = 1.4$ in PL-DVFS (Fig. 3.4c), because in strict deadlines O_RAFTM behaves as O_RAM. Regarding O_TDM, when it can find a solution, it provides a high reliability, since it duplicates all tasks. Therefore, changing from TL-DVFS to PL-DVFS and SL-DVFS, has a low impact on the achieved reliability. When a solution is found in strict deadlines, it has usually high reliability, but at the price of high energy consumption, due to the high frequencies required to meet the strict deadlines. For all DVFS schemes in relaxed deadlines, the reliability achieved by the proposed approach and O_TDM is the same, since O_RAFTM and O_TDM behave similarly, when deadlines are relaxed enough.

Task duplication: Fig. 3.6 depicts the percentage of duplicated tasks by O_RAFTM in all DVFS schemes. We remind O_RAM approach does not duplicate tasks (0%) and O_TDM duplicates all tasks (100%). Under same set-up, except the cases of very strict and very relaxed deadlines, O_RAFTM decides the highest task duplication in TL-DVFS, and the least in SL-

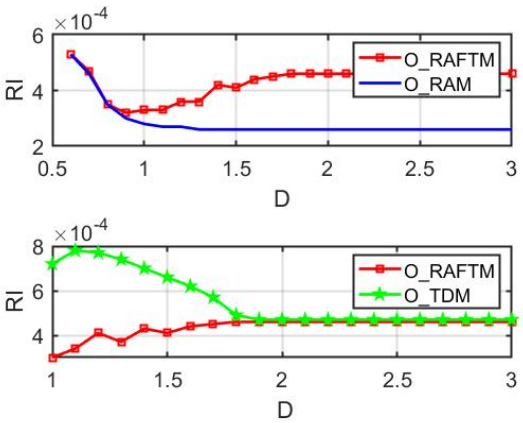


(a) TL-DVFS, $M = 2$

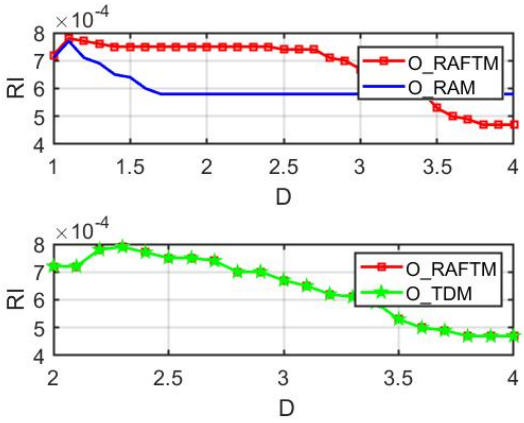
(b) TL-DVFS, $M = 4$



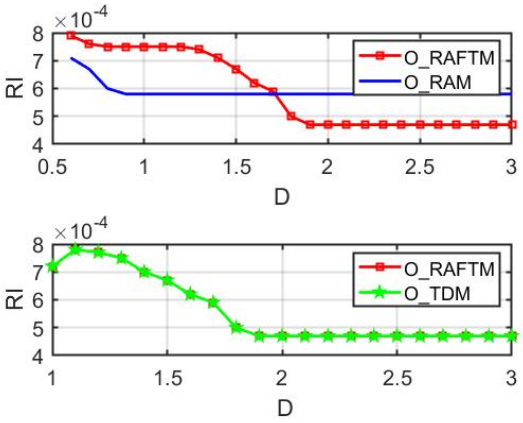
(c) PL-DVFS, $M = 2$



(d) PL-DVFS, $M = 4$

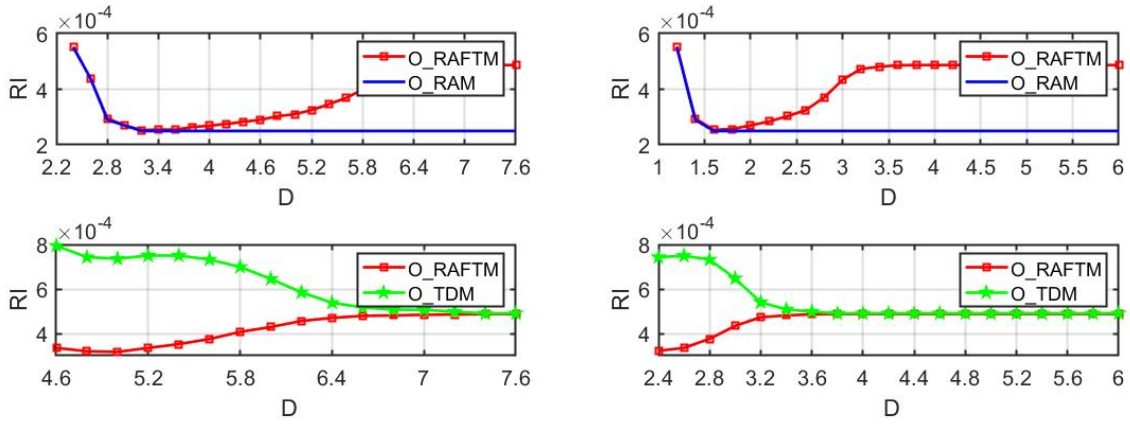


(e) SL-DVFS, $M = 2$

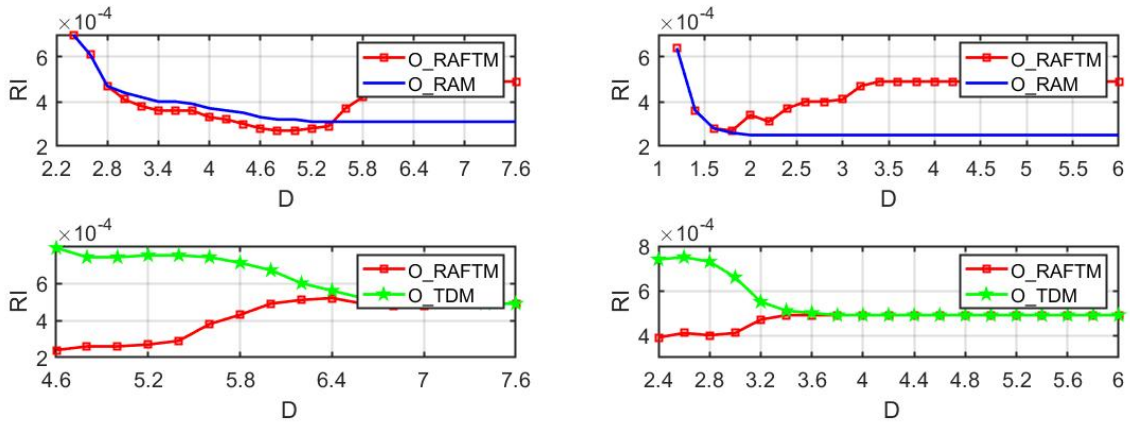


(f) SL-DVFS, $M = 4$

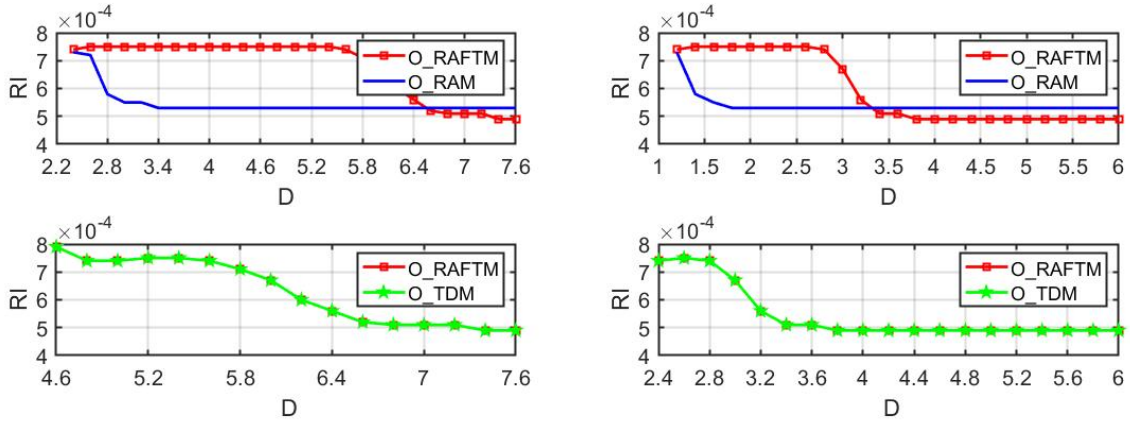
Figure 3.4: Reliability improvement for independent tasks ($N = 10$) under all DVFS schemes.



(a) TL-DVFS, $M = 2$ (b) TL-DVFS, $M = 4$

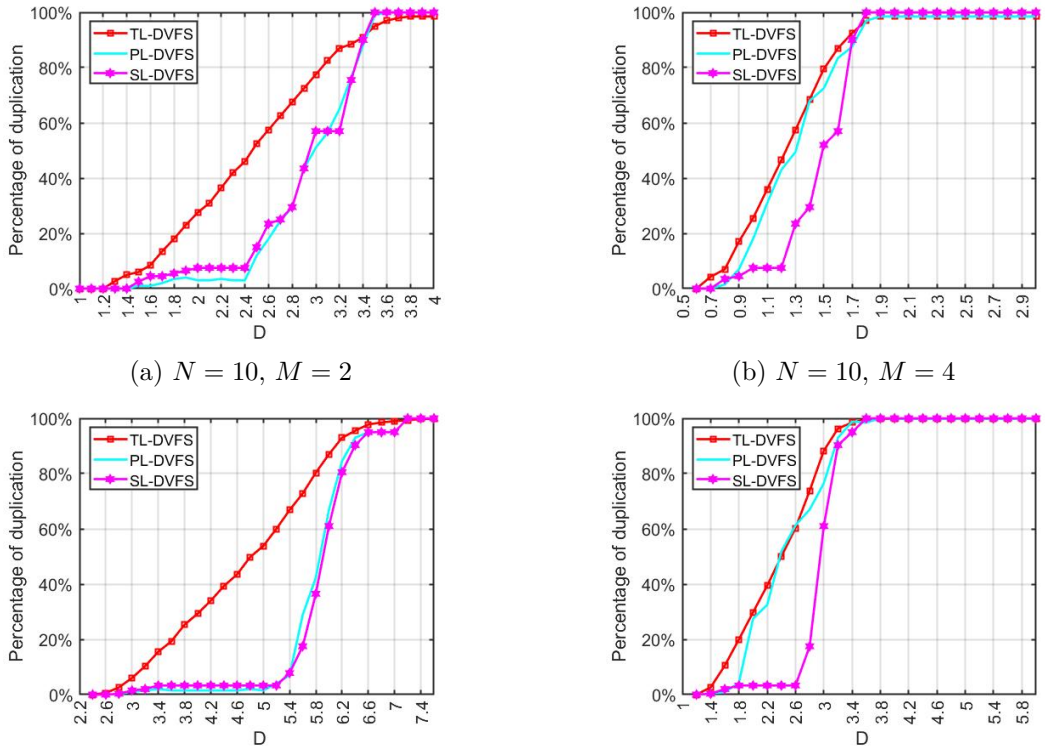


(c) PL-DVFS, $M = 2$ (d) PL-DVFS, $M = 4$



(e) SL-DVFS, $M = 2$ (f) SL-DVFS, $M = 4$

Figure 3.5: Reliability improvement for independent tasks ($N = 20$) under all DVFS schemes.



(a) $N = 10$, $M = 2$ (b) $N = 10$, $M = 4$

(c) $N = 20, M = 2$ (d) $N = 20$, $M = 4$

Figure 3.6: O_RAFTM task duplication under all DVFS schemes.

DVFS. The reasons are the flexibility in assigning task frequencies in different DVFS schemes and the reliability threshold's value. TL-DVFS assigns frequencies per task, providing more opportunities in duplicating task and executing them with low frequency, in order to achieve energy savings. In SL-DVFS, all tasks are executed with same frequency, and thus, the task frequency assignment is restricted. This reduces the possibilities to reduce energy consumption, by task duplication and execution with lower energy consumption. At strict deadlines, for all DVFS schemes, few tasks are duplicated. This is because the execution of duplicated tasks requires time and resources. Thus, it is not possible to duplicate tasks and meet the strict deadlines. When the number of processors is increased, more tasks can be duplicated. When the deadline is more relaxed, O_RAFTM takes advantage of the time slack and duplicates more tasks, using lower frequencies, and thus, achieving less energy consumption. At very relaxed deadlines, the percentage of task duplication does not always reach 100% for TL-DVFS and PL-DVFS, as SL-DVFS does. This occurs when the reliability threshold of a task is satisfied, by executing only the original task and, at same time, its energy consumption is lower than the energy consumption when the task is duplicated.

Table 3.6: Computation time (sec) for independent tasks ($N = 10$, $M = 2$) under all DVFS schemes.

	TL-DVFS			PL-DVFS			SL-DVFS		
D	O_RAFTM	O_RAM	O_TDM	O_RAFTM	O_RAM	O_TDM	O_RAFTM	O_RAM	O_TDM
1.0	0.11	0.06	-	2.48	0.18	-	0.38	0.08	-
1.1	3.81	1.62	-	1.67	0.15	-	0.41	0.09	-
1.2	1.23	0.50	-	1.07	0.11	-	0.29	0.16	-
1.3	1.77	0.08	-	1.27	0.16	-	0.45	0.13	-
1.4	0.82	0.03	-	1.23	0.10	-	0.36	0.14	-
1.5	1.96	0.62	-	1.13	0.09	-	0.28	0.14	-
1.6	0.53	0.01	-	1.01	0.29	-	0.26	0.17	-
1.7	1.82	0.01	-	0.73	0.12	-	0.28	0.17	-
1.8	2.56	0.01	-	0.80	0.06	-	0.26	0.26	-
1.9	1.85	0.01	-	0.92	0.06	-	0.25	0.16	-
2.0	2.38	0.01	1.51	1.20	0.09	0.20	0.27	0.17	0.39
2.1	2.81	0.02	9.98	1.23	0.06	0.49	0.30	0.18	0.47
2.2	2.29	0.02	8.55	1.74	0.04	0.53	0.29	0.15	0.40
2.3	2.95	0.02	13.49	1.94	0.03	1.61	0.32	0.16	0.42
2.4	2.93	0.02	12.81	2.25	0.03	0.83	0.37	0.16	0.45
2.5	3.37	0.02	6.70	3.01	0.02	0.60	0.28	0.15	0.40
2.6	3.96	0.02	14.11	3.88	0.02	0.78	0.32	0.18	0.38
2.7	1.80	0.01	14.49	3.15	0.02	0.66	0.27	0.16	0.38
2.8	1.58	0.01	11.95	3.75	0.02	0.70	0.29	0.14	0.36
2.9	1.19	0.01	11.17	4.68	0.01	0.60	0.26	0.16	0.34
3.0	1.05	0.01	7.76	4.78	0.02	0.51	0.26	0.18	0.33
3.1	0.86	0.01	8.79	4.29	0.02	0.78	0.29	0.18	0.32
3.2	0.49	0.01	6.05	4.41	0.01	0.94	0.32	0.17	0.31
3.3	0.32	0.01	10.87	7.80	0.02	0.72	0.31	0.16	0.31
3.4	0.17	0.01	6.48	5.89	0.01	0.66	0.28	0.15	0.29
3.5	1.03	0.01	2.50	3.51	0.02	0.20	0.25	0.15	0.25
3.6	0.06	0.01	0.73	2.92	0.02	0.09	0.24	0.16	0.24
3.7	0.04	0.01	0.84	1.70	0.02	0.11	0.24	0.15	0.22
3.8-4.0	0.01	0.01	0.01	0.13	0.02	0.06	0.23	0.15	0.22

Table 3.7: Computation time (sec) for independent tasks ($N = 10$, $M = 4$) under all DVFS schemes.

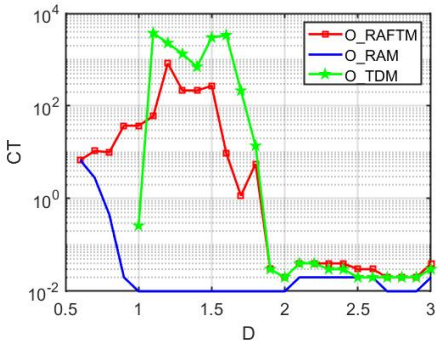
	TL-DVFS			PL-DVFS			SL-DVFS		
D	O_RAFTM	O_RAM	O_TDM	O_RAFTM	O_RAM	O_TDM	O_RAFTM	O_RAM	**O_TDM**
0.5	-	-	-	-	-	-	-	-	-
0.6	6.79	6.84	-	57.62	9.78	-	0.51	0.21	-
0.7	10.72	2.82	-	45.81	8.93	-	0.55	0.25	-
0.8	10.03	0.45	-	76.49	6.39	-	0.43	0.25	-
0.9	37.59	0.02	-	110.45	2.03	-	0.49	0.23	-
1.0	37.44	0.01	0.26	112.50	0.51	2312.10	0.34	0.20	0.94
1.1	60.24	0.01	3755.76	183.62	0.30	110698.16	0.46	0.20	1.17
1.2	830.27	0.01	2283.87	269.00	0.50	4260.27	0.48	0.20	0.91
1.3	217.16	0.01	1335.57	566.26	0.08	38399.02	0.58	0.21	1.05
1.4	215.56	0.01	702.14	424.56	0.03	46466.88	0.67	0.21	0.85
1.5	273.33	0.01	2991.75	438.05	0.03	196016.53	0.56	0.20	0.91
1.6	9.52	0.01	3390.81	191.38	0.03	162484.17	0.48	0.20	0.76
1.7	1.16	0.01	214.75	35026.12	0.03	120531.33	0.50	0.21	0.48
1.8	5.62	0.01	13.83	31.67	0.03	41.78	0.32	0.20	0.30
1.9-3.0	0.03	0.02	0.03	0.58	0.04	0.33	0.29	0.22	0.27

Table 3.8: Computation time (sec) for independent tasks ($N = 20$, $M = 2$) under all DVFS schemes.

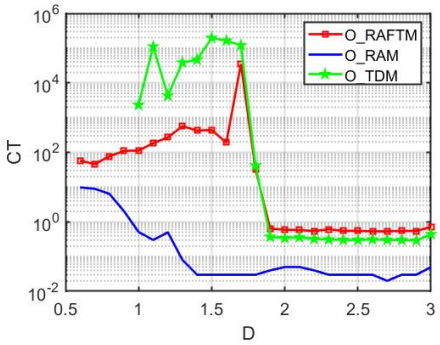
	TL-DVFS			PL-DVFS			SL-DVFS		
D	O_RAFTM	O_RAM	O_TDM	O_RAFTM	O_RAM	O_TDM	O_RAFTM	O_RAM	O_TDM
2.2	-	-	-	-	-	-	-	-	-
2.4	3.38	9.17	-	3.46	0.27	-	NaN	0.09	-
2.6	2.95	26.80	-	6.39	0.80	-	0.66	0.20	-
2.8	2.47	47.22	-	3.10	0.12	-	0.45	0.16	-
3.0	1.42	0.23	-	2.07	0.10	-	0.20	0.17	-
3.2	2.25	1.07	-	2.44	0.10	-	0.11	0.16	-
3.4	2.36	0.01	-	3.78	0.09	-	0.16	0.18	-
3.6	1.59	0.01	-	3.65	0.11	-	0.28	0.17	-
3.8	2.55	0.01	-	4.79	0.27	-	0.11	0.18	-
4.0	2.39	0.01	-	5.50	0.21	-	0.13	0.19	-
4.2	2.43	0.01	-	4.73	0.46	-	0.27	0.16	-
4.4	3.31	0.01	-	4.26	0.13	-	0.34	0.22	-
4.6	4.62	0.01	6.26	4.25	0.81	0.56	0.25	0.15	0.36
4.8	4.87	0.01	7.04	3.45	0.48	0.67	0.23	0.17	0.40
5.0	4.28	0.01	2.44	3.46	0.08	0.79	0.25	0.17	0.41
5.2	4.51	0.01	4.28	7.76	0.05	1.33	0.22	0.20	0.47
5.4	3.85	0.01	4.36	28.74	0.04	1.18	0.23	0.17	0.48
5.6	2.97	0.01	5.21	48.56	0.03	1.50	0.30	0.17	0.47
5.8	2.67	0.01	7.40	73.50	0.03	0.96	0.59	0.17	0.45
6.0	1.51	0.01	6.33	98.67	0.06	0.97	0.20	0.22	0.34
6.2	2.13	0.02	2.94	65.51	0.06	0.47	0.23	0.17	0.34
6.4	0.82	0.02	2.00	42.77	0.04	0.40	0.19	0.15	0.27
6.6	0.24	0.02	0.63	6.06	0.03	0.34	0.31	0.14	0.29
6.8	0.03	0.01	1.13	8.28	0.03	0.79	0.19	0.17	0.31
7.0	0.05	0.01	0.19	11.32	0.03	0.31	0.20	0.21	0.28
7.2	0.02	0.01	0.42	13.92	0.03	0.28	0.36	0.18	0.27
7.4	0.02	0.01	0.02	0.38	0.03	0.27	0.17	0.18	0.29
7.6	0.02	0.01	0.02	0.39	0.03	0.27	0.55	0.27	0.27

Table 3.9: Computation time (sec) for independent tasks ($N = 20$, $M = 4$) under all DVFS schemes.

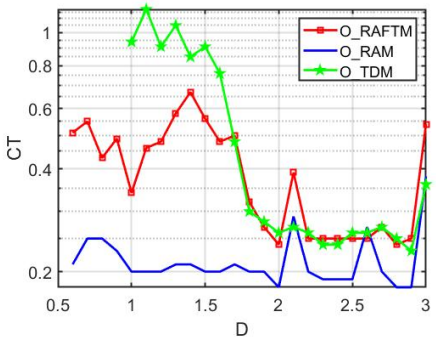
	TL-DVFS			PL-DVFS			SL-DVFS		
D	O_RAFTM	O_RAM	O_TDM	O_RAFTM	O_RAM	O_TDM	O_RAFTM	O_RAM	O_TDM
1.0	-	-	-	-	-	-	-	-	-
1.2	2576.81	63381.17	-	207.57	99.97	-	0.66	0.30	-
1.4	1909.92	57079.83	-	7040.64	172.35	-	0.58	0.17	-
1.6	521.33	791.30	-	1401.37	25.66	-	0.40	0.15	-
1.8	673.87	0.08	-	16611.93	7.37	-	0.26	0.16	-
2.0	399.08	0.07	-	8037.54	0.98	-	0.35	0.18	-
2.2	1237.58	0.04	-	24671.73	0.12	-	0.41	0.17	-
2.4	4356.88	0.05	189.97	2041.74	0.13	17625.22	0.50	0.17	1.38
2.6	165.41	0.05	1193.19	48888.64	0.15	19109.32	0.83	0.24	2.12
2.8	38.65	0.05	1447.58	76090.41	0.12	82873.64	5.30	0.18	3.71
3.0	25.84	0.04	4258.07	86462.91	0.11	8431.17	1.02	0.18	0.88
3.2	3.80	0.05	31173.63	47538.99	0.12	11296.63	0.78	0.23	0.45
3.4	0.19	0.04	4917.95	202.77	0.17	3509.51	0.63	0.33	0.43
3.6	0.87	0.05	11.45	34489.84	0.07	63.94	0.38	0.25	0.25
3.8-6.0	0.04	0.02	0.03	1.64	0.06	0.52	0.28	0.15	0.14



(a) TL-DVFS



(b) PL-DVFS



(c) SL-DVFS

Figure 3.7: Computation time on a logarithmic scale ($N = 10$, $M = 4$) under all DVFS schemes.

Computation Time: Tables 3.6, 3.7, 3.8 and 3.9 provide the average time required to find the solution, for all approaches and DVFS schemes. The '-' means that there is no feasible solution found at the corresponding deadlines. Generally speaking, for the proposed approach, less time is needed to obtain the solutions when deadline is strict or relaxed. However, for intermediate deadlines, more time is required, since O_RAFTM explores the available time slack to decide which, and how many, tasks to be duplicated, without violating constraints, while consuming the least energy. O_TDM is the most time expensive approach, because all tasks are duplicated, increasing the number of tasks to be scheduled, and thus, the time to find the solutions. For O_RAM, as it only executes original tasks, it takes the least time to obtain a solution. However, it provides less energy savings as we observed previously.

Taking $N = 10$ and $M = 4$ as a representative example, the computation time in logarithmic form is depicted in Fig 3.7. We observe that O_RAFTM consumes more time than O_RAM especially at strict deadlines. O_TDM consumes more time than O_RAFTM at strict deadlines (when O_TDM finds a solution,) and consumes same time as O_RAFTM when deadlines are relaxed due to that O_RAFTM and O_TDM behave in similar way in these cases.

With same number of tasks and different number of processors (Table 3.6, Table 3.7, Table 3.8

and Table 3.9), more time is required when more resources are available: in this case, the solver requires more time to optimally decide how to use the extra processors in order to obtain the global optimal solution. One can see that computation time becomes sometimes very high for $M = 4$. We will discuss about a way to cope this issue in Chapter 4.

Regarding different DVFS schemes, for all approaches, more time is consumed to obtain solutions under PL-DVFS compared to TL-DVFS especially when there are more processors, and least time is consumed under SL-DVFS. This is due to the fact that all processors run at same frequency under SL-DVFS as explained above.

3.3 Task Mapping Problem for dependent tasks

In this section, we extend the task mapping problem, with the goal of minimizing the system energy consumption subject to a set of reliability and real-time constraints, for dependent tasks. Similar to the case of independent tasks, the proposed approach concurrently decides: 1) task frequency assignment ($\boldsymbol{s}$), 2) task duplication decision ($\boldsymbol{\sigma}$), 3) task allocation ($\boldsymbol{q}$). Furthermore, the proposed approach also decides the task start time ($\boldsymbol{t^s}$) since we focus on tasks with dependencies. We will describe the proposed approach under TL-DVFS scheme, and then extend it to PL-DVFS and SL-DVFS.

3.3.1 System Model

We adapt the system model by extending the task model presented in the previous section. We consider an application consisting of $\boldsymbol{N}$ frame-based non-preemptive dependent tasks, which is represented by a Directed Acyclic Graph (DAG) $G(\mathbb{V}, \mathbb{E})$, where $\mathbb{V}$ denotes the set of $\boldsymbol{N}$ tasks and $\mathbb{E}$ represents the partial order, corresponding to the precedence constraints among tasks. All the tasks are released at time 0 and have a global deadline D, given by the application frame H. We consider that the global deadline is equal to the application frame, i.e., $D = H$. The ready time of a task is the time instant at which all its predecessors have been completed. If a task has no predecessors (successors), it corresponds to an entry task τ_{entry} (exit task τ_{exit}). A task

Notations	**Definitions**
–	Table 3.2
Parameters	
$\overline{\boldsymbol{N}}$	$\{1, \ldots, 2N\}$, with $2N$ number of tasks
O_{ij}	$O_{ij} = 1$ if task τ_j is dependent on task τ_i, else, $O_{ij} = 0$
H	frame size
Continuous Variables	
t_i^s	the start time of task τ_i

Table 3.10: Main notations for dependent tasks

is ready for execution when all its predecessors have been completed. Similar to Section 3.2.1, each task τ_i is described by a tuple $\{W_i, R_i^{th}\}$, where W_i is the Worst Case Execution Cycles (WCEC) and R_i^{th} is its reliability threshold. To better formulate the studied problem, we define the task set is extended with N duplicated tasks, i.e., $\overline{\boldsymbol{N}} = \{1, \ldots, N, N+1, \ldots, 2N\}$. Tasks $\{\tau_1, \ldots, \tau_N\}$ are original tasks and tasks $\{\tau_{N+1}, \ldots, \tau_{2N}\}$ are duplicated tasks. At this step, we thus assume every task may be duplicated.

3.3.2 Problem Constraints

The problem constraints regarding **frequency assignment**, **task duplication decision**, **task allocation** introduced in Section 3.2.2 are still valid and re-formulated as follows:

$$\sum\nolimits_{l \in \boldsymbol{L}} s_{il} = \sigma_i, \ \forall i \in \overline{\boldsymbol{N}}. \tag{3.20}$$

$$\delta_i - (1+\delta_i)\sigma_{N+i} \leq \sum\nolimits_{l \in \boldsymbol{L}} s_{il} e^{-\varphi_i(f_l)} - R_i^{th} \leq 1 - \sigma_{N+i}, \ \forall i \in \boldsymbol{N}. \tag{3.21}$$

$$\sum\nolimits_{m \in \boldsymbol{M}} q_{im} = \sigma_i, \ \forall i \in \overline{\boldsymbol{N}}. \tag{3.22a}$$

$$q_{im} + q_{N+i,m} \leq 1, \ \forall i \in \boldsymbol{N}, \ \forall m \in \boldsymbol{M}. \tag{3.22b}$$

The **real-time requirement** is modified due to the dependency constraints since the exit task must be executed before the deadline D_i,

$$t_{exit}^s + \sum\nolimits_{l \in \boldsymbol{L}} s_{il} \frac{W_{exit}}{f_l} \leq D. \tag{3.23}$$

For DAG-based dependent tasks, we also need to set the following two constraints:

Task non-overlapping

When task τ_i is executed on processor θ_m with frequency f_l, its execution time is $\sum_{l \in \boldsymbol{L}} s_{il} \frac{W_i}{f_l}$ and, as tasks are executed in a non-preemptive manner, the task end time is $t_i^e = t_i^s + \sum_{l \in \boldsymbol{L}} s_{il} \frac{W_i}{f_l}$. We need to guarantee that a task execution cannot overlap with any other task execution, when assigned to the same processor. The ordering of tasks, assigned on the same processor, is given by a binary matrix $\boldsymbol{w} = [w_{ij}]_{2N \times 2N}$. For any two tasks τ_i and τ_j, when $w_{ij} = 1$, tasks τ_i and τ_j are allocated on the same processor, and τ_i is executed before τ_j. Otherwise, either τ_i and τ_j are allocated on different processors or τ_i is executed after τ_j. Therefore, when both $q_{im} = 1$ and $q_{jm} = 1$, only one of w_{ij} and w_{ji} can be equal to 1, i.e., $w_{ij} + w_{ji} = 1$. Otherwise, both w_{ij} and w_{ji} are equal to 0, i.e., $w_{ij} + w_{ji} = 0$. On this basis, the non-overlapping constraints are

formulated as:

$$t_i^e \leq t_j^s + (2 - q_{im} - q_{jm})H + (1 - w_{ij})H,\ \forall i, j \in \overline{\boldsymbol{N}},\ \forall m \in \boldsymbol{M},\ i \neq j, \tag{3.24}$$

$$w_{ij} + w_{ji} = \sum\nolimits_{m \in \boldsymbol{M}} q_{im} q_{jm},\ \forall i, j \in \overline{\boldsymbol{N}},\ i \neq j, \tag{3.25}$$

Task dependencies

Based on the task graph, we introduce a Task Dependency matrix $\boldsymbol{O} = [o_{ij}]_{2N \times 2N}$. When $o_{ij} = 1$, task τ_i precedes task τ_j and (3.26) becomes $t_j^s \geq t_i^s + \sum_{l \in \mathcal{L}} s_{il} \frac{W_i}{f_l}$. Otherwise, (3.26) is always satisfied.

$$t_j^s + (1 - o_{ij})H \geq t_i^s + o_{ij} \sum\nolimits_{l \in \boldsymbol{L}} s_{il} \frac{W_i}{f_l},\ \forall i, j \in \overline{\boldsymbol{N}},\ i \neq j. \tag{3.26}$$

3.3.3 Objective Function and Problem Formulation

TL-DVFS scheme

The goal is to minimize total energy consumption, thus the Primal Problem ($\mathbf{PP-TL}$) for dependent tasks is formulated as an MINLP:

$$\mathbf{PP\text{-}TL}: \min_{\substack{\boldsymbol{s},\boldsymbol{q},\boldsymbol{\sigma},\\ \boldsymbol{w},\boldsymbol{t^s}}} \sum\nolimits_{i \in \overline{\boldsymbol{N}}} \left(\sum\nolimits_{l \in \boldsymbol{L}} s_{il} \frac{W_i}{f_l} P_l \right) \tag{3.27}$$

$$\text{s.t.} \begin{cases} (3.20), (3.21), (3.22\text{a}), (3.22\text{b}), (3.23), (3.24), (3.25), (3.26) \\ \sigma_i = 1,\ i \in \boldsymbol{N} \\ s_{il}, q_{im}, \sigma_i, w_{ij} \in \{0, 1\},\ 0 \leq t_i^s \leq D, \\ \forall i \in \overline{\boldsymbol{N}},\ \forall m \in \boldsymbol{M},\ \forall l \in \boldsymbol{L}. \end{cases}$$

The **variable replacement** method explained previously in this chapter (see Section 3.2.3) is used to transfer the above MINLP problem into MILP form. Let $h_{ij}^m = q_{im} q_{jm}$ ($i, j \in \overline{\boldsymbol{N}}$, $m \in \boldsymbol{M}$, $i \neq j$) and $h_{ii}^m = 0$, when $i = j$. (3.25) can be equally expressed as

$$w_{ij} + w_{ji} = h_{ij}^m,\ \forall i, j \in \overline{\boldsymbol{N}},\ i \neq j, \tag{3.28}$$

$$-q_{im} + h_{ij}^m \leq 0,\ -q_{jm} + h_{ij}^m \leq 0,\ q_{im} + q_{jm} - h_{ij}^m \leq 1, \\ \forall i, j \in \overline{\boldsymbol{N}},\ \forall m \in \boldsymbol{M},\ i \neq j. \tag{3.29}$$

Then, the above $\mathbf{PP - TL}$ for dependent tasks is reformulated as the following MILP prob-

lem:

$$\textbf{RAFTM-TL}: \min_{\substack{\boldsymbol{s},\boldsymbol{q},\boldsymbol{\sigma},\\ \boldsymbol{w},\boldsymbol{h},\boldsymbol{t^s}}} \sum\nolimits_{i\in\overline{\boldsymbol{N}}} \left(\sum\nolimits_{l\in\boldsymbol{L}} s_{il}\frac{W_i}{f_l}\right) P_l \tag{3.30}$$

$$\text{s.t.} \begin{cases} (3.20), (3.21), (3.22\text{a}), (3.22\text{b}), (3.23), (3.24), (3.28), (3.29), (3.26) \\ \sigma_i = 1,\ i \in \boldsymbol{N} \\ s_{il}, q_{im}, \sigma_i, w_{ij}, h_{ij}^m \in \{0,1\}, 0 \le t_i^s \le D_i, \\ \forall i \in \overline{\boldsymbol{N}},\ \forall m \in \boldsymbol{M},\ \forall l \in \boldsymbol{L}. \end{cases}$$

Extension to PL-DVFS and SL-DVFS schemes

When platforms support PL-DVFS, the constraints of **frequency assignment**, **task duplication decision** and **real-time requirement** introduced in Section 3.3.2 are considered but re-formulated as follows:

$$\sum\nolimits_{l\in\boldsymbol{L}} s_{ml} = 1,\ \forall i \in \boldsymbol{M}. \tag{3.31}$$

$$\delta_i - (1+\delta_i)\sigma_{N+i} \le \sum\nolimits_{m\in\boldsymbol{M}} q_{im}\left(\sum\nolimits_{l\in\boldsymbol{L}} s_{ml}\right) e^{-\varphi_i(f_l)} - R_i^{th} \le 1 - \sigma_{N+i},\ \forall i \in \boldsymbol{N}. \tag{3.32}$$

$$t_{exit}^s + \sum\nolimits_{m\in\boldsymbol{M}} q_{im}\left(\sum\nolimits_{l\in\boldsymbol{L}} s_{ml}\right)\frac{W_{exit}}{f_l} \le D. \tag{3.33}$$

The proposed approach with PL-DVFS can be formulated as

$$\textbf{PP-PL}: \min_{\substack{\boldsymbol{s},\boldsymbol{q},\boldsymbol{\sigma},\\ \boldsymbol{w},\boldsymbol{t^s}}} \sum\nolimits_{i\in\overline{\boldsymbol{N}}} \left[\sum\nolimits_{m\in\boldsymbol{M}} q_{im}\left(\sum\nolimits_{l\in\boldsymbol{L}} s_{ml}\frac{W_i}{f_l}P_l\right)\right] \tag{3.34}$$

$$\text{s.t.} \begin{cases} (3.31), (3.32), (3.22\text{a}), (3.22\text{b}), (3.24), (3.33), (3.28), (3.29), (3.26) \\ \sigma_i = 1,\ i \in \boldsymbol{N} \\ s_{ml}, q_{im}, \sigma_i, w_{ij} \in \{0,1\},\ 0 \le t_i^s \le D_i, \\ \forall i \in \overline{\boldsymbol{N}},\ \forall m \in \boldsymbol{M},\ \forall l \in \boldsymbol{L}. \end{cases}$$

When platforms support SL-DVFS, the constraints of **frequency assignment**, **task duplication decision** and **real-time requirement** introduced in section 3.3.2 are considered but re-formulated as follows:

$$\sum\nolimits_{l\in\boldsymbol{L}} s_l = 1. \tag{3.35}$$

$$\delta_i - (1+\delta_i)\sigma_{N+i} \le \sum\nolimits_{l\in\boldsymbol{L}} s_l e^{-\varphi_i(f_l)} - R_i^{th} \le 1 - \sigma_{N+i},\ \forall i \in \boldsymbol{N}. \tag{3.36}$$

$$t^s_{exit} + \sum\nolimits_{l \in \boldsymbol{L}} s_l \frac{W_{exit}}{f_l} \leq D. \tag{3.37}$$

The proposed approach with SL-DVFS can be formulated as

$$\textbf{PP-SL}: \min_{\substack{\boldsymbol{s},\boldsymbol{q},\boldsymbol{\sigma},\\ \boldsymbol{w},\boldsymbol{t^s}}} \sum\nolimits_{i \in \overline{\boldsymbol{N}}} \left[\sum\nolimits_{m \in \boldsymbol{M}} q_{im} \left(\sum\nolimits_{l \in \boldsymbol{L}} s_{ml} \frac{W_i}{f_l} P_l \right) \right] \tag{3.38}$$

$$\text{s.t.} \begin{cases} (3.35), (3.36), (3.22\text{a}), (3.22\text{b}), (3.24), (3.37), (3.28), (3.29), (3.26) \\ \sigma_i = 1, \ i \in \boldsymbol{N} \\ s_{ml}, q_{im}, \sigma_i, w_{ij} \in \{0,1\}, \ 0 \leq t^s_i \leq D_i, \\ \forall i \in \overline{\boldsymbol{N}}, \ \forall m \in \boldsymbol{M}, \ \forall l \in \boldsymbol{L}. \end{cases}$$

Both $\mathbf{PP-PL}$ and $\mathbf{PP-SL}$ are MINLP problems. The **variable replacement** method explained in section 3.2.3 is used to equivalently transfer the above MINLP problem into MILP forms $\mathbf{RAFTM-PL}$ and $\mathbf{RAFTM-SL}$ and we do not repeat the process here.

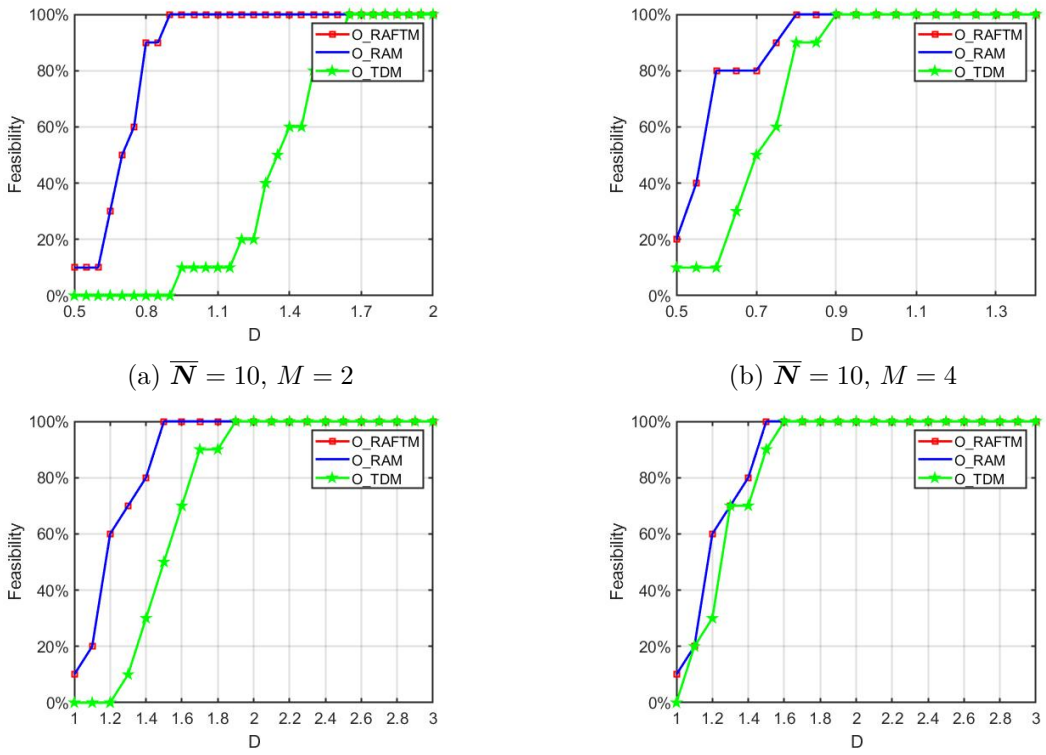
3.3.4 Evaluation

In this section, we evaluate the performance of the proposed approach considering dependent task model compared to the two SoA approaches O_RAM and O_TDM introduced in section 3.2.4. The results are provided with optimal solutions using Gurobi solver. We considered two set-ups in this evaluation section. In the first set-up we explore the impact of the different DVFS schemes, when the reliability constraints can be always met with the higher frequency of the platform, similar to the previous section. In the second set-up, we evaluate the behavior of the proposed approaches when the reliability constraints cannot always be met with the higher frequency of the platform, under the TL-DVFS scheme.

Experimental set-up: Reliability constraints always met

In this first set-up, the platform characteristics are the same as presented in section 3.2.4 in Table 3.3. A large and diverse set of experiments is performed, by tuning the:

1. Number of processors ($M = 2, 4, 6$).
2. Size of task set ($\overline{\boldsymbol{N}} = 10, 20$).
3. Platform DVFS scheme (TL-DVFS, PL-DVFS, and SL-DVFS).
4. For a task set size, 10 experiments are performed. For each experiment, the task characteristics (W_i and R^{th}_i) are selected as in section 3.2.4. The deadline D is tuned a with a step of 0.05 for $\overline{\boldsymbol{N}} = 10$ and 0.1 for $\overline{\boldsymbol{N}} = 20$ by adjusting k).



(a) $\overline{\boldsymbol{N}} = 10$, $M = 2$

(b) $\overline{\boldsymbol{N}} = 10$, $M = 4$

(c) $\overline{\boldsymbol{N}} = 20$, $M = 4$

(d) $\overline{\boldsymbol{N}} = 20$, $M = 6$

Figure 3.8: Feasibility for dependent tasks under all DVFS schemes.

Similar to the independent task section, the approaches are implemented and solved with Gurobi 9.0.1 (MILP solver) on several servers, as hundreds of experiments took place. We present a subset of the experiments (TL-DVFS, PL-DVFS and SL-DVFS for $\overline{\boldsymbol{N}} = 10$, $M = 2$ and $M = 4$ and $\overline{\boldsymbol{N}} = 20$, $M = 4$ and $M = 6$.

As we done for independent tasks, to evaluate the approaches, the metrics of Feasibility, Energy consumption (EC), Reliability Improvement (RI), Duplication percentage and Computation time (CT) are considered.

Experimental results: Reliability constraints always met

Feasibility: The feasibility of O_RAFTM, O_RAM and O_TDM is depicted in Fig. 3.8, for $\overline{\boldsymbol{N}} = 10$ with $M = 2$ and $M = 4$ (Fig. 3.8a and Fig. 3.8b) and for $\overline{\boldsymbol{N}} = 20$ with $M = 4$ and $M = 6$ (Fig. 3.8c and Fig. 3.8d).

Regarding the comparison with O_TDM, O_RAFTM is able to find solutions in significantly more experiments than O_TDM, since O_RAFTM is not obliged to duplicate reliable tasks, compared to O_TDM. When feasibility is not 100% for both approaches, O_RAFTM can find a solution, on average, in 61%, 30%, 33% and 10% more experiments than O_TDM (Fig. 3.8).

As an example, when $k = 1$, $k = 1.1$, $k = 1.2$ ($\overline{\boldsymbol{N}} = 20$ and $M = 4$), O_TDM cannot find any solution. As the number of processors is increased, e.g., from $M = 4$ to $M = 6$, the capability of O_TDM to find solutions improves, as more cores are available to schedule the duplicated tasks, until approaching the O_RAFTM feasibility with 6 processors. O_RAFTM achieves 100% feasibility in earlier deadlines than O_TDM, i.e., $k = 0.8$ ($\overline{\boldsymbol{N}} = 10$ and $M = 4$) and $k = 1.5$ ($\overline{\boldsymbol{N}} = 20$ and $M = 4$, $\overline{\boldsymbol{N}} = 20$ and $M = 6$). When the deadline becomes relaxed (and O_TDM can *always* find solutions), i.e., $k = 0.9$ ($\overline{\boldsymbol{N}} = 10$ and $M = 4$), $k = 1.9$ ($\overline{\boldsymbol{N}} = 20$ and $M = 4$), and $k = 1.6$ ($\overline{\boldsymbol{N}} = 20$ and $M = 6$), O_RAFTM has a similar behaviour with O_TDM, since it decides to duplicate the majority of the tasks, achieving similar gains, as explained in the next section.

Regarding the comparison with O_RAM, in this experimental set-up, O_RAFTM and O_RAM have the same feasibility, since the reliability constraints can always be met by executing only the original task with a high processor frequency. Note that we explore the difference of feasibility between O_RAFTM and O_RAM in the second experimental set-up, where high frequencies cannot always satisfy reliability thresholds. When the number of cores increases, feasibility behaviour of O_RAFTM and O_RAM is not changed, due to the dependencies of the task graph.

Energy consumption: The energy consumption (EC) in mJ of O_RAM and O_TDM, compared to the proposed approach, is depicted in Fig. 3.9 and Fig. 3.10. The minimum, average and maximum gains are depicted in Table 3.11. Note that, the minimum gain is 0 between O_TDM and O_RFTAM, when the deadlines are less strict. When deadline is relaxed, O_RFTAM performs duplication for all tasks, as O_TDM. Compared to O_RAM, we observe a minimum gain of 0, only in very few strict deadlines, either with a high number of tasks while few processors or due to SL-DVFS, where the frequency assignment is very restricted. In the strict deadlines, O_RFTAM and O_RAM have a similar behavior: applying a high frequency to meet the timing constraint, thus no duplication is possible. From the obtained results, we can make the following general observations:

— When the number of tasks is increased from $\overline{\boldsymbol{N}} = 10$ to $\overline{\boldsymbol{N}} = 20$ with the same number of cores (from Fig. 3.9b to Fig. 3.10a, from Fig. 3.9d to Fig. 3.10c, and from Fig. 3.9f to Fig. 3.10e), the energy savings remain high for the proposed O_RFTAM approach, compared to O_RAM and O_TDM. We observe a slight decrease in average savings regarding O_RAM in TL-DVFS and PL-DVFS schemes, due to the fact that O_RFTAM behaves as O_RAM in very strict deadlines. On the one hand, O_RAM consumes more energy compared to O_RFTAM, as Table 3.11 shows. On the other hand, O_RFTAM is able to find solutions in more cases, compared to O_TDM, especially for strict deadlines, as Fig. 3.8 shows.

— When the number of cores increases from $M = 4$ to $M = 6$, with the same number of tasks

$\overline{N} = 20$, (from Fig. 3.10a to Fig. 3.10b, from Fig. 3.10c to Fig. 3.10d, and from Fig. 3.10e to Fig. 3.10f), the energy savings of O_RFTAM compared to O_RAM are enlarged, on average, to 47.3% (TL-DVFS), 45.7% (PL-DVFS), and 72.3% (SL-DVFS). In fact, when more processors are available, our proposed approach has more freedom to use the available processors when performing the task mapping, minimizing the total energy consumption.

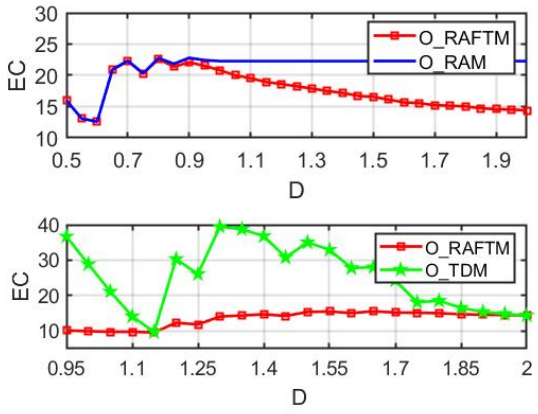
— Among different DVFS schemes, we observed that SL-DVFS has a higher impact on the observed gains of O_RFTAM vs O_RAM, compared to the impact it has on the observed gains of O_RFTAM vs O_TDM, as the number of tasks and cores increases. When the supported DVFS scheme is flexible, O_RAM performs a more fine-grained assignment, achieving a lower energy consumption. However, when SL-DVFS is supported, O_RAM is obliged to select a high frequency in order to meet the highest reliability threshold of the tasks and executes all tasks with this high frequency, causing large energy consumption. On the other hand, the proposed O_RFTAM can exploit task duplication, applying a lower frequency, and thus, reducing the energy consumption, when time slack is available for relaxed deadline cases.

Table 3.11: Min, avg. and max energy gains (%) for dependent tasks under all DVFS schemes.

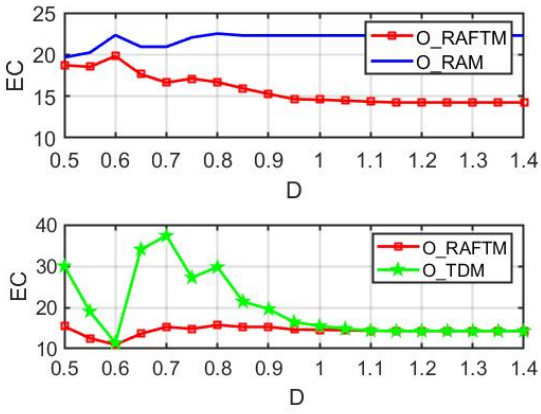
		TL-DVFS			**PL-DVFS**			**SL-DVFS**		
$\overline{N}$	**M**	**Min**	**Avg.**	**Max**	**Min**	**Avg.**	**Max**	**Min**	**Avg.**	**Max**
O_RAFTM vs O_RAM										
10	2	0	23.54	54.94	0	10.72	52.33	0	15.49	73.6
10	4	5.02	40.69	56.35	1.81	35.49	53.44	0	43.23	80.17
20	4	1.16	41.53	59.98	0	37.11	59.98	0	58.34	105.7
20	6	9.76	47.35	59.98	8.63	45.71	59.61	0	72.29	105.7
O_RAFTM vs O_TDM										
10	2	0	91.93	261.83	0	79.21	218.5	0	66.35	170.72
10	4	0	37.27	149.93	0	36.68	149.04	0	37.6	137.75
20	4	0	33.55	123.05	0	37.46	140.71	0	30.02	130.32
20	6	0	17.94	90.86	0	19.83	114.77	0	22.98	113.99

With a more detailed observation, when both O_RFTAM and O_TDM approaches can find solutions, O_RFTAM provides a solution that consumes significantly less energy. More precisely, when $\overline{N} = 20$ and $M = 4$, for TL-DVFS we observe a maximum gain of 123.0%, with an average gain of 33.5%, for PL-DVFS, we observe a maximum gain of 140.7% with an average gain of 37.5%, and, for SL-DVFS, there is a maximum gain 130.3%, with an average gain of 30.0%.

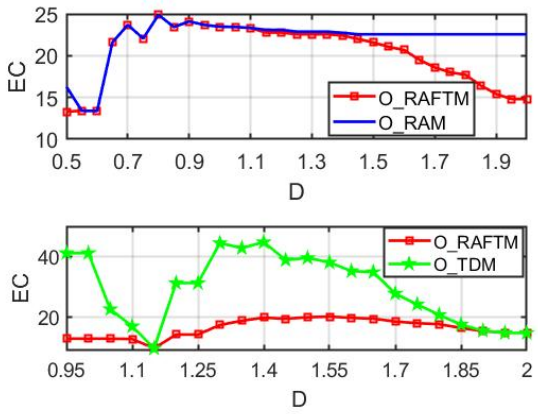
Furthermore, O_RFTAM finds a solution, that consumes less energy than O_RAM, since O_RFTAM can duplicate some tasks in order to use lower frequencies, reducing energy consumption. When the deadline is not so strict (part of graph with almost flat average gain in the sub-plot of Fig. 3.9 and Fig. 3.10), the highest energy gains are observed. More precisely, when $M = 4$, we observe an average gain of 40.7% (TL-DVFS), 35.5% (PL-DVFS), and 43.2%



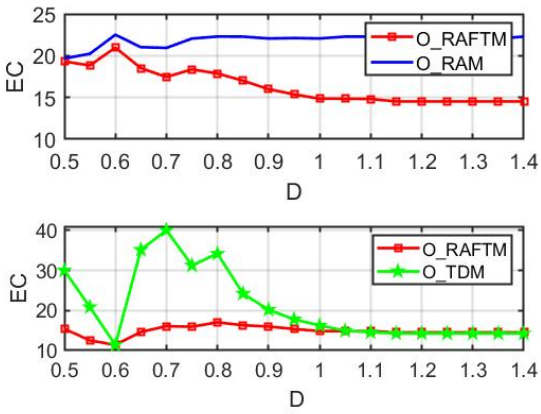
(a) TL-DVFS, $M = 2$



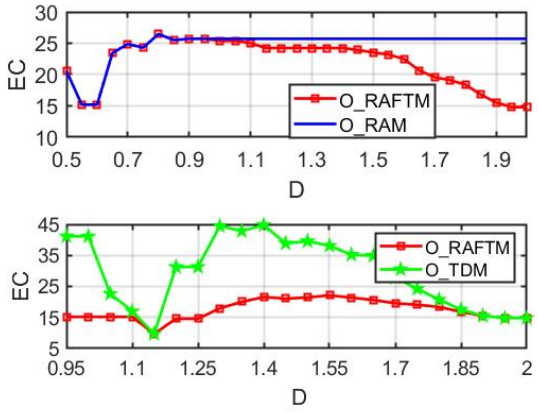
(b) TL-DVFS, $M = 4$



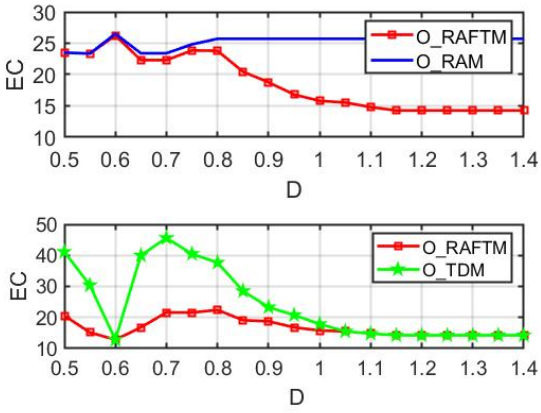
(c) PL-DVFS, $M = 2$



(d) PL-DVFS, $M = 4$

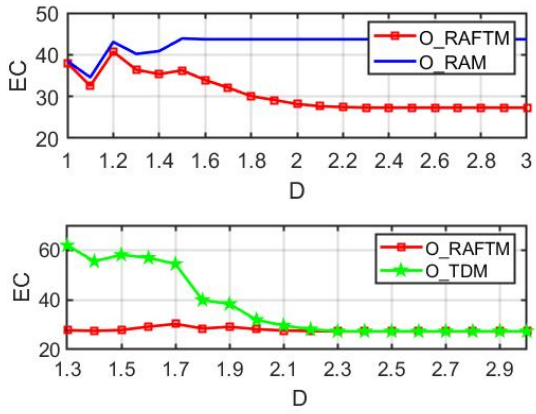


(e) SL-DVFS, $M = 2$

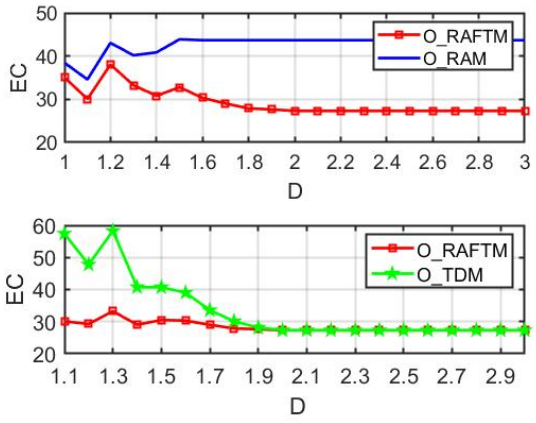


(f) SL-DVFS, $M = 4$

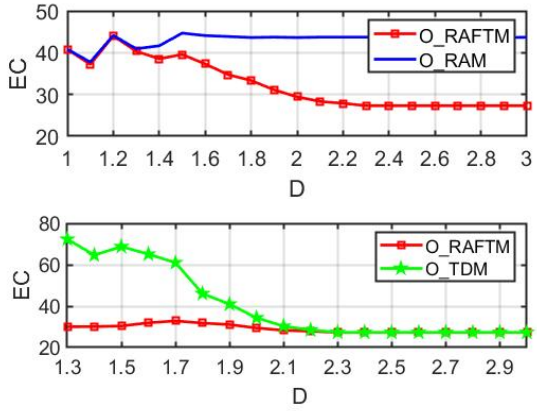
Figure 3.9: Energy consumption (mJ) for dependent tasks ($\overline{\boldsymbol{N}} = 10$) under all DVFS schemes.



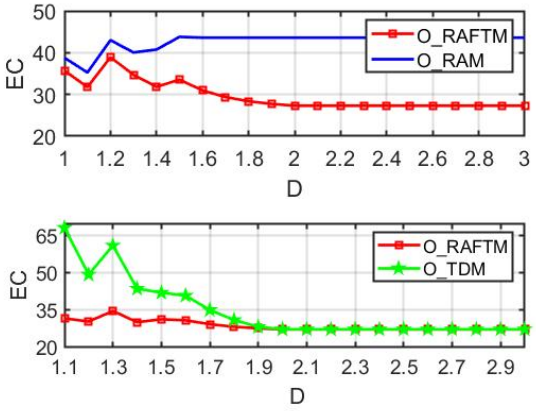
(a) TL-DVFS, $M = 4$



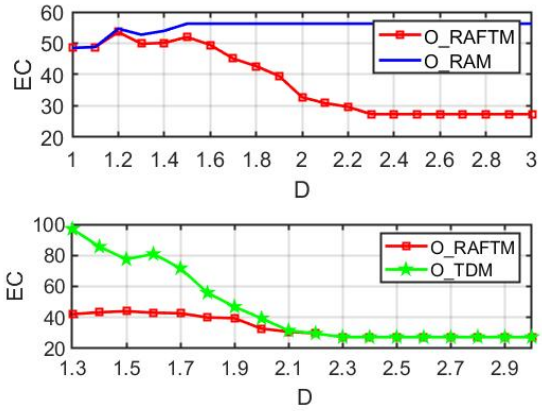
(b) TL-DVFS, $M = 6$



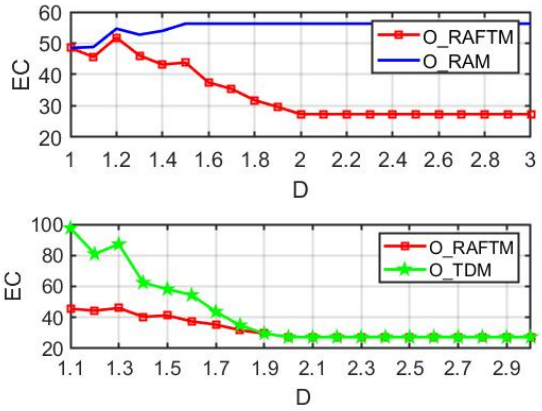
(c) PL-DVFS, $M = 4$



(d) PL-DVFS, $M = 6$



(e) SL-DVFS, $M = 4$



(f) SL-DVFS, $M = 6$

Figure 3.10: Energy consumption (mJ) for dependent tasks ($\overline{\boldsymbol{N}} = 20$) under all DVFS schemes.

(SL-DVFS) for $\overline{\boldsymbol{N}} = 10$ and an average gain of 41.5% (TL-DVFS), 37.1% (PL-DVFS), and 58.3% (SL-DVFS) for $\overline{\boldsymbol{N}} = 20$. When $M = 6$ and $\overline{\boldsymbol{N}} = 20$, the observed average gain is 47.3% (TL-DVFS), 45.7% (PL-DVFS), and 72.3% (SL-DVFS). For very strict deadlines, the energy gains of O_RFTAM and O_RAM are similar; O_RFTAM behaves as O_RAM, since there is no available time slack, and thus, O_RFTAM assigns high frequencies, without task duplication. We observe minimum gains when $k = 0.5$ ($\overline{\boldsymbol{N}} = 10$) and $k = 1$ ($\overline{\boldsymbol{N}} = 20$). When SL-DVFS and less cores are used ($M = 4$), O_RFTAM cannot exploit duplication and use different frequencies to reduce energy consumption, and thus, it behaves as RAM. When time slack exists, O_RFTAM can take advantage of it and duplicate some tasks, if energy gains can be achieved.

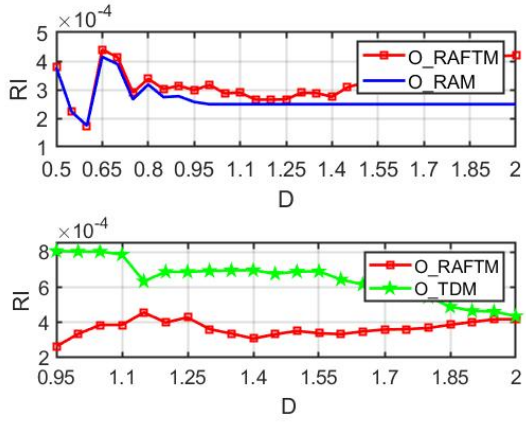
Reliability Improvement: We also compare the reliability improvement with regard to the reliability threshold in Fig. 3.11 and Fig. 3.12, for the three DVFS schemes and for $\overline{\boldsymbol{N}} = 10$ with $M = 2$ and $M = 4$ and $\overline{\boldsymbol{N}} = 20$ with $M = 4$ and $M = 6$. Generally:

— Regarding O_RAM, when TL-DVFS is considered, it has the lowest reliability improvement. This is because O_RAM only requires to meet the reliability threshold in order to find a solution. However, when PL-DVFS and SL-DVFS are considered, O_RAM is obliged to select a higher frequency, even for tasks with lower reliability threshold, due to the restrictions in the DVFS schemes. As a result, the reliability improvement of O_RAM is increased, when the DVFS schemes are more restricted. Note that, when the number of tasks is increased in SL-DVFS, O_RAM is obliged to always select high frequencies to meet the highest reliability threshold, among all tasks, and then, use this frequency for task execution, even if time slack exists.

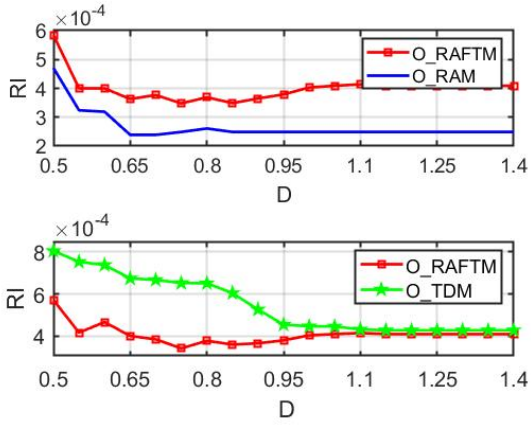
— Regarding O_RAFTM, it provides higher reliability improvements than O_RAM, for all deadlines. For tight deadlines, it provides lower reliability improvements than O_TDM, as it partially duplicates the task set. However, as discussed in the previous section, O_RAFTM is more capable of finding solutions compared to O_TDM, and with significantly reduced energy consumption. When the deadline is not so strict, it provides the same reliability improvements as O_TDM, since they behave in a similar way with duplication.

— Regarding O_TDM, since it always duplicates the tasks, it provides a high reliability improvement, if it can find a solution. Therefore, going from TL-DVFS to PL-DVFS and SL-DVFS, does not have a high impact on the reliability improvement. Moreover, when a solution is found in strict deadlines, it has typically significant reliability improvement, at the price of large energy consumption, since high frequencies are required to meet the strict timing constraints.

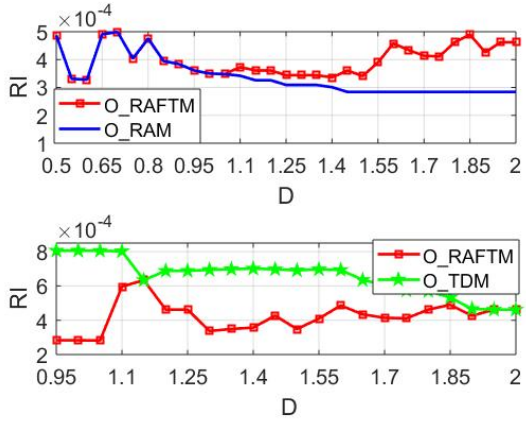
Task duplication: Fig. 3.13 depicts the task duplication of the proposed approach. We remind O_RAM approach does not apply duplication for any task (0%) and O_TDM duplicates all tasks



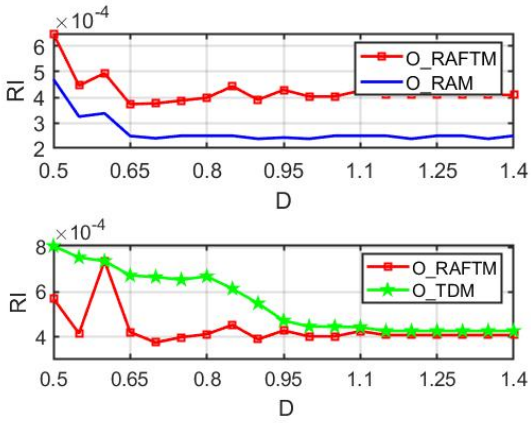
(a) TL-DVFS, $M = 2$



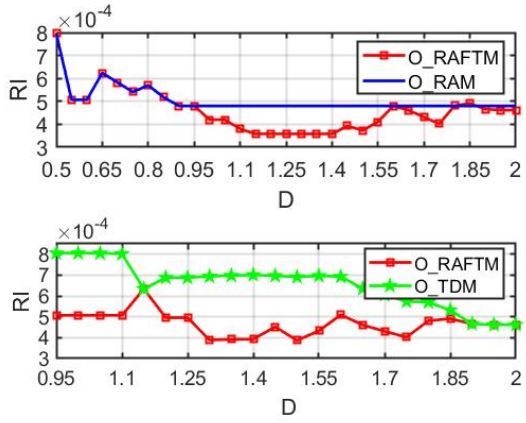
(b) TL-DVFS, $M = 4$



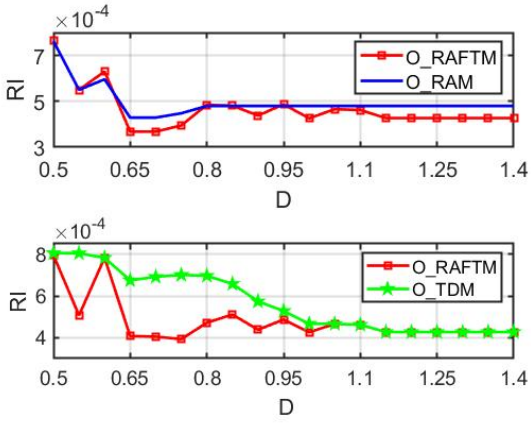
(c) PL-DVFS, $M = 2$



(d) PL-DVFS, $M = 4$

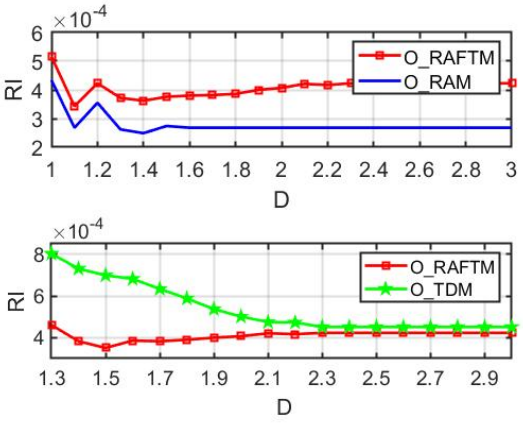


(e) SL-DVFS, $M = 2$

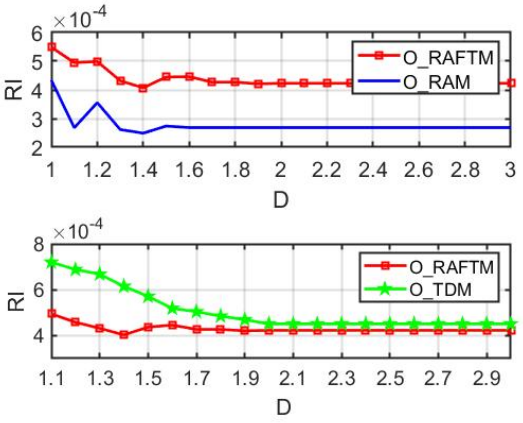


(f) SL-DVFS, $M = 4$

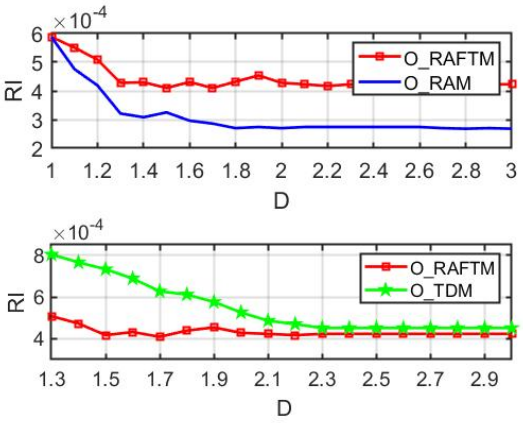
Figure 3.11: Reliability improvement for dependent tasks ($\overline{\boldsymbol{N}} = 10$) under all DVFS schemes.



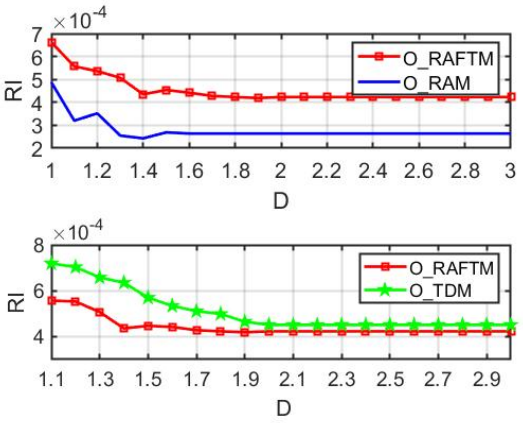
(a) TL-DVFS, $M = 4$



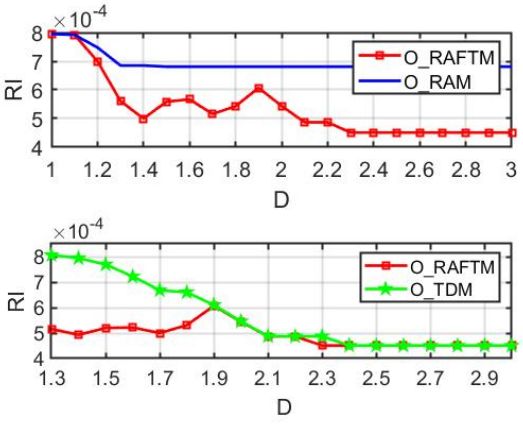
(b) TL-DVFS, $M = 6$



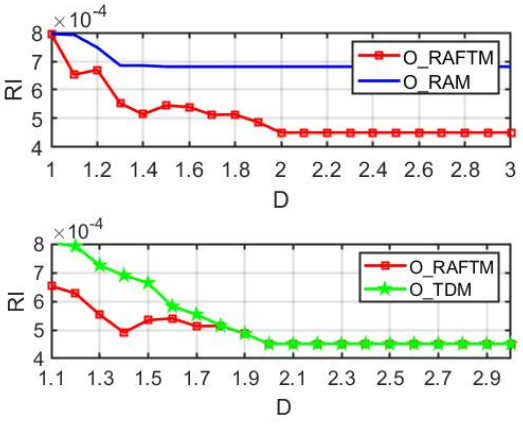
(c) PL-DVFS, $M = 4$



(d) PL-DVFS, $M = 6$



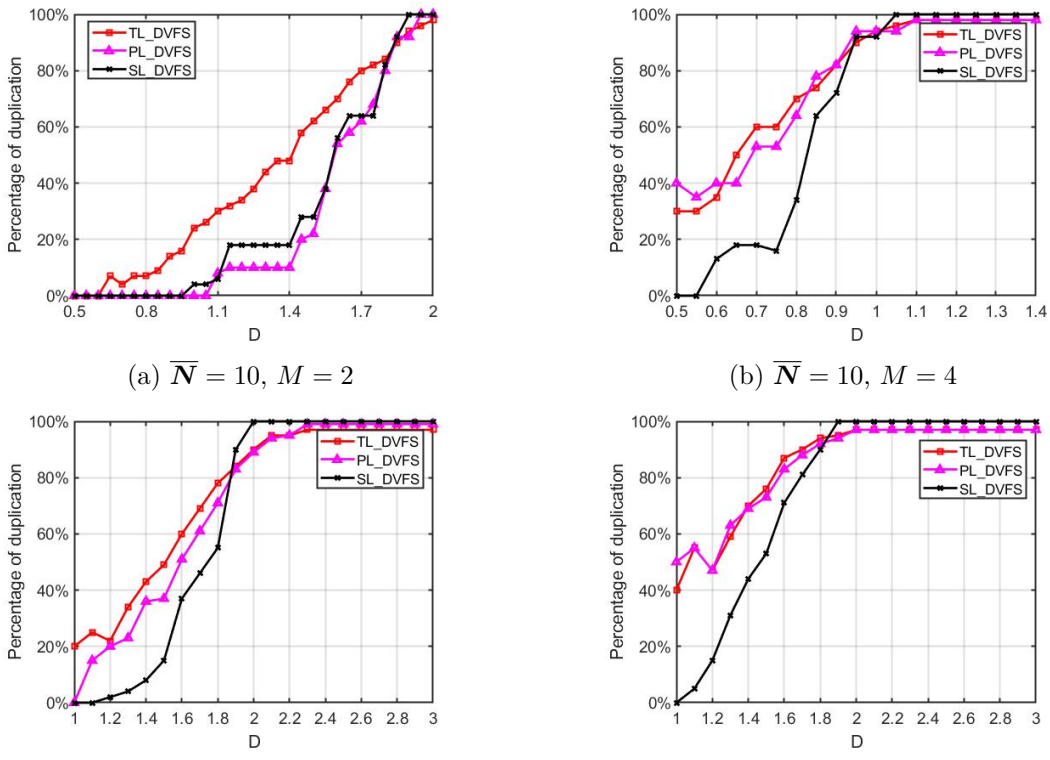
(e) SL-DVFS, $M = 4$



(f) SL-DVFS, $M = 6$

Figure 3.12: Reliability improvement for dependent tasks ($\overline{\boldsymbol{N}} = 20$) under all DVFS schemes.

(100%). For all DVFS schemes, as expected, when the deadline is more relaxed, O_RAFTM can duplicate more tasks using lower frequency, thus achieving less energy consumption, due to the available time slack. Furthermore, as all processors have the same frequency in SL-DVFS, this DVFS scheme has less flexibility in task duplication and scheduling, when deadlines are strict. In Fig. 3.13, there are two interesting observations: i) for some points, the duplication percentage decreases, when deadline is relaxed (Fig. 3.13c, when $D = 1.2$ for TL-DVFS, and Fig. 3.13d, when $D = 1.2$ for both TL-DVFS and PL-DVFS), and ii) the duplication percentage does not reach 100% for TL-DVFS and PL-DVFS, as SL-DVFS does. The reason is the flexibility to decide the task frequency for different DVFS schemes, and the reliability threshold's value. When the task's reliability threshold can be satisfied without duplication and with a low frequency, and energy consumption is less than the energy consumption when duplicating with the lowest frequency, then TL-DVFS and PL-DVFS schemes do not duplicate the task. However, with SL-DVFS, the same frequency must be assigned to all processors. Hence, a task with a high reliability constraint forces other tasks with low reliability constraints to execute with a higher frequency than the required one. As a result, duplication does not provide benefits, even if time



(a) $\overline{\boldsymbol{N}} = 10$, $M = 2$

(b) $\overline{\boldsymbol{N}} = 10$, $M = 4$

(c) $\overline{\boldsymbol{N}} = 20$, $M = 4$

(d) $\overline{\boldsymbol{N}} = 20$, $M = 6$

Figure 3.13: O_RAFTM duplication percentage ($\overline{\boldsymbol{N}} = 10$, $M = 2$ and $M = 4$, and $\overline{\boldsymbol{N}} = 20$, $M = 4$ and $M = 6$) for dependent tasks under all DVFS schemes.

slack exists. Due to these reasons, the duplication percentage is usually smaller for SL-DVFS, compared to TL-DVFS and PL-DVFS, except for relaxed enough deadlines.

Computation Time: Tables 3.12 to 3.14 provide the time required to find a solution, when $\overline{\boldsymbol{N}} = 10$ and $\overline{\boldsymbol{N}} = 20$ with $M = 4$, and $\overline{\boldsymbol{N}} = 20$ with $M = 6$. Comparing Table 3.13 and Table 3.12, with more tasks, more time is required to find a solution, as expected. For O_RAFTM, when the deadline is very strict or very relaxed, less time is needed to obtain the solutions. However, with intermediate deadlines, more time is required as O_RAFTM needs to explore the available time slack to decide which, and how many, tasks to be duplicated, without violating constraints, while consuming the least energy. O_TDM is the most running time expensive approach, because all tasks need to be duplicated, increasing the number of tasks to be scheduled, and thus, the time to find the solutions. Similar to independent tasks, O_RAM is the least running time expensive approach. However, as we have seen previously, this approach provides less energy savings compared to O_RAFTM.

Table 3.12: Computation time (sec) for dependent tasks ($\overline{\boldsymbol{N}} = 10$, $M = 4$) under all DVFS schemes.

	deadline(D)	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95	1	1.05	1.1	1.15	1.2
TL DVFS	O_RAFTM	0.45	0.57	0.95	2.17	3.31	4.31	3.77	3.11	5.57	1.47	1.92	3.71	0.51	0.3	0.27
	O_RAM	0.3	0.25	0.27	0.22	0.43	0.33	0.16	0.16	0.21	0.18	0.18	0.16	0.19	0.16	0.19
	O_TDM	2.52	0.56	0.61	0.97	1.53	2.65	2.45	2.08	1.26	1.08	0.72	0.72	0.55	0.36	0.37
PL DVFS	O_RAFTM	1.65	6.03	8.2	15.26	24.06	32.3	34.38	40.76	29.12	29.05	13.9	13.54	10.94	4.19	3.51
	O_RAM	0.82	1.78	1.51	1.7	1.94	1.72	1.79	1.79	2.15	2.22	1.87	1.88	2.1	1.77	1.84
	O_TDM	2.55	2.27	1.7	4.45	5.97	9.17	5.04	4.97	4.83	3.61	3.22	2.47	2.34	1.84	1.87
SL DVFS	O_RAFTM	0.28	0.19	0.26	0.83	1.47	1.86	1.18	1.13	0.84	0.92	0.67	0.59	0.53	0.27	0.23
	O_RAM	0.11	0.08	0.11	0.09	0.19	0.2	0.2	0.07	0.09	0.08	0.08	0.09	0.09	0.08	0.08
	O_TDM	0.48	0.14	1.02	1.64	1.51	2.5	1.83	1.59	1.43	1.7	0.88	0.68	0.71	0.54	0.44

Table 3.13: Computation time (sec) for dependent tasks ($\overline{\boldsymbol{N}} = 20$, $M = 4$) under all DVFS schemes.

	deadline(D)	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2.0	2.1	2.2
TL DVFS	O_RAFTM	14.9	61	74.9	138.2	202.1	256.9	327.3	219	1039.3	1054.7	126.8	58.7	123.1
	O_RAM	0.07	0.04	0.067	0.054	0.041	0.061	0.066	0.045	0.043	0.036	0.053	0.052	0.044
	O_TDM	-	-	-	9188.3	1550.4	1015.9	1165.4	2304.7	2814.6	1350.6	700.6	549.3	237.8
PL DVFS	O_RAFTM	228.6	662.2	529.9	1092	1815	4665	15722	4676	33294	70465	24239	24054	24072
	O_RAM	84.3	99.2	44.4	74.3	76.5	65.5	52.4	55	56.3	41.7	42.8	57.4	56.3
	O_TDM	-	-	-	2740	5677.5	5084.1	9641.1	14770.4	10571.4	8235.2	5517.1	3619.0	3169.6
SL DVFS	O_RAFTM	0.53	0.3	5.76	14.14	21.69	98.22	91.88	183.39	1301.28	585.03	65.29	28.56	15.08
	O_RAM	0.08	0.06	0.038	0.069	0.045	0.066	0.059	0.036	0.027	0.026	0.025	0.037	0.025
	O_TDM	-	-	-	238.51	230.59	304.66	370.62	1099.44	686.62	225.18	239.50	154.98	41.93

Table 3.14: Computation time (sec) for dependent tasks ($\overline{\boldsymbol{N}} = 20$, $M = 6$) under all DVFS schemes.

	deadline(D)	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2.0	2.1	2.2
TL DVFS	O_RAFTM	0.8	58.1	26.9	89.9	81.9	109.8	58.6	62.8	59.8	14.7	2.5	0.7	0.6
	O_RAM	0.07	0.06	0.08	0.09	0.10	0.06	0.05	0.05	0.07	0.05	0.06	0.08	0.05
	O_TDM	-	70.8	333.4	124.4	298.2	227.4	70.5	160.6	203.4	39.8	2.9	1.4	0.9
PL DVFS	O_RAFTM	114.2	697.3	730.8	661.5	859.5	824.0	1090.1	503.3	569.2	188.6	95.9	82.5	75.0
	O_RAM	566.0	397.9	533.6	671.5	546.1	820.5	954.5	562.0	687.4	693.2	799.0	1113.8	700.0
	O_TDM	-	2861.2	2874.1	5312.8	8795.7	1477.3	1237.6	1243.7	1794.6	232.5	41.7	42.2	35.2
SL DVFS	O_RAFTM	0.4	0.3	1.4	12.1	12.6	23.5	19.2	19.7	16.1	21.3	5.9	4.7	5.6
	O_RAM	0.09	0.07	0.06	0.12	0.08	0.06	0.04	0.04	0.04	0.04	0.04	0.05	0.04
	O_TDM	-	8.7	111.0	37.8	71.9	83.0	52.3	52.6	41.9	32.8	19.5	20.2	16.3

Experimental set-up: Reliability constraints not always met

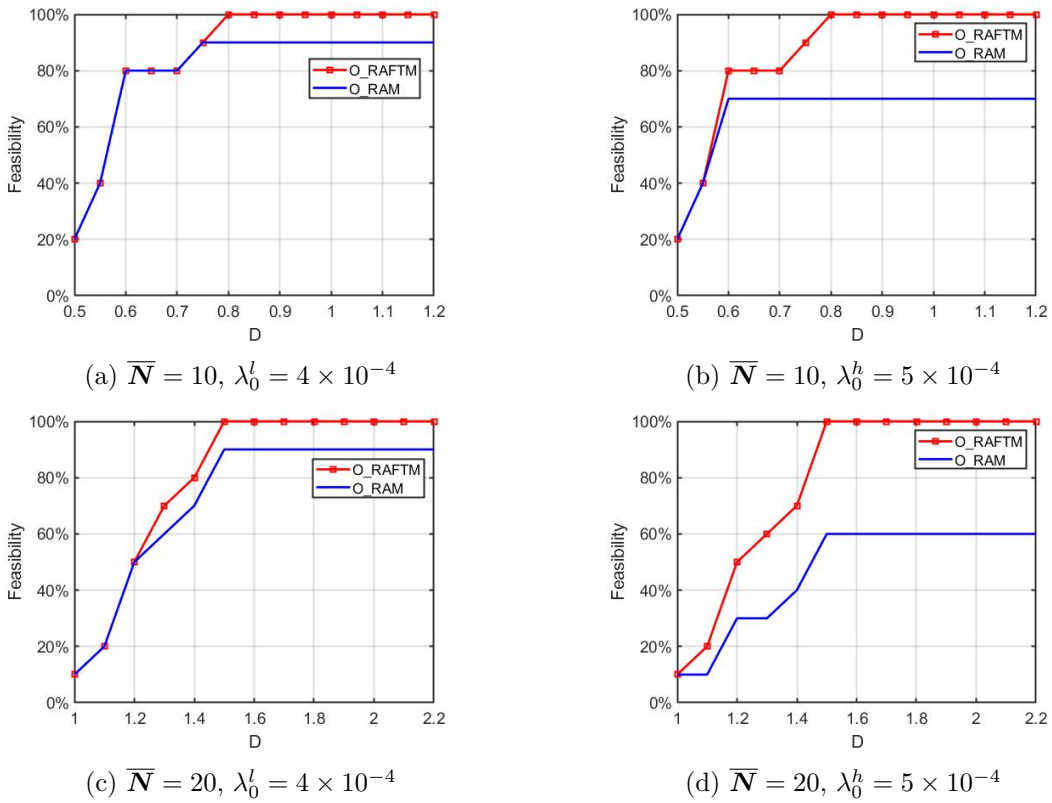
In this second experimental set-up, we explore the behavior of O_RAFTM and O_RAM at different failure rates λ_0 under TL-DVFS scheme [3]. First, we extend the range of the task reliability threshold, in order to have few tasks with higher reliability requirements. Then, we tune the parameter λ_0 in order to change the failure rate. This experimental set-up shows how the fault model influences the ability of obtaining feasible solutions, through a slight increase of λ_0. We will show that the proposed approach provides better results than O_RAM, with the fault rate changing. These modifications will affect only the proposed O_RAFTM and O_RAM approaches, since they use the reliability constraint to decide the task mapping. Two different values λ_0 are chosen for experiments: $\lambda_0^l = 4 \times 10^{-4}$, and $\lambda_0^h = 5 \times 10^{-4}$.

Experimental results: Reliability constraints not always met

Feasibility: Fig. 3.14 depicts the ability of O_RAM, compared to O_RAFTM, to obtain feasible solution, for $\lambda_0^l = 4 \times 10^{-4}$ (Fig. 3.14a and Fig. 3.14c) and $\lambda_0^h = 5 \times 10^{-4}$ (Fig. 3.14b and Fig. 3.14d), with $\overline{\boldsymbol{N}} = 10$ and $\overline{\boldsymbol{N}} = 20$, $M = 4$. Generally:

— On the one hand, as the λ_0 increases, the reliability of a task, due to the processor fault rate, is decreased. Therefore, with λ_0 increasing, the capability of O_RAM to always find a frequency, that meets the reliability threshold of all tasks, decreases, especially for tasks with high reliability requirements. On the other hand, even with λ_0 increasing, O_RAFTM is able to still meet these high reliability requirements, by duplicating tasks and assigning a high frequency. This can be observed by comparing Fig. 3.14a to Fig. 3.14b, where the feasibility of O_RAM is reduced by 20%, and Fig. 3.14c to Fig. 3.14d, where the feasibility of O_RAM is reduced by 30%. However, O_RAFTM feasibility remains the same, always finding a solution at relaxed deadlines.

3. we remind λ_0 is the average fault rate at maximum frequency, see section 1.5.2



(a) $\overline{\boldsymbol{N}} = 10$, $\lambda_0^l = 4 \times 10^{-4}$

(b) $\overline{\boldsymbol{N}} = 10$, $\lambda_0^h = 5 \times 10^{-4}$

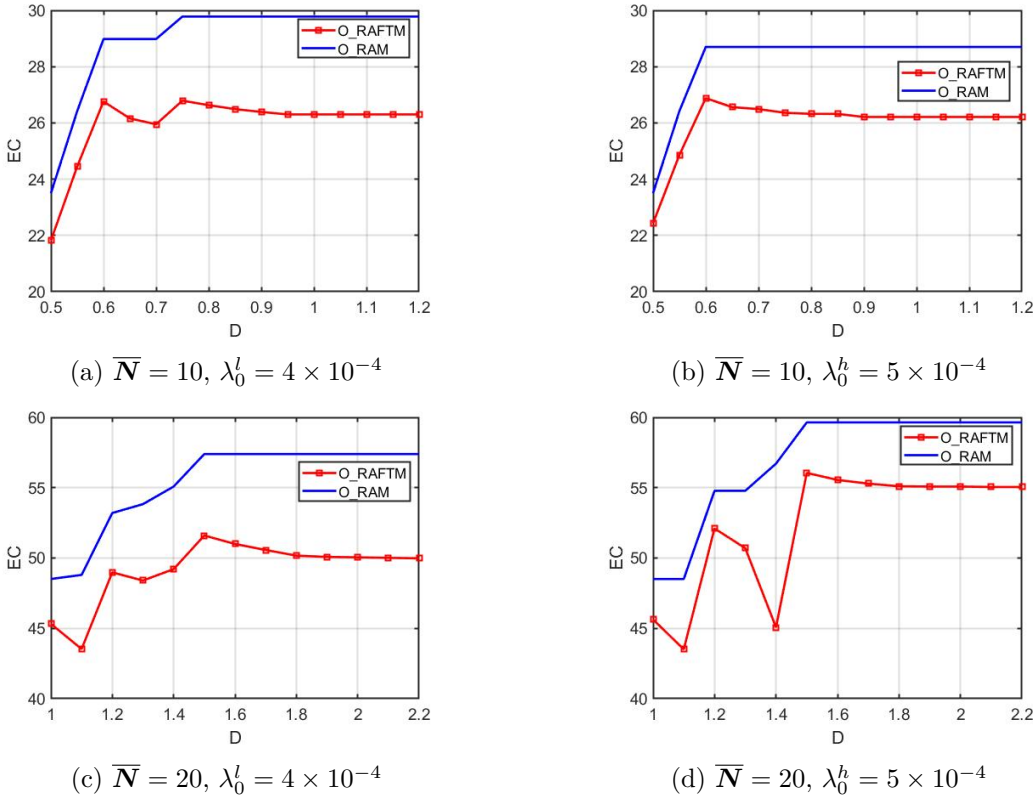
(c) $\overline{\boldsymbol{N}} = 20$, $\lambda_0^l = 4 \times 10^{-4}$

(d) $\overline{\boldsymbol{N}} = 20$, $\lambda_0^h = 5 \times 10^{-4}$

Figure 3.14: Feasibility for dependent tasks ($M = 4$) with $\lambda_0^l = 4 \times 10^{-4}$ and $\lambda_0^h = 5 \times 10^{-4}$ for TL-DVFS scheme.

— As the number of tasks increases, the feasibility is affected. As we observe by comparing Fig. 3.14a to Fig. 3.14c and Fig. 3.14b to Fig. 3.14d, the curves of O_RAM and O_RAFTM separate at stricter deadlines, when $\overline{\boldsymbol{N}} = 20$, compared to the ones when $\overline{\boldsymbol{N}} = 10$. Comparing the feasibility between the first and the second set-ups, we observe that the feasibility of first set-up can be 100% for both and O_RAM, with a smaller average fault rate λ_0 at relaxed deadlines. However, with the average fault rate increasing, it is not possible for O_RAM to find a solution satisfying the reliability requirements, even with maximum frequency and relaxed deadline. This observation worsens with λ_0 increasing. Thus, with a different fault model, executing only the original task may not be able to provide reliable execution. Replication is needed in order to guarantee the reliability threshold of the tasks. It is the reason why O_RAFTM find solutions much more easily than O_RAM when deadline is not too strict.

Energy Consumption: Fig. 3.15 depicts the energy consumption of O_RAM, compared to the proposed approach, when RAM found a solution.



(a) $\overline{\boldsymbol{N}} = 10$, $\lambda_0^l = 4 \times 10^{-4}$

(b) $\overline{\boldsymbol{N}} = 10$, $\lambda_0^h = 5 \times 10^{-4}$

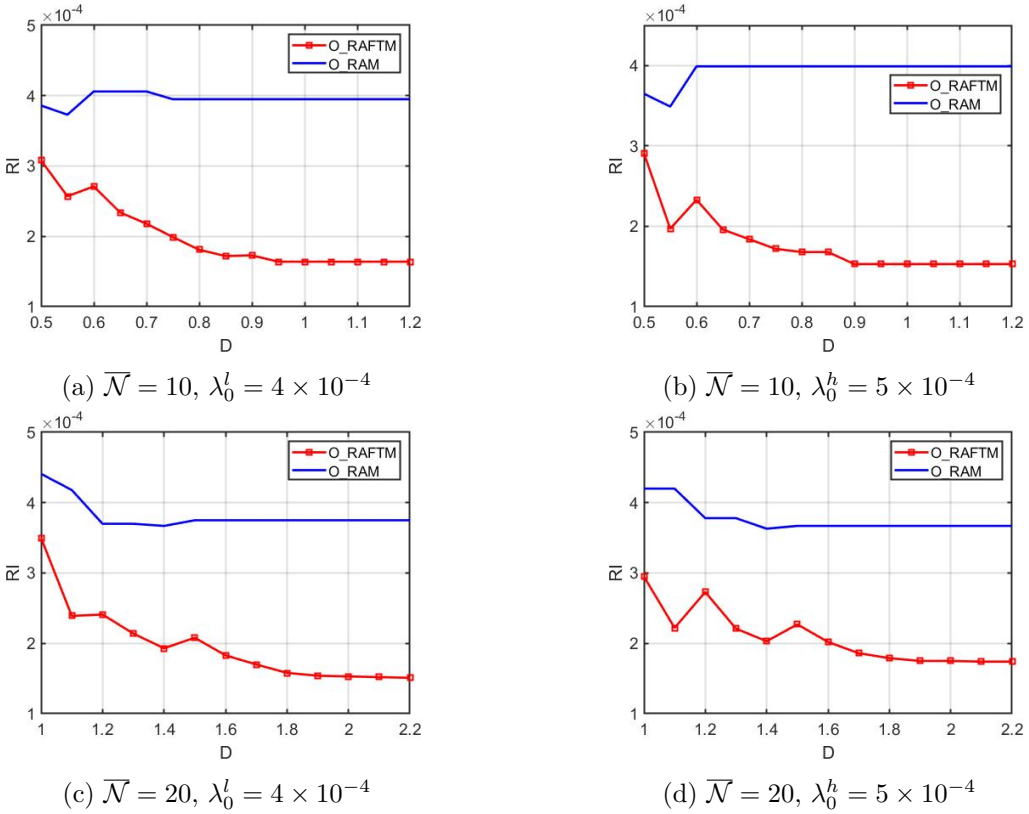
(c) $\overline{\boldsymbol{N}} = 20$, $\lambda_0^l = 4 \times 10^{-4}$

(d) $\overline{\boldsymbol{N}} = 20$, $\lambda_0^h = 5 \times 10^{-4}$

Figure 3.15: Energy consumtpion for dependent tasks ($M = 4$) with $\lambda_0^l = 4 \times 10^{-4}$ and $\lambda_0^h = 5 \times 10^{-4}$ for TL-DVFS scheme.

— As the λ_0 increases, the processor sensibility to faults is increased. Then, the energy gains are slightly reduced, from 11.61% to 8.51% ($\overline{\boldsymbol{N}} = 10$, Fig. 3.15a to Fig. 3.15b) and from 12.41% to 9.23% ($\overline{\boldsymbol{N}} = 20$, Fig. 3.15c to Fig. 3.15d), on average This is due to the fact that, based on $\lambda(f)$ (see Section 3.2.1), the reliability with same frequency decreases, when λ_0 increases. In order to guarantee a reliable execution, a high frequency must be assigned to meet the reliability requirement, leading to higher energy consumption for the proposed approach under same condition except when λ_0 increasing from $\lambda_0^l = 4 \times 10^{-4}$ to $\lambda_0^h = 5 \times 10^{-4}$, which makes the energy gains between O_RAFTM amd O_RAM reduced.

— Comparing the first and second set-ups, we have similar observation by comparing Fig. 3.9b with Fig. 3.15a and Fig. 3.15b, and Fig. 3.10a with Fig. 3.15c and Fig. 3.15d, the energy savings are decreased between O_RAFTM and O_RAM. For instance, from 36.6% (first set-up) to 11.61% and 8.51%, on average when $\overline{\boldsymbol{N}} = 10$ and $M = 4$ with λ_0 increasing from $\lambda_0 = 5 \times 10^{-5}$, to $\lambda_0^l = 4 \times 10^{-4}$ and to $\lambda_0^h = 5 \times 10^{-4}$.



(a) $\overline{\mathcal{N}} = 10$, $\lambda_0^l = 4 \times 10^{-4}$

(b) $\overline{\mathcal{N}} = 10$, $\lambda_0^h = 5 \times 10^{-4}$

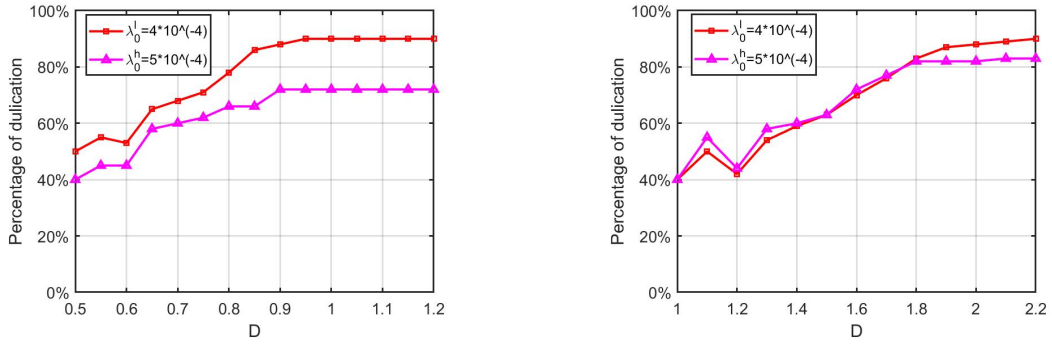
(c) $\overline{\mathcal{N}} = 20$, $\lambda_0^l = 4 \times 10^{-4}$

(d) $\overline{\mathcal{N}} = 20$, $\lambda_0^h = 5 \times 10^{-4}$

Figure 3.16: Reliability improvement for dependent tasks ($M = 4$) with $\lambda_0^l = 4 \times 10^{-4}$ and $\lambda_0^h = 5 \times 10^{-4}$ for TL-DVFS scheme.

Reliability Improvement: Fig. 3.16 depicts the reliability improvement of O_RAFTM and O_RAM, for the cases where O_RAM was able to find a solution. Generally, when more energy is consumed, we can observe the trend that a higher reliability achievement is obtained. With λ_0 increasing, the reliability decreases under same conditions. A higher frequency is needed to meet the reliability requirement for O_RAM, which increases the energy consumption and the reliability achievement. On the basis of satisfying the reliability requirements, O_RAM achieves a slightly better reliability improvement than our approach, while it sacrifices the ability of obtaining feasible solutions.

Task Duplication: The percentage of duplicated task is depicted in Fig. 3.17. As Based on the fault model in Section 1.5.2, the reliability decreases with higher λ_0^h. When the timing constraints are always satisfied, a higher λ_0^h with the same frequency, leads to lower reliability. To meet reliability constraints, either a very high frequency is assigned to original tasks, or a relative high frequency is required for both original and duplicated tasks. The first option consumes less energy in this case. As observed in Fig. 3.17a and Fig. 3.17b O_RAFTM duplicates more tasks



(a) $\overline{\boldsymbol{N}} = 10$ (b) $\overline{\boldsymbol{N}} = 20$

Figure 3.17: O_RAFTM task duplication percentage for dependent tasks (a) $\overline{\boldsymbol{N}} = 10$, and b) $\overline{\boldsymbol{N}} = 20$, $M = 4$) with $\lambda_0^l = 4 \times 10^{-4}$ and $\lambda_0^h = 5 \times 10^{-4}$ for TL-DVFS scheme.

when $\lambda_0^l = 4 \times 10^{-4}$, specifically with relaxed deadlines, which sounds reasonable.

Computation Time: Table 3.15 and 3.16 give the time needed to obtain a solution for our proposed O_RAFTM and O_RAM after the extension of fault rate λ_0. Similar to the time analysis in set-up 1, O_RAM needs less running time than our proposed approach when finding a feasible solution, at the price of more energy consumption and less ability to obtain feasible solutions. Of course computation time increases with number of processors. And similarly to experiment set-up 1, strict and very relaxed deadlines takes less time than other deadlines.

Table 3.15: Computation time (sec) ($\overline{\boldsymbol{N}} = 10$, $M = 4$) with $\lambda_0^l = 4 \times 10^{-4}$ and $\lambda_0^h = 5 \times 10^{-4}$ for TL-DVFS scheme.

	deadline(D)	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95	1	1.05	1.1	1.15	1.2
λ_0^l	O_RAFTM	0.77	0.93	1.01	1.49	1.48	2.00	1.28	1.31	1.21	0.74	0.68	0.66	0.62	0.64	0.62
	O_RAM	0.07	0.04	0.06	0.07	0.13	0.19	0.12	0.05	0.06	0.05	0.06	0.06	0.05	0.04	0.04
λ_0^h	O_RAFTM	0.67	0.61	0.87	1.18	0.87	1.39	0.80	0.72	0.66	0.55	0.53	0.58	0.55	0.57	0.50
	O_RAM	0.10	0.05	0.05	0.06	0.13	0.20	0.13	0.05	0.05	0.08	0.05	0.05	0.05	0.04	0.07

Table 3.16: Computation time (sec) ($\overline{\boldsymbol{N}} = 20$, $M = 4$) with $\lambda_0^l = 4 \times 10^{-4}$ and $\lambda_0^h = 5 \times 10^{-4}$ for TL-DVFS scheme.

	deadline(D)	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2.0	2.1	2.2
λ_0^l	O_RAFTM	26.85	194.76	98.45	208.45	224.04	315.23	167.25	326.79	278.74	99.46	216.81	31.26	4.79
	O_RAM	0.04	0.03	0.02	0.05	0.03	0.04	0.04	0.04	0.03	0.02	0.02	0.03	0.02
λ_0^h	O_RAFTM	32.08	206.57	583.86	156.99	155.10	117.55	155.63	174.00	19.15	24.26	33.65	3.52	3.06
	O_RAM	0.05	0.03	0.03	0.05	0.04	0.05	0.05	0.04	0.04	0.02	0.03	0.04	0.02

3.4 Conclusion

In this chapter, we discussed task mapping of both independent tasks and dependent tasks on multiprocessor systems. Our contributions in this chapter include formulating the studied problem by jointly deciding the frequency assignment to tasks, task allocation to processors and the tasks to be duplicated in order to minimize total energy consumption, under real-time and reliability requirement constraints. As a fault tolerance technique, we show the importance of applying partial duplication where decisions need to be made to select the part of the set of tasks to be duplicated. This approach provides a good trade-off between energy saving and reliability improvement. This reliability-aware Fault-tolerant Task Mapping (RAFTM) approach can be used to solve the studied problems under task level (TL), processor level (PL) and system level (SL) DVFS schemes.

First, we presented the problems we want to address for both independent tasks and dependent tasks. The problems are originally formulated as non-linear problems. We proved that a reliable replacement method can be used to safely and equivalently transfer the non-linear problems into Mixed-Integer-Linear-Programming (MILP) problems, which can be solved with optimal solutions.

Then, we provided simulation-based evaluations for our proposed approach and two other SoA approaches which focus on solving the same problems for comparison. A large number of experiments are done. As expected, experimental results show that TL-DVFS achieves less energy consumption, thanks to flexibility in task frequency assignment, compared to SL-DVFS. Results show that the proposed approach is generally able to provide better energy savings, and at the same time, higher feasibility even when existing approaches may fail to find a solution, without violating timing and reliability constraints, under the three possible DVFS schemes.

Chapter 4

Energy-Efficient Fault Tolerant Task Mapping with Heuristic Solutions

As shown in the previous chapter, since the problem of task mapping on multicore platforms is NP-hard, it is time-consuming to obtain optimal solutions, even when a few number of tasks and processors is considered. Therefore, it is not practically realistic to search for optimal solutions especially when the problem size becomes large. In this chapter, we propose a set of heuristic algorithms to solve the problems presented in Chapter 3 in order to reduce the computational complexity and obtain near-optimal solutions. The heuristic algorithms for task mapping we propose under the three DVFS schemes for independent tasks are presented in Sections 4.1, 4.2 and 4.3. We first describe in details the heuristic idea for TL-DVFS in Section 4.1. Then we highlight the parts that are different for PL-DVFS and SL-DVFS. The heuristic algorithms for task mapping when tasks have dependencies are then explained in Sections 4.4, 4.5 and 4.6. Experiment results based on simulations are provided and comparison with SoA approaches is done.

4.1 Independent Tasks under Task Level DVFS

4.1.1 Reliability-aware Fault-tolerant Task Mapping heuristic

We propose a *Reliability-aware Fault-tolerant Task Mapping heuristic (H_RAFTM)* to solve the task mapping problem for independent tasks when supporting TL-DVFS, presented in Section 3.2. Table 4.1 summarizes the main notations. The proposed heuristic is based on the following definitions and constraints:

Definition 1 (Configuration)**.** *A task may be executed in several different configurations in the procedure of task mapping. A configuration j of a task τ_i is a 7-tuple set, denoted as $C_i^j = \{f_i^o, f_i^d, et_i^o, et_i^d, E_i^o, E_i^d, R_i\}$, where f_i^o (f_i^d) is the assigned frequency, et_i^o (et_i^d) is the required execution time, E_i^o (E_i^d) is the energy consumption of the original (duplicated) task and*

Notations	Definitions
τ_i^o/τ_i^d	the original/duplicated copy of task τ_i
(v_l, f_l)	the l^{th} voltage/frequency level
W_i	WCET of task τ_i
D	the global deadline
R_i^{th}	reliability threshold of task τ_i
$TotalET_m$	total workload of processor θ_m
et_i	execution time of task τ_i
$rank_i$	the rank of task τ_i
C_i^j	the j^{th} configuration of task τ_i
SC_i	scheduled configuration of task τ_i in current task mapping
$TM_i^{C_i^j}$	the mapping of task τ_i in configuration C_i^j
AM	the mapping of the application

Table 4.1: Main notations for independent tasks under TL-DVFS scheme.

R_i is the reliability of the task τ_i (original and potentially duplicated task). If the task is not duplicated, we have $f_i^d = et_i^d = E_i^d = 0$.

Definition 2 (Task Mapping)**.** *A mapping of a task τ_i, under the task configuration C_i^j, is a 6-tuple set, denoted as $TM_i^{C_i^j} = \{C_i^j, \theta_i^o, \theta_i^d\}$, where θ_i^o (θ_i^d) is the allocated processor for original (duplicated) task.*

Definition 3 (Application Mapping)**.** *The mapping of the application (AM) is given by the set of mappings of N original tasks and $S \subseteq N$ duplicated tasks. The mapping is valid if the real-time constraints are satisfied.*

Definition 4 (Total Execution Time)**.** *The total execution time of a processor θ_m is $TotalET_m = \sum_{\theta_i=\theta_m, i\in\boldsymbol{N}, k\in\{o,d\}} et_i^k$.*

Constraint 1 (Reliability Constraint)**.** *A task must be executed meeting its reliability requirement, i.e., $R_i \geq R_i^{th}$.*

Constraint 2 (Deadline constraint)**.** *The application must finish before the deadline D. For each processor, all tasks (including original and potentially duplicated tasks) allocated on it must be executed within D:*

$$\sum_{i\in\boldsymbol{N}, \theta_i^o=\theta_m} et_i^o + \sum_{i\in\boldsymbol{N}, \theta_i^d=\theta_m} et_i^d \leq D, \forall m \in \boldsymbol{M}. \tag{4.1}$$

The total execution time of each processor should not exceed the global deadline D .

The proposed heuristic is described by Algorithm 1 and it has two phases:

1. `Phase A` obtains, per task, the set of possible configurations that meet the reliability constraint, ordered in decreasing energy consumption.

Algorithm 1 Proposed H_RAFTM algorithm for independent tasks under TL_DVFS scheme.

Input: Task graph ($\boldsymbol{G}$) and set of processors ($\boldsymbol{M}$).
Output: Application mapping (AM).

`// Phase A`
1: **for** each task τ_i in N **do**
2: $RTE_i = \{C_i^j$: C_i^j is the $j-th$ configuration of $\tau_i\}$;
3: $FC_i = RTE_i$ - $\{C_i^j$: $R_i < R_i^{th}\}$;
4: $BC_i = \{FC_i$: $f_i^d = 0\}$;
5: **for** each bc in BC_i of task τ_i **do**
6: $PC_i = FC_i$ - $\{FC_i$: $f_i^d \neq 0 \wedge \sum\{et_i^o, et_i^d\} \geq et_i^{bc} \wedge \sum\{E_i^o, E_i^d\} > E_i^{bc}\}$;
7: **end for**
8: $rPC_i = \{PC_i$: $PC_i[j]$ decreasing energy consumption$\}$;
9: **end for**
`// Phase B`
10: **for** each task τ_i in $\boldsymbol{N}$ **do**
11: Compute $rank_i$;
12: **end for**
13: PL-T = $\{N$: ordered in decreasing $rank_{\tau_i}\}$;
14: **for** each task τ_i in PL-T **do**
15: $SC_i = rPC_i[0]$;
16: Obtain $TM_i^{SC_i}$ (θ_m with $\min_{m \in \boldsymbol{M}} TotalET_m$);
17: **end for**
18: $AM_0 = \{TM_i^{SC_i}, i \in \boldsymbol{N}\}$;
19: Compute $\{TotalET_m^{AM_0}, m \in \boldsymbol{M}\}$ of AM_0;
20: **if** $\exists \text{TotalET}_m^{AM_0} > D$ **then**
21: Infeasible problem, algorithm stops.
22: **else if** $\forall \text{TotalET}_m^{AM_0} = D$ **then**
23: $AM = AM_0$, algorithm stops.
24: **else if** $\exists \text{TotalET}_m^{AM_0} < D$ **then**
25: AM relaxation (Algorithm 2);
26: **end if**

2. `Phase B` obtains the application mapping, by allocating tasks to processors using the least total energy consumption, under real-time constraints.

Phase A: Task configurations under reliability constraint.

`Phase A` (L. 1-9) is applied per task. For each task (L. 1), a Reliability, execution Time, Energy consumption (RTE) table is created based on all possible configurations (L. 2). A pruning step removes the task configurations that do not satisfy the reliability constraint (L. 3). The result is the Feasible Configurations (FC) space of the task. FC_i considering only the original task τ_i^o (when $f_i^d = 0$, no duplicated task exists) serve as Baseline Configurations (BC) (L. 4). The next step prunes any feasible configuration with duplicated tasks, if both energy consumption and execution time are larger than any BC_i (L. 5-7). The result is the Possible Configurations (PC) space. The PCs are ranked based on decreasing energy consumption (rPC) (L. 8).

Phase B: Application mapping under real-time constraint

`Phase B` uses `Phase A` task configurations and performs the application mapping, subject to the real-time constraint. `Phase B` consists of three steps (L. 10-26):

Step 1 (L. 10-13): Priorities are given to tasks for task allocation based on the largest-average-execution-time-first rule. We define the rank value of each task as $rank_i = \overline{et_i}$ (L. 11). In the rest of the manuscript, for any set X, $|X|$ denotes the size of set X. The average execution time of a task is computed by the average execution time among all possible configurations, i.e.,

$$\overline{et_i} = \frac{\sum_{j \in PC_i, k \in \{o,d\}} et_i^k[j]}{|PC_i|} \tag{4.2}$$

where $et_i^k[j]$ defines the execution time of task i under configuration j. The Priority List of tasks (PL-T) is ordered in decreasing rank value (L. 13).

Step 2 (L. 14-23): The initial application mapping AM_0 is generated to check if the problem is feasible and time slack is available. For each task, AM_0 uses the first configuration in rPC_i as the Selected Configuration SC_i (L. 15). For each task, choosing the processor with least $TotalET$, we obtain the task mapping ($TM_i^{SC_i}$) (L. 16). The set of all task mappings provides the AM_0 (L. 18) and the total execution time of all processors $TotalET_0$ is obtained (L. 19). If the total execution time of some processors exceeds the global deadline D, the studied problem is infeasible (L. 20-21), and the algorithm stops. If the total execution time of all processors is equal to the deadline, the initial application mapping is the final mapping and the algorithm stops (L. 22-23).

Step 3: (L. 24-26) Otherwise, if there is no processor that its workload exceeds D, and for some processors, time slack exists after the initial task mapping is obtained (L. 24), the mapping can be relaxed leading to energy savings. Overall, different task configurations and different tasks can be relaxed. Algorithm 2 decides which task and with which configuration to be selected for relaxation (L. 25). As a first step, the current mapping (AM, $TotalET$) is initialised with the initial mapping (AM_0, $TotalET^{AM_0}$) (L. 1). The algorithm is applied iteratively, until there is no available time slack for relaxation or all tasks have reached their configuration with the least energy consumption (L. 2). Before the relaxation, we compute the energy saving (ES) and execution time increase (TI) for each task and each remaining configuration, compared to the first configuration used in current task and application mapping (L. 3-7). We combine two criteria to select the task, with a potential new configuration (NC), to do the relaxation in order to save energy. First, we consider a global search among all tasks with all their possible configurations to select a new configuration (NC) for a task that achieves the highest value ES/TI (L. 8) and this task τ_{rel} with new selected configuration $SC_{\tau_{rel}}$ is selected to do the relaxation. Also a local search explores rPC_i sequentially for each task, by selecting always the first configuration, for relaxation (L. 9). Note that, when the conditions in L. 2 cannot be satisfied for the first time,

Algorithm 2 Mapping Relaxation Algorithm for independent tasks under TL_DVFS scheme.

1: $AM = AM_0$, $TotalET = TotalET^{AM_0}$;
2: **while** ∃TotalET < D **and** rPC$_i$> 1($\forall\tau_i$) **do**
3: **for** each task τ_i in $\boldsymbol{N}$ **do**
4: **for** each configuration $j (j \neq 0)$ in rPC_i **do**
5: Compute $ES_i[j]$ and $TI_i[j]$ compared to configuration $j = 0$;
6: **end for**
7: **end for**
`// global search relaxation:`
8: $\tau_{rel} = \tau_i$ with $NC_{\tau_{rel}} = rPC_i[j] : rPC_i[j] = \max_{i \in \boldsymbol{N}} \{\sum_{c \in rPC_i} (ES_i[c]/TI_i[c])\}$;
`// local search relaxation:`
9: $\tau_{rel} = \tau_i$ with $NC_{\tau_{rel}} = rPC_i[0], i \in \boldsymbol{N}$;
10: **for** each task τ_i in PL-T **do**
11: Obtain $TM_i^{SC_i}$ (θ_m with $\min_{m \in \boldsymbol{M}} TotalET$);
12: **end for**
13: $AM = \{TM_i^{SC_i}, i \in \boldsymbol{N}\}$;
14: Compute $TotalET_m^{AM}$ of current AM for each processor θ_m;
15: **for** each configuration in $rPC_{\tau_{rel}}$ **do**
16: $rPC_{\tau_{rel}} = rPC_{\tau_{rel}}$ - $\{rPC_{\tau_{rel}}$: $E^j_{\tau_{rel}} \geq E^{NC_{\tau_{rel}}}_{\tau_{rel}}\}$;
17: **end for**
18: **end while**

the global search is stopped and the local search is applied. By combining the global and local search, we can fully explore the available time slack for relaxation, especially for exploring in a fine-grained way the time slack left after the global search. After selecting a task with a new configuration, all task mappings are updated accordingly (L. 10-12). Furthermore, application mapping AM (L. 13) and the total execution time $TotalET$ for each processor are obtained (L. 14). Last, for the relaxed task with the new selected configuration ($NC_{\tau_{rel}}$), all configurations that have a higher energy consumption than the selected one are removed from $rPC_{\tau_{rel}}$ (L. 15-17).

4.1.2 Evaluation results

This section evaluates the proposed heuristic (H_RAFTM) with i) the optimal approach using Gurobi 9.0.2 (O_RAFTM) presented in Section 3.2.4, and ii) two SoA heuristics. similar to Chapter 3, the SoA approaches are the heuristic versions of a) the Reliability-Aware Mapping (H_RAM), that meets the reliability constraint without task duplication, similar to [62] and ESRG algorithm in [24], and b) the Task Duplication Mapping (H_TDM), applying task duplication for all tasks, similar to [32, 24], when the number of replicas is two, or to [37], with 100% task duplication.

The model of the processor, the parameters of tasks and the reliability requirements are same as in Table 3.3 in Section 3.2.4. A large and diverse set of experiments is performed, by tuning the:

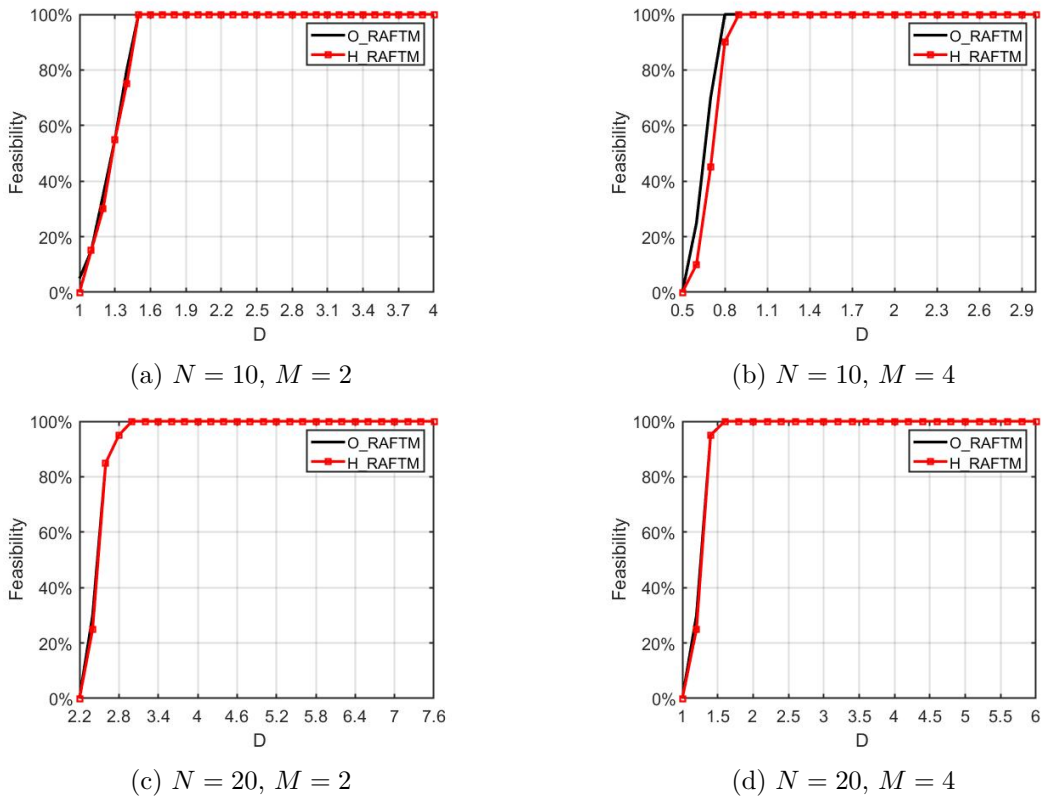
1. Number of processors ($\boldsymbol{M} = 2, 4$).
2. Size of task set ($\boldsymbol{N} = 10, 20$).
3. For each application task graph, a number of experiments (denoted as NE) is performed, each time with different task characteristics (W_i and R_i^{th}).

To evaluate the approaches, the Feasibility, Energy Consumption (EC) in mJ, Reliability Improvement (RI) and Computation time (CT) in seconds (sec.) are presented.

Comparison with the optimal approach

This section compares the behavior of proposed heuristic and optimal solutions for independent tasks under TL-DVFS, considering small scale problems. We present the results for $NE = 10$, considering random graphs with $N = 10$ and $N = 20$ original tasks and $M = 2$ and $M = 4$ processors.

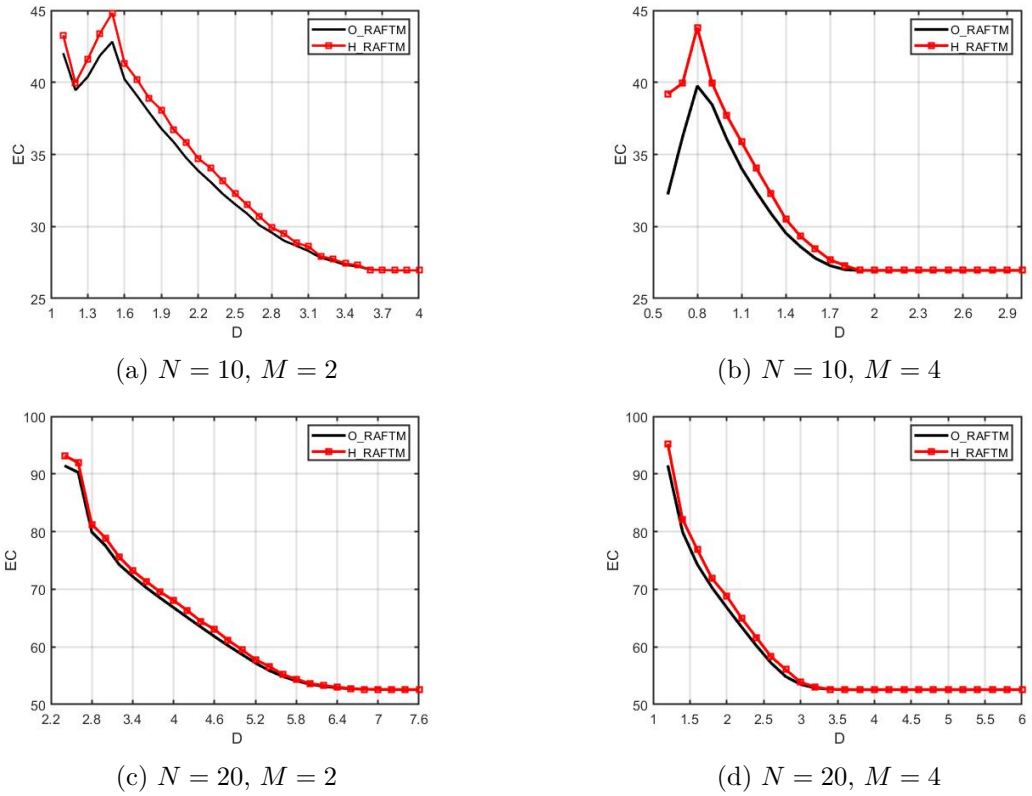
Regarding feasibility, as shown in Fig. 4.1, when the deadline is relaxed, the H_RAFTM achieves same feasibility as the optimal approach O_RAFTM. Overall, when the deadline is



(a) $N = 10$, $M = 2$ (b) $N = 10$, $M = 4$

(c) $N = 20$, $M = 2$ (d) $N = 20$, $M = 4$

Figure 4.1: Feasibility of optimal and heuristic approaches for independent tasks under TL-DVFS scheme.



(a) $N = 10$, $M = 2$

(b) $N = 10$, $M = 4$

(c) $N = 20$, $M = 2$

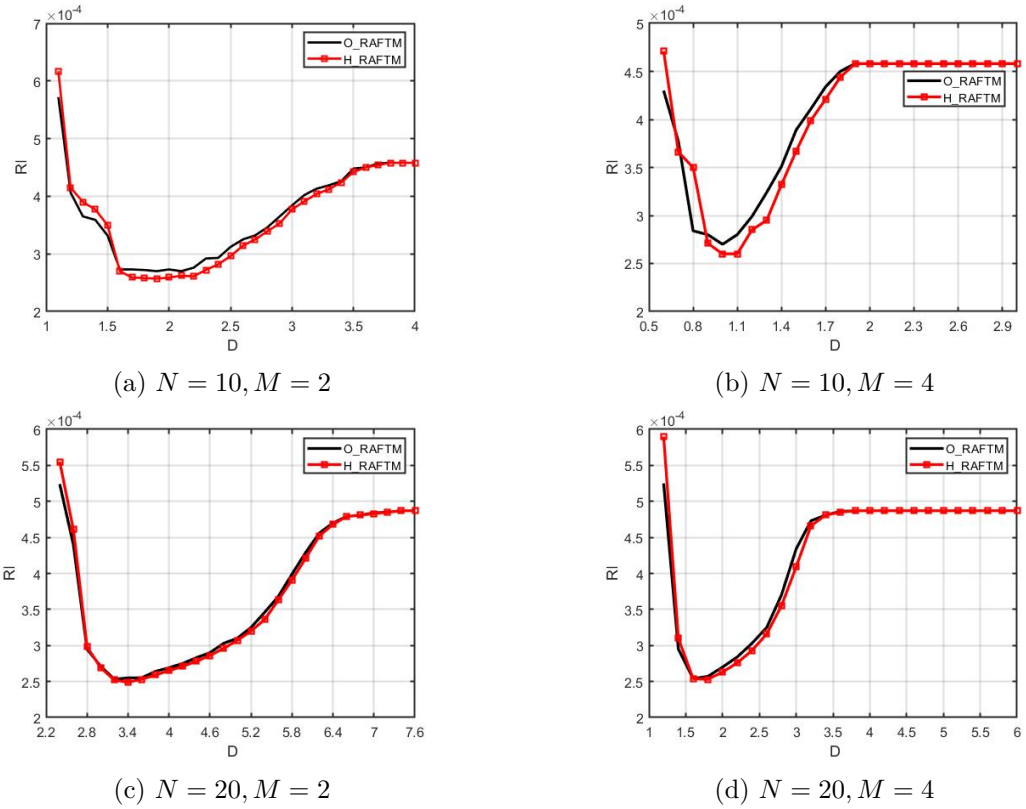
(d) $N = 20$, $M = 4$

Figure 4.2: Energy consumption (mJ) of optimal and heuristic approaches for independent tasks under TL-DVFS scheme.

strict, the H_RAFTM feasibility is very close to the optimal feasibility. The average difference between H_RAFTM and O_RAFTM, before the deadline for which both are able to achieve 100% feasibility, is 5% (Fig. 4.1a) when $N = 10$ and $M = 2$. A slightly higher difference (16.7%) is observed when $N = 10$ and $M = 4$ (Fig. 4.1b). When $N = 20$ the average difference is 1.67% for $M = 2$ (Fig. 4.1c) and 2.5% for $M = 4$ (Fig. 4.1d).

Regarding energy consumption, as shown in Fig. 4.2, overall, H_RAFTM consumes slightly more energy than O_RAFTM. H_RAFTM consumes on average 2.14% when $N = 10$ and $M = 2$ (Fig. 4.2a) and 5.85% $M = 4$ (Fig. 4.2b) more energy than the optimal solutions, when the number of tasks is increased to 20, H_RAFTM consumes on average 1.12% for $M = 2$ (Fig. 4.2c) and 1.96% for $M = 4$ (Fig. 4.2d) more than the optimal solution. When the deadline is relaxed, H_RAFTM and O_RAFTM obtain solutions with the same energy consumption since the proposed heuristic is able to fully explore available time slack for task mapping.

Regarding reliability improvement, H_RAFTM provides overall comparable reliability improvement with the optimal solutions at the price of consuming slightly more energy for a given deadline, as depicted in Fig. 4.3.



(a) $N = 10, M = 2$ (b) $N = 10, M = 4$

(c) $N = 20, M = 2$ (d) $N = 20, M = 4$

Figure 4.3: Reliability improvement of optimal and heuristic approaches for independent tasks under TL-DVFS scheme.

The average computation time of O_RAFTM and H_RAFTM is computed taking into account the experiments when a feasible solution is found. Table 4.2 shows the results in seconds per deadline D. It can be observed that, although few tasks and processors are used, the time to obtain the optimal solution is very long, especially when there are more tasks ($N = 20$) and processors $M = 4$. The difference in computation time between optimal approach and the proposed H_RAFTM, on average, is $\times 10^4$ when deadlines are not very relaxed. This is due to the fact that the computational complexity to obtain optimal solutions for NP-hard problems increases dramatically with problem size increasing.

Overall, the obtained results show that i) H_RAFTM provides near-optimal solutions, and ii) as expected, H_RAFTM takes significantly less time to obtain the results compared to the optimal approaches.

Comparison with heuristic approaches

The feasibility of the proposed and the two SoA heuristics is depicted in Fig. 4.4. Compared to H_TDM, the proposed H_RAFTM can find solutions in significantly more experiments than

Table 4.2: Computation time (sec) of optimal and heuristic approaches for independent tasks under TL-DVFS scheme.

$N = 10$, $M = 2$													
D	**1.1**	**1.2**	**1.3**	**1.4**	**1.5**	**1.6**	**1.7**	**1.8**	**1.9**	**2.0**	**2.1**	**2.2**	**2.3**
O_RAFTM	3.81	1.32	1.77	0.87	1.96	0.53	1.82	2.56	1.85	2.38	2.81	2.29	2.95
H_RAFTM	~ 0.01												
D	**2.4**	**2.5**	**2.6**	**2.7**	**2.8**	**2.9**	**3.0**	**3.1**	**3.2**	**3.3**	**3.4**	**3.5**	**3.6-4.0**
O_RAFTM	2.93	3.37	3.96	1.80	1.58	1.19	1.05	0.86	0.49	0.32	0.17	1.03	$\sim$0.05
H_RAFTM	~ 0.01												

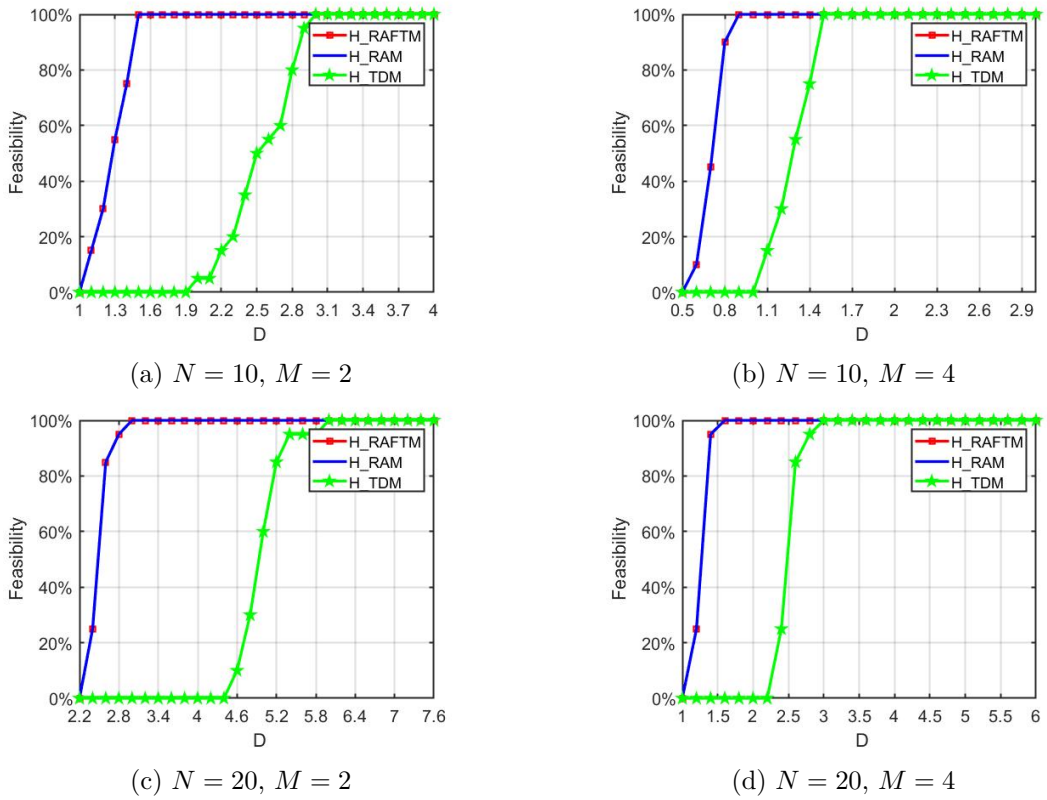
$N = 10$, $M = 4$							
D	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**	**1.1**	**1.2**
O_RAFTM	10.34	11.44	9.47	37.59	37.44	60.24	830.27
H_RAFTM	~ 0.01						
D	**1.3**	**1.4**	**1.5**	**1.6**	**1.7**	**1.8**	**1.9-3.0**
O_RAFTM	217.16	215.56	273.33	9.52	1.16	5.62	$\sim$0.03
H_RAFTM	~ 0.01						

$N = 20$, $M = 2$											
D	**2.4**	**2.6**	**2.8**	**3.0**	**3.2**	**3.4**	**3.6**	**3.8**	**4.0**	**4.2**	**4.4**
O_RAFTM	3.85	2.95	2.47	1.42	2.25	2.36	1.59	2.55	2.39	2.43	3.31
H_RAFTM	0.01	~ 0.02		~ 0.03							
D	**4.6**	**4.8**	**5.0**	**5.2**	**5.4**	**5.6**	**5.8**	**6.0**	**6.2**	**6.4**	**6.6-7.6**
O_RAFTM	4.62	4.87	4.28	4.51	3.85	2.97	2.67	1.51	2.13	0.82	$\sim$0.06
H_RAFTM	~ 0.03			~ 0.04							

$N = 20$, $M = 4$						
D	**1.2**	**1.4**	**1.6**	**1.8**	**2.0**	**2.2**
O_RAFTM	3058.33	1909.92	521.33	673.87	399.08	1237.58
H_RAFTM	0.02	~ 0.03				
D	**2.4**	**2.6**	**2.8**	**3.0**	**3.2**	**3.4-6.0**
O_RAFTM	4356.88	165.41	38.65	25.84	3.80	$\sim$0.11
H_RAFTM	~ 0.04					

H_TDM, especially when the deadline is not fully relaxed or the number of cores is reduced. When tasks meet their reliability constraint, H_RAFTM does not need to duplicate these tasks. However, H_TDM duplicates all tasks, and thus, it is able to find solutions only when the deadline is relatively relaxed or several processors exist to run the tasks in parallel. Before obtaining 100% feasibility for both approaches, on average, H_RAFTM finds a solution in more experiments than H_TDM, i.e., 66.0% with $M = 2$ and 63.3% with $M = 4$, for $N = 10$, and 68.6% with $M = 2$ and 68.3% with $M = 4$ for $N = 20$. Note that, H_RAFTM and H_RAM have the same feasibility. This behavior is explained as follows: when H_RAM finds a solution, it means that the reliability constraint of all tasks can be met by executing only the original task with a high frequency. After obtaining all possible configurations in Phase A, for each task, the configurations (PC) of H_RAFTM always include all the configurations (PC) of H_RAM. In this case, H_RAFTM can also find the H_RAM solution.

The energy consumption obtained by the solutions of the three heuristics is depicted in Fig. 4.5. Comparing H_RAFTM and H_RAM (first row), we observe that they consume similar



(a) $N = 10$, $M = 2$

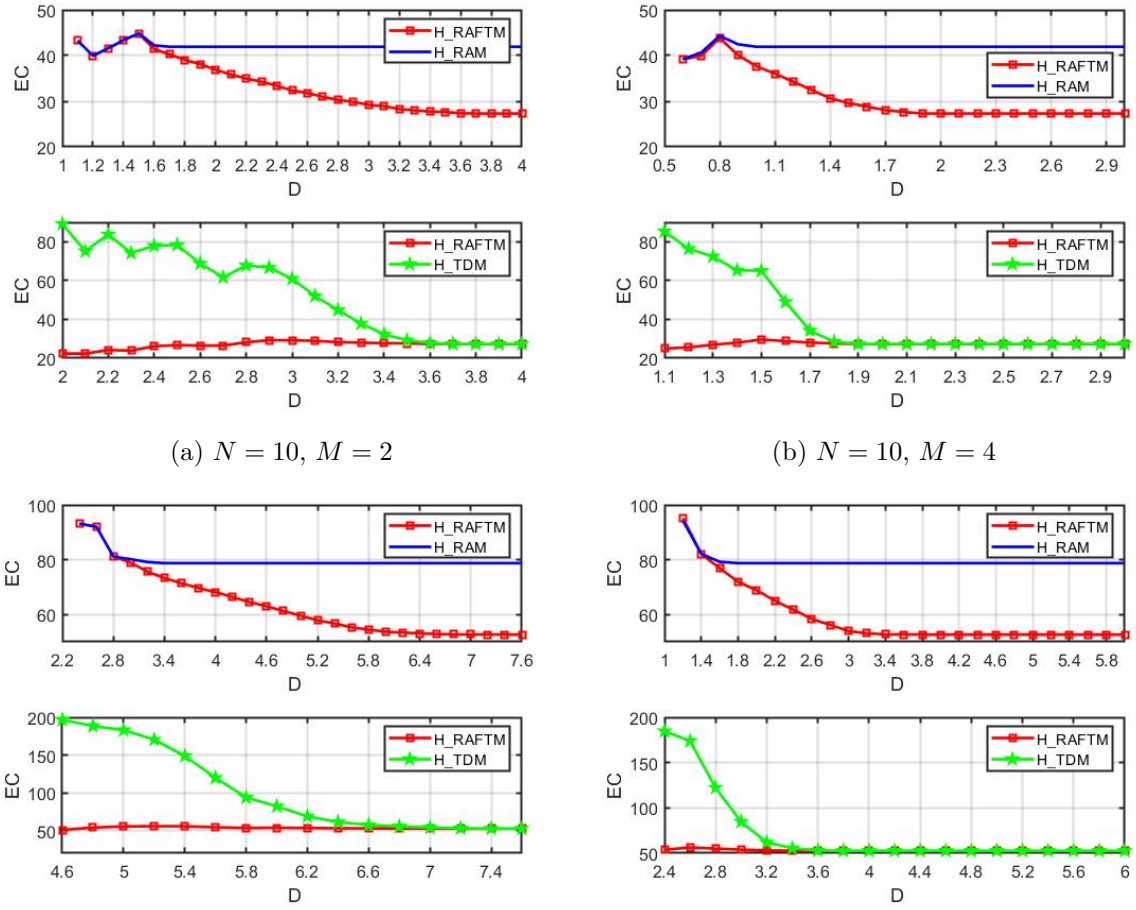
(b) $N = 10$, $M = 4$

(c) $N = 20$, $M = 2$

(d) $N = 20$, $M = 4$

Figure 4.4: Feasibility of heuristics for independent tasks under TL-DVFS scheme.

energy at very strict deadlines, when the number of processors is small. In this case, H_RAFTM behaves similarly to H_RAM, i.e., mainly executing the original tasks with the frequency required to achieve the reliability constraint. With deadline relaxing, H_RAFTM starts to consume less energy than H_RAM. H_RAFTM achieves this gain by exploring the available time slack to duplicate tasks in order to save energy, e.g., up to 53.5% for $N = 10$ and 49.5% for $N = 20$ at relaxed deadlines. Similarly, when more processors are available, H_RAFTM can take advantage of these resources and execute duplicated task in parallel. Comparing H_RAFTM and H_TDM (second row), as H_TDM applies task duplication for every task, it cannot find solutions in very strict deadlines. When H_TDM starts finding solutions at a relatively relaxed deadline, H_RAFTM is able to use the available time slack to do partial duplication. Therefore, considering the experiments where both H_RAFTM and H_TDM can find solutions, H_RAFTM consumes significantly less energy than H_TDM. For the decision to do partial duplication, H_RAFTM selects the task configuration, if exists, with only the original task, meeting the reliability constraint and consuming less energy than configurations with duplicated tasks. Since H_TDM duplicates all tasks, its energy consumption can be significant, when it finds a solution. In very relaxed deadlines, H_RAFTM and H_TDM behave similar,



(a) $N = 10$, $M = 2$

(b) $N = 10$, $M = 4$

(c) $N = 20$, $M = 2$

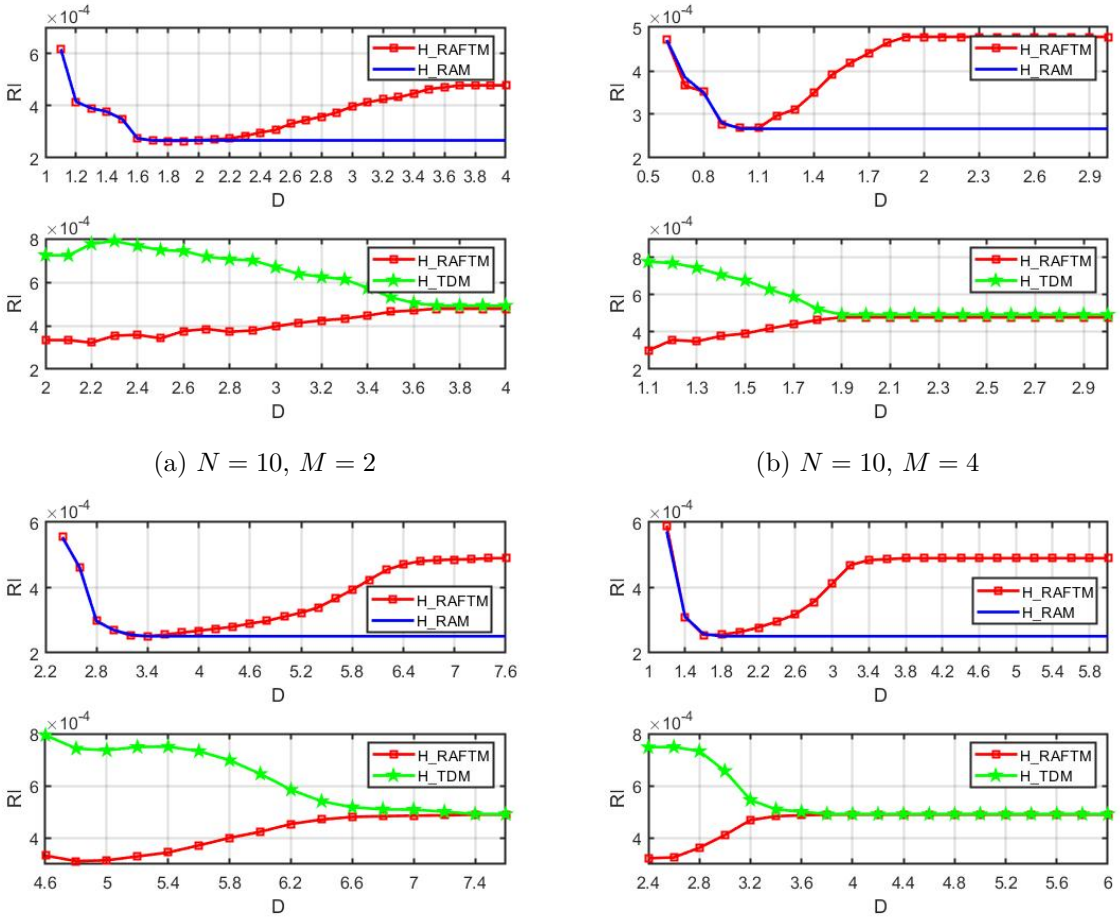
(d) $N = 20$, $M = 4$

Figure 4.5: Energy consumption (mJ) of heuristics for independent tasks under TL-DVFS scheme.

i.e., duplicate the tasks with configurations when less energy is consumed.

The reliability improvement is given in Fig. 4.6. H_RAFTM achieves higher reliability than H_RAM (first row), except in very strict deadlines when H_RAFTM behaves similar to H_RAM without task duplication. As explained above, when there is not available time slack to perform duplication, H_RAFTM behaves similar to H_RAM as most of the tasks are executed with only their original copy. Compared to H_TDM (second row), H_RAFTM provides lower reliability for tight deadlines, as it duplicates only a part of the task-set. The same reliability improvement can be achieved in relaxed deadlines, since both H_RAFTM and H_TDM duplicate tasks similarly.

The computation time in seconds of H_RAFTM, H_RAM and H_TDM heuristics is depicted in Fig. 4.7. Overall, when the deadline increases, the trends are as follows: the H_RAM



(a) $N = 10$, $M = 2$

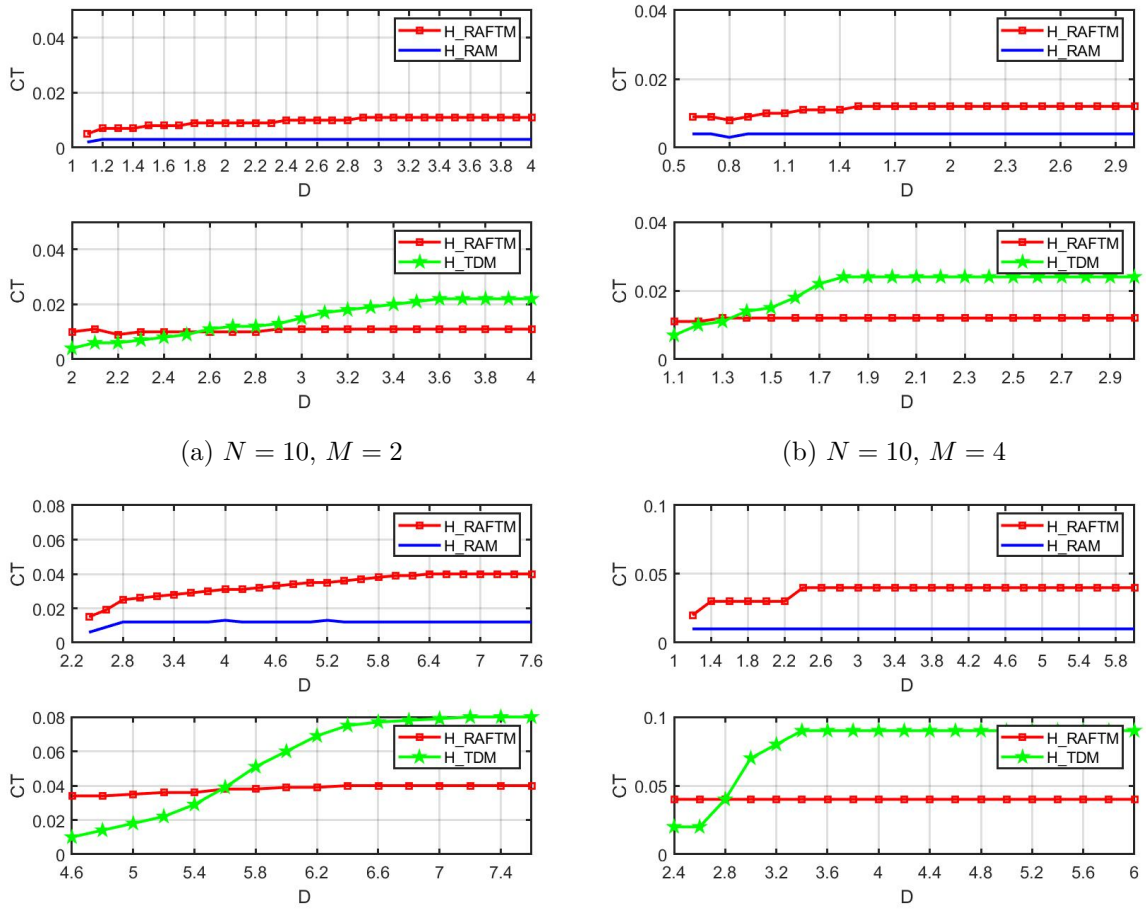
(b) $N = 10$, $M = 4$

(c) $N = 20$, $M = 2$

(d) $N = 20$, $M = 4$

Figure 4.6: Reliability improvement of heuristics for independent tasks under TL-DVFS scheme.

computation time remains stable and the H_RAFTM and H_TDM computation times slightly increase. The computation time to obtain a feasible solution increases with deadline relaxing, due to the fact that the proposed heuristic explores the PC space for each task, based on the deadline constraints. Therefore, the more relaxed the deadline is, the larger is the PC space to be explored per task, and thus, more time is needed. Note that, the H_TDM is generally the most expensive approach in terms of computation time, when deadline is very relaxed. This behavior is due to the fact that all tasks are duplicated, which increases the total number of tasks (original task and duplicated task) to be scheduled, and thus, the number of PCs in each task PC space, and the time required to find a solution. For H_RAM, since it only executes original tasks, it has a reduced number of PCs in the PC space, taking the least time to obtain a solution. However, it provides less energy savings as explained above, especially at relaxed deadlines.



(a) $N = 10$, $M = 2$

(b) $N = 10$, $M = 4$

(c) $N = 20$, $M = 2$

(d) $N = 20$, $M = 4$

Figure 4.7: Computation time (sec) of heuristics for independent tasks under TL-DVFS scheme.

4.2 Independent Tasks under Processor Level DVFS

4.2.1 Reliability-aware Fault-tolerant Task Mapping heuristic

The proposed *Reliability-aware Fault-tolerant Task Mapping heuristic (H_RAFTM)* is leveraged in order to be able to handle the PL-DVFS scheme. The aforementioned definitions and constraints for TL-DVFS scheme are valid and any difference will be explicitly described. Algorithm 3 describes the proposed approach, which is explained in the next paragraphs.

Phase A: Task configurations under reliability constraint.

The Phase A for H_RAFTM adapted for PL-DVFS is similar to the H_RAFTM for TL-DVFS. However, there is a difference when configurations are pruned. Under PL-DVFS, a higher number of possible configurations are kept for each task in phase A (L. 6 in Algorithm 3) com-

pared to TL-DVFS (L. 6 in Algorithm 1). Configurations where one frequency, used to execute original or duplicated task, is equal to the frequency of baseline configurations (BC), are pruned (L. 6 in Alg. 3), similar to TL-DVFS scheme (L. 6 in Alg. 1). However, configurations where one of the frequencies, used to execute the original and duplicated tasks, is equal to the frequency of baseline configurations, but the minimal execution time of original and duplicated tasks is larger than the execution time of the baseline configurations and the total energy consumption is higher than baseline configurations, i.e., $\sum\{et_i^o, et_i^d\} \geq et_i^{bc} \wedge \sum\{E_i^o, E_i^d\} > E_i^{bc}$ (L. 6 in Algorithm 1), are pruned under TL-DVFS while kept under PL-DVFS scheme. In PL-DVFS, all tasks executed on same processor have the same frequency. Therefore, we need to keep these configurations, since the final frequency depends on all tasks allocated on the same core. To illustrate that with an example, let's assume that the BC of a task is the following: the original task is executed with f_5 with an energy consumption equal to 4.108mJ. A configuration that uses f_2/f_5 for original and duplicated tasks with higher energy consumption than 4.108 mJ is pruned according to L. 6 in Alg. 3. However, this configuration is still kept under PL-DVFS, because PL-DVFS has less flexibility in frequency assignment compared to TL-DVFS. The frequency assignment depends on the task allocation that is performed at later later steps, and over-pruning at this step will lead to infeasible solutions. To avoid this, we keep the configurations with $\sum\{et_i^o, et_i^d\} \geq et_i^{bc} \wedge \sum\{E_i^o, E_i^d\} > E_i^{bc}$ under PL-DVFS.

Phase B: Application mapping under real-time constraint

`Phase B` uses `Phase A` task configurations and performs the application mapping, subject to deadline constraint introduced in Equation (4.1). `Phase B` consists of three steps (L. 10-26):

Step 1 (L. 10-14): This step is the same as described by Equation 4.2, where priorities are given to tasks for task allocation based on the largest-average-execution-time $rank_i = \overline{et_i}$ (L. 11). The Priority List of tasks (PL-T) is ordered in decreasing rank value (L. 13). We list all possible combinations of Frequency-To-Processor (FTP) assignment for all processors (L. 14) and order them in a decreasing order based on the the sum of frequency indexes, named rFTP, which describes the ranked FTP space. For instance, assuming $M = 3$, the first group is $FTP = \{f_{L-1}, f_{L-1}, f_{L-1}\}$, where all processors are assigned with highest frequency f_{L-1}, whereas the last group is $FTP = \{f_0, f_0, f_0\}$ when all processors are assigned with lowest frequency. Since the processors are homogeneous, we consider identical the combinations where the sum of frequency indexes is the same. For example, $FTP = \{f_0, f_1, f_2\}$ and $FTP = \{f_2, f_0, f_1\}$ correspond to the same FTP group with a sum of frequency indexes equal to 3.

Step 2 (L. 15-25): The initial application mapping AM_0 is used to check whether the problem is feasible and time slack is available. To obtain initial task mapping AM_0, we start with $FTP = \{f_{L-1}, \ldots, f_{L-1}\}$ where all cores are assigned with the highest frequency f_{L-1} (L. 15). For each FTP group, we list all available configurations (ACs) for each task from the PC space (L. 17).

Algorithm 3 Proposed H_RAFTM algorithm for independent tasks under PL_DVFS scheme.

Input: Task graph (G) and set of processors (M).
Output: Application mapping (AM).

// Phase A
1: **for** each task τ_i in N **do**
2: $RTE_i = \{C_i^j$: C_i^j is the j-th configuration of $\tau_i\}$;
3: $FC_i = RTE_i$ - $\{C_i^j$: $R_i < R_i^{th}\}$;
4: $BC_i = \{FC_i$: $f_i^d = 0\}$;
5: **for** each bc in BC_i of task τ_i **do**
6: $PC_i = FC_i$ - $\{FC_i$: $f_i^d \neq 0$ $\}$;
7: **end for**
8: $rPC_i = \{PC_i$: $PC_i[j]$ increasing energy consumption$\}$;
9: **end for**
// Phase B
10: **for** each task τ_i in $\boldsymbol{N}$ **do**
11: Compute $rank_i$;
12: **end for**
13: PL-T = $\{N$: ordered in decreasing $rank_{\tau_i}\}$;
14: Obtain all possible frequency-to-processor groups (FTP) and order in decreasing sum of frequency index;
15: Start with all processor in highest frequency f_{L-1}, i.e., $FTP = \{f_{L-1}, ..., f_{L-1}\}$
16: **for** each task τ_i in PL-T **do**
17: List all available configurations (AC);
18: Compute $TM_i^{SC_i}$ (θ_m with $\min_{m \in \boldsymbol{M}} TotalET_m$);
19: **end for**
20: $AM_0 = \{TM_i^{SC_i}, i \in \boldsymbol{N}\}$;
21: Compute $TotalET_m^{AM_0}$ of AM_0 for each processor θ_m;
22: **if** $\exists \text{TotalET}_m^{AM_0} > D$ **then**
23: Infeasible problem, algorithm stops.
24: **else if** $\forall \text{TotalET}_m^{AM_0} = D$ **then**
25: $AM = AM_0$, algorithm stops.
26: **else if** $\exists \text{TotalET}_m^{AM_0} < D$ **then**
27: AM relaxation (Algorithm 4);
28: **end if**

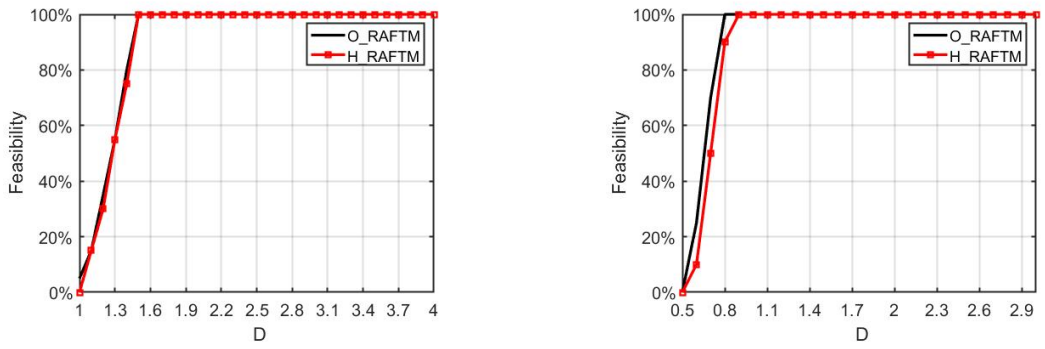
For example, the frequency assignments for a task among all its ACs with FTP=$\{f_0, f_1, f_2\}$ can be f_0/f_1, f_0/f_2 and f_1/f_2 for the execution of original and duplicated tasks. The two available cores with the least $TotalET$ are chosen to execute the original and duplicated tasks for τ_i (L. 18). If a task does not need to be duplicated, the execution time of the duplicated task is set zero. The set of all task mappings provides the AM_0 (L. 20) and the total execution time $TotalET$ for each processor is obtained (L. 21). If there exists any processor that its $TotalET$ exceeds the deadline, the problem is infeasible (L. 22-23), and the algorithm stops. If the $TotalET$ for all processors is equal to the deadline, the initial application mapping is the final mapping and the algorithm stops (L. 24-25).

Step 3: (L. 26-28) Otherwise, if time slack exists for some processors based on the initial task mapping (L. 26), the frequency assignments can be relaxed leading to energy savings.

Algorithm 4 Mapping Relaxation algorithm for independent tasks under PL_DVFS scheme.

1: $AM = AM_0$, $TotalET = TotalET^{AM_0}$;
2: **while** $TotalET < D$ $(\exists\theta)$ and $|rFTP| > 1$ **do**
3: **for** each FTP in $rFTP$ **do**
4: **for** every task τ_i in PL-T **do**
5: List all available configurations (AC);
6: Compute $TM_i^{SC_i}$ $(\theta_m with \min_{m \in \boldsymbol{M}} TotalET)$
7: **end for**
8: $AM = \{TM_i^{SC_i}, i \in \boldsymbol{N}\}$;
9: Compute $TotalET_m^{AM}$ of AM for each processor θ_m;
10: **end for**
11: remove FTP from $rFTP$
12: **end while**

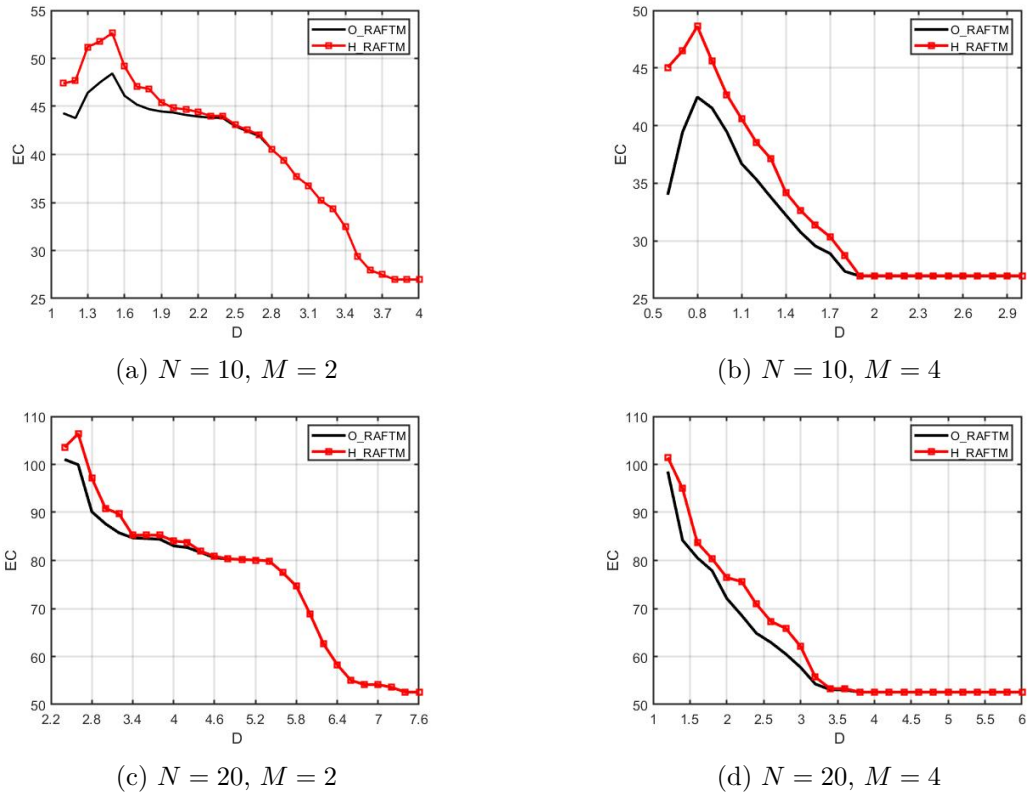
Alg. 4 selects the FTP for the new application mapping. Initially, the current mapping (and its *TotalET* for each processor) is initialised with the initial mapping (L. 1). The algorithm is applied iteratively, as long as there exists time slack for a processor and the last FTP group is not reached (L. 2). In each iteration, the first FTP group in $rFTP$ is selected (L. 3). Then, we



(a) $N = 10$, $M = 2$ (b) $N = 10$, $M = 4$

(c) $N = 20$, $M = 2$ (d) $N = 20$, $M = 4$

Figure 4.8: Feasibility of optimal and heuristic approaches for independent tasks under PL-DVFS scheme.



(a) $N = 10$, $M = 2$ (b) $N = 10$, $M = 4$

(c) $N = 20$, $M = 2$ (d) $N = 20$, $M = 4$

Figure 4.9: Energy consumption (mJ) of optimal and heuristic approaches for independent tasks under PL-DVFS scheme.

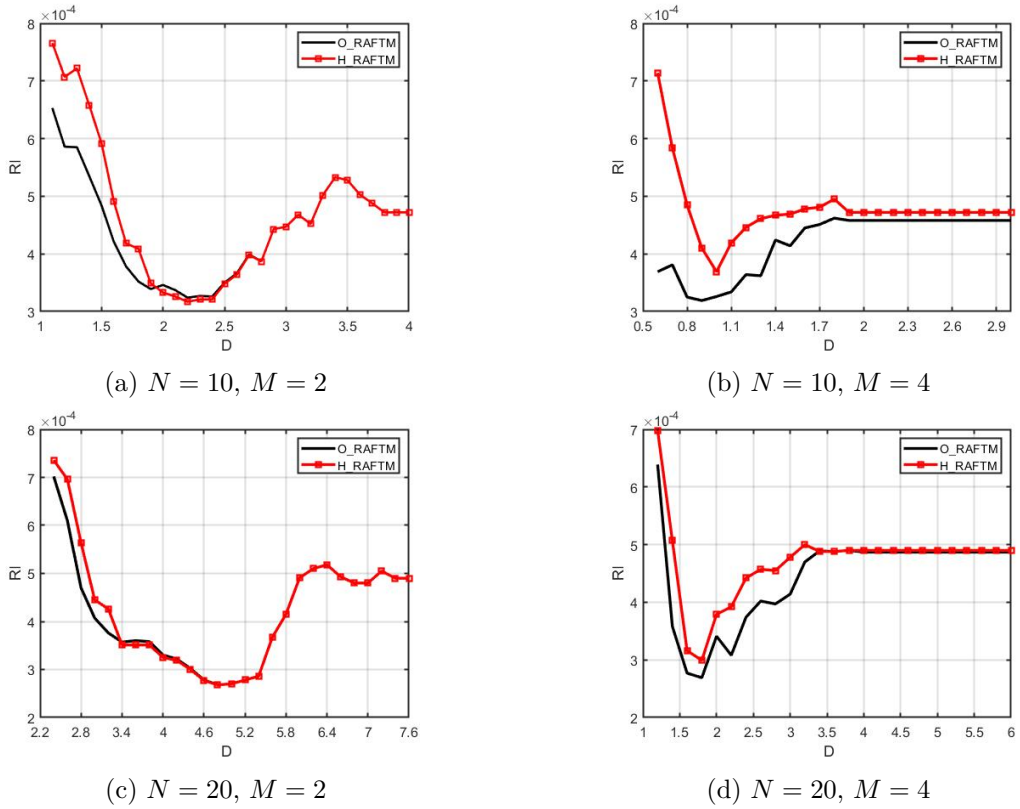
list all available configurations (ACs) for each task according to the selected FTP group (L. 5) and the task mapping is computed (L. 6). The new application mapping AM (L. 8) and the total execution time $TotalET$ for each processor (L. 9) are obtained. If there still exists time slack, the current FTP is removed from $rFTP$ (L. 11) and a new iteration with the next FTP group is explored.

4.2.2 Evaluation results

Comparison with the optimal approach

Regarding feasibility, as shown in Fig. 4.8, the obtained results show that the feasibility of PL-DVFS is the same as the feasibility of TL-DVFS in Section 4.1.2. This behavior is explained based on the fact that in each experiment, the first deadline, where a feasible solution can be obtained, is the same among different DVFS schemes. The initial task mapping starts with every task in TL-DVFS and every processor in PL-DVFS assigned with highest frequency, which meets the reliability constraints.

Regarding energy consumption, as shown in Fig. 4.9, in general, H_RAFTM consumes



(a) $N = 10$, $M = 2$ (b) $N = 10$, $M = 4$

(c) $N = 20$, $M = 2$ (d) $N = 20$, $M = 4$

Figure 4.10: Reliability improvement of optimal and heuristic approaches for independent tasks under PL-DVFS scheme.

slightly more energy than O_RAFTM. Before the deadline, for which both obtained solutions consume the same energy, H_RAFTM consumes, on average, 3.9% ($N = 10$ and $M = 2$), 10.8% ($N = 10$ and $M = 4$), 2.5% ($N = 20$ and $M = 2$) and 5.8% ($N = 20$ and $M = 4$) more energy than the optimal solutions. When deadline is relaxed, H_RAFTM and O_RAFTM obtain solutions with the same energy consumption, similar to the results obtained for TL-DVFS.

Regarding reliability improvement, as shown in Fig. 4.10, H_RAFTM provides higher reliability improvements than optimal solutions at the price of consuming slightly more energy at strict deadlines.

The average computation time of O_RAFTM and H_RAFTM is computed over the number of experiments when a feasible solution is found for both approaches. Table 4.3 shows the results in seconds per deadline D. Similar to TL-DVFS, it can be observed that although few tasks and processors are used, the time to obtain the optimal solution is very long, especially when a higher number of tasks ($N = 20$) and processors ($M = 4$) is used and when deadlines are not very relaxed. The difference on computation time between optimal approach and the proposed H_RAFTM on average is around a factor of $\times 5 \times 10^4$ when $N = 20$ and $M = 4$.

Table 4.3: Computation time (seconds) of optimal and heuristic approaches for independent tasks under PL-DVFS scheme.

$N = 10,\ M = 2$														
D	**1.1**	**1.2**	**1.3**	**1.4**	**1.5**	**1.6**	**1.7**	**1.8**	**1.9**	**2.0**	**2.1**	**2.2**	**2.3**	**2.4**
O_RAFTM	1.67	1.23	1.27	1.29	1.13	1.01	0.73	0.80	0.92	1.20	1.23	1.74	1.94	2.25
H_RAFTM	~ 0.02													
D	**2.5**	**2.6**	**2.7**	**2.8**	**2.9**	**3.0**	**3.1**	**3.2**	**3.3**	**3.4**	**3.5**	**3.6**	**3.7**	**3.8-4.0**
O_RAFTM	3.01	3.88	3.15	3.75	4.68	4.78	4.29	4.41	7.80	5.89	3.51	2.92	1.70	0.13
H_RAFTM	~ 0.02													

$N = 10,\ M = 4$							
D	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**	**1.1**	**1.2**
O_RAFTM	104.78	53.31	75.08	110.45	112.50	183.62	269.00
H_RAFTM	0.15	0.16	~ 0.17				
D	**1.3**	**1.4**	**1.5**	**1.6**	**1.7**	**1.8**	**1.9-3.0**
O_RAFTM	566.26	424.56	438.05	191.38	35026.12	31.67	∼0.58
H_RAFTM	~ 0.17						

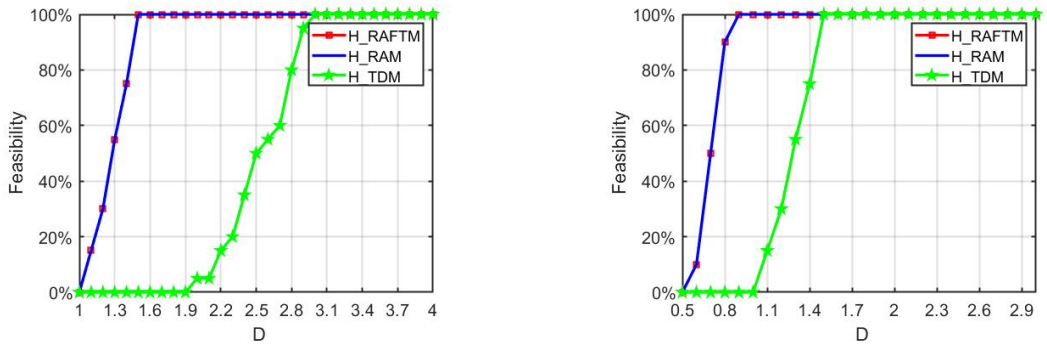
$N = 20,\ M = 2$													
D	**2.4**	**2.6**	**2.8**	**3.0**	**3.2**	**3.4**	**3.6**	**3.6**	**4.0**	**4.2**	**4.4**	**4.6**	**4.8**
O_RAFTM	4.00	6.39	3.10	2.07	2.44	3.78	3.65	4.79	5.50	4.73	4.26	4.25	3.45
H_RAFTM	0.03	~ 0.04											
D	**5.0**	**5.2**	**5.4**	**5.6**	**5.8**	**6.0**	**6.2**	**6.4**	**6.6**	**6.8**	**7.0**	**7.2**	**7.4-7.6**
O_RAFTM	3.46	7.76	28.74	48.56	73.50	98.67	65.51	42.77	6.06	8.28	11.32	13.92	∼0.38
H_RAFTM	~ 0.04												

$N = 20,\ M = 4$							
D	**1.2**	**1.4**	**1.6**	**1.8**	**2.0**	**2.2**	**2.4**
O_RAFTM	237.01	7040.64	1401.37	16611.93	8037.54	24671.73	2041.74
H_RAFTM	~ 0.3						
D	**2.6**	**2.8**	**3.0**	**3.2**	**3.4**	**3.6**	**3.8-6.0**
O_RAFTM	48888.64	76090.41	86462.91	47538.99	202.77	34489.84	∼1.64
H_RAFTM	~ 0.3						

Comparison with heuristic approaches

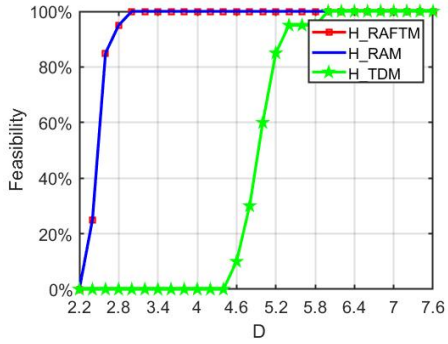
The feasibility of the three heuristics is depicted in Fig. 4.11, where we observe similar trends TL-DVFS scheme. Since the initial task mapping starts with every task and processor assigned with highest frequency, the existence of the feasible solution exists is the same among the different DVFS schemes.

The energy consumption obtained by the solutions of the three heuristics is depicted in Fig. 4.12. Comparing H_RAFTM and H_RAM, we observe that they consume similar energy at very strict deadlines except few cases, where the number of processors is small (Fig 4.12b). In this case, H_RAFTM behaves similarly to H_RAM, i.e., mainly executing the original tasks with the frequency required to achieve the reliability constraint. In few deadlines, such as $D = 0.6$ in Fig 4.12b, H_RAFTM consumes slightly more energy than H_RAM. This occurs because in PL-DVFS scheme, when deciding which potential processors to execute a task, we choose the processors (e.g., θ_1 and θ_2) with EST. If this task has a configuration with both original and duplicated tasks in which the frequencies fit θ_1 and θ_2, this configuration is selected even though the energy consumption is higher. While for H_RAM, since no duplication is used such a configuration does not exist. With deadline relaxing, H_RAFTM starts to consume less energy

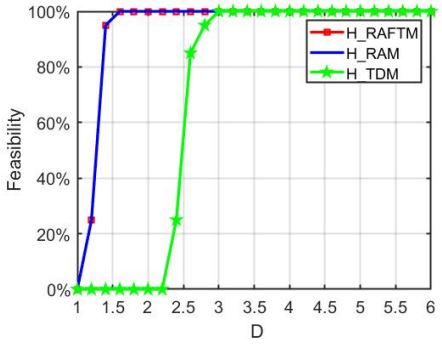


(a) $N = 10, M = 2$

(b) $N = 10, M = 4$



(c) $N = 20, M = 2$



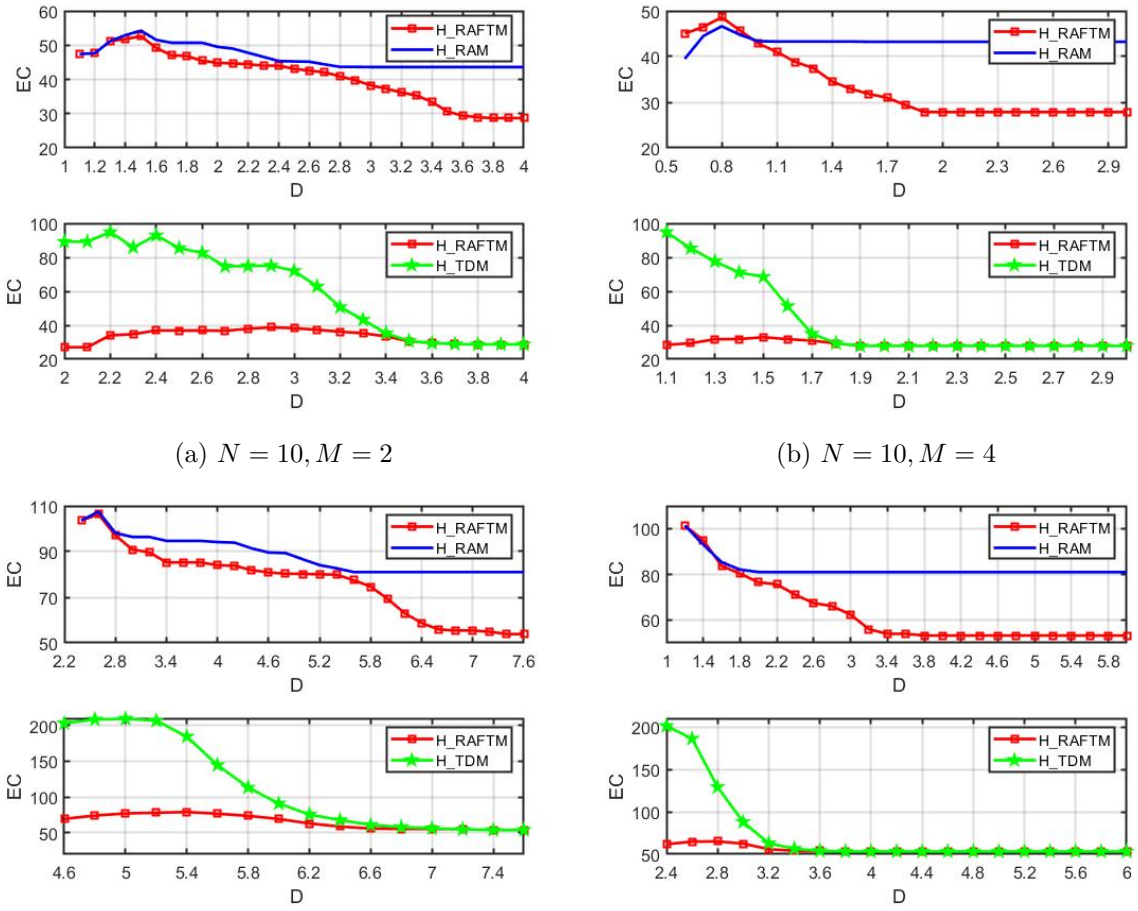
(d) $N = 20, M = 4$

Figure 4.11: Feasibility of heuristics for independent tasks under PL-DVFS

than H_RAM. The gains start to flatten at earlier deadlines with processor number increasing from $M = 2$ to $M = 4$. H_RAFTM achieves gains by exploring the available time slack to duplicate tasks in order to save energy, e.g., up to 55.2% for $N = 10$ and 52.2% for $N = 20$ at relaxed deadlines. Comparing H_RAFTM and H_TDM, the observations are similar to TL-DVFS, i.e., H_TDM cannot find solutions in very strict deadlines as it applies task duplication for every task. When H_TDM becomes able to find solutions at relatively relaxed deadlines, the solutions of H_TDM consume much more energy than H_RAFTM. In very relaxed deadlines, H_RAFTM and H_TDM behave similarly, i.e., duplicate the tasks with configurations when less energy is consumed.

The reliability improvement obtained by the solutions of the heuristics is depicted in Fig. 4.13. The observations are similar to TL-DVFS scheme, shown in Section 4.1.2. The difference is that H_RAM achieves higher reliability than H_RAFTM, except in very strict deadlines. We remind that RI is the reliability improvement computed as the difference of actual reliability and the reliability threshold. Therefore, all approaches satisfy the reliability constraint. Even through H_RAM achieves a higher RI in few strict deadlines, it has higher energy consumption.

The computation time of H_RAFTM, H_RAM and H_TDM heuristics is depicted in



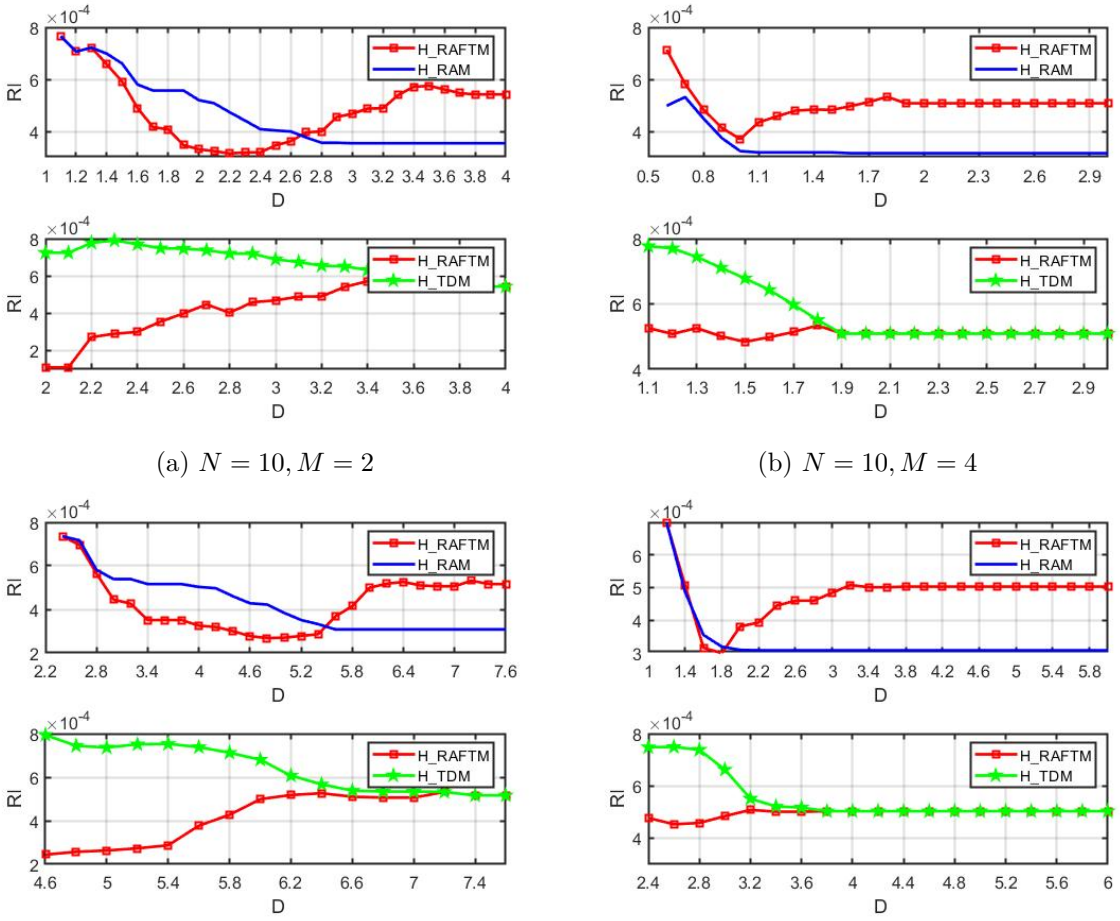
(a) $N = 10, M = 2$

(b) $N = 10, M = 4$

(c) $N = 20, M = 2$

(d) $N = 20, M = 4$

Figure 4.12: Energy consumption (mJ) of heuristics for independent tasks under PL-DVFS scheme.



(a) $N = 10, M = 2$

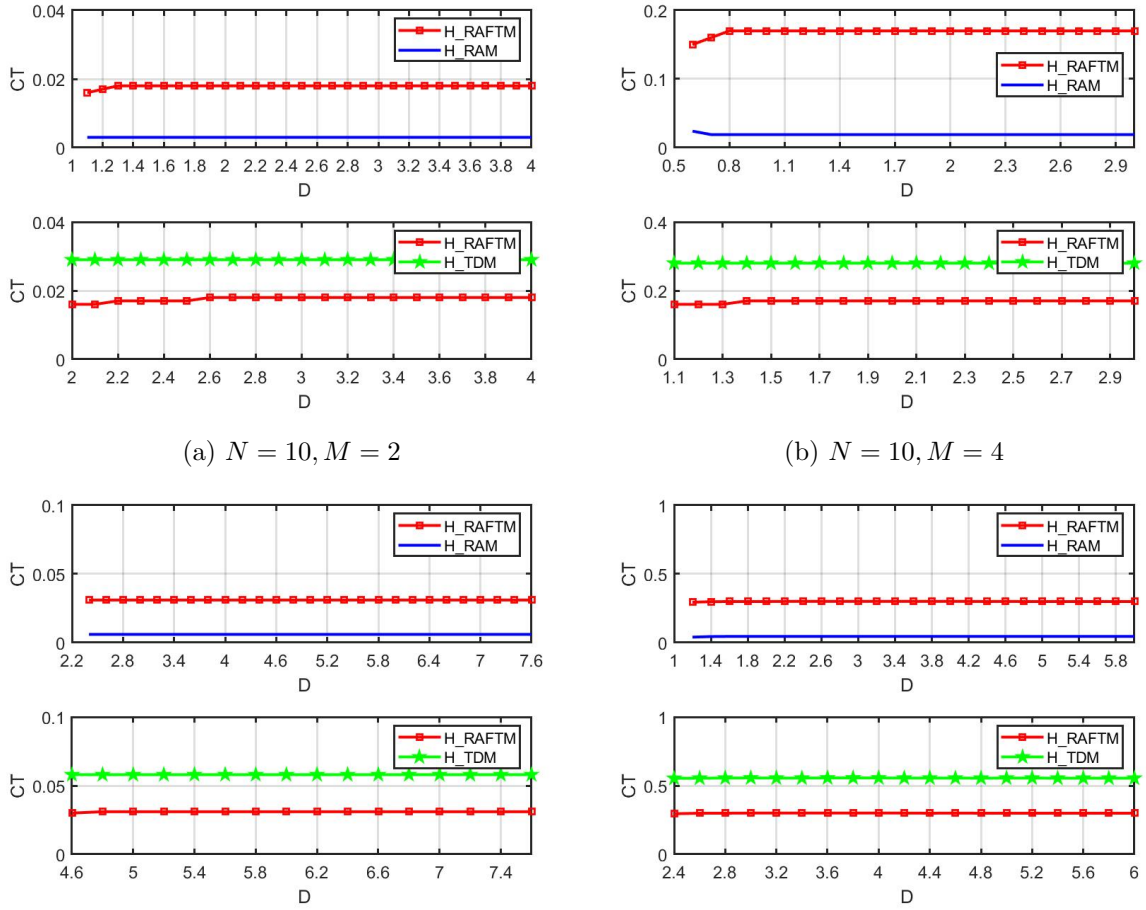
(b) $N = 10, M = 4$

(c) $N = 20, M = 2$

(d) $N = 20, M = 4$

Figure 4.13: Reliability improvement of heuristics for independent tasks under PL-DVFS scheme.

Fig. 4.14. The observations are similar to TL-DVFS, i.e., H_TDM takes the most time to obtain the solution and H_RAM takes the least time, while our approach is in between. The computation time stays stable when the deadline increases. Under PL-DVFS if available time slack exists, different FTP combinations are explored for relaxation. For each FTP combination, the available configurations per task are obtained. The total number of available configurations per task is similar and as the heuristics do task mapping based on the available configurations of each FTP combination, the computation time does not change a lot.



(a) $N = 10, M = 2$

(b) $N = 10, M = 4$

(c) $N = 20, M = 2$

(d) $N = 20, M = 4$

Figure 4.14: Computation time (sec) of heuristics for independent tasks under PL-DVFS scheme.

4.3 Independent Tasks under System Level DVFS

4.3.1 Reliability-aware Fault-tolerant Task Mapping heuristic

In this part, we present the proposed Reliability-aware Fault-tolerant Task Mapping heuristic (H_RAFTM) adapted for SL-DVFS scheme.

Phase A: Task configurations under reliability constraint.

Since in SL-DVFS all processors have the same frequency, all tasks are executed at same frequency. Different from TL-DVFS and PL-DVFS, the Reliability, execution Time, Energy consumption (RTE) table (L. 1) is created based on all possible configurations that include either only the original task or the original and duplicated tasks with same frequency. The remaining pruning (obtain Feasible Configurations (FC) (L. 3) space, the Possible Configurations (PC)

space (L. 6) and the ranked PCs in increasing energy consumption (rPC) (L. 8)) are the same as in Section 4.2.1.

Phase B: Application mapping under real-time constraint

`Phase B` uses `Phase A` task configurations and performs the application mapping, subject to Deadline constraint introduced in Equation 4.1. `Phase B` consists of three steps (L. 10-26):
Step 1 (L. 10-14): Similar to PL-DVFS, the Priority List of tasks (PL-T) is ordered in decreasing rank value (L. 10-13). We define a variable called Frequency-to-system (FTS) assignment, which represents the frequency assigned to all processors of the system. We list all possible L Frequency-to-system (FTS) assignment groups in (L. 14) and put them in frequency decreasing order in order to obtain the ranked rFTS space.

Algorithm 5 Proposed H_RAFTM algorithm for independent tasks under SL_DVFS scheme.

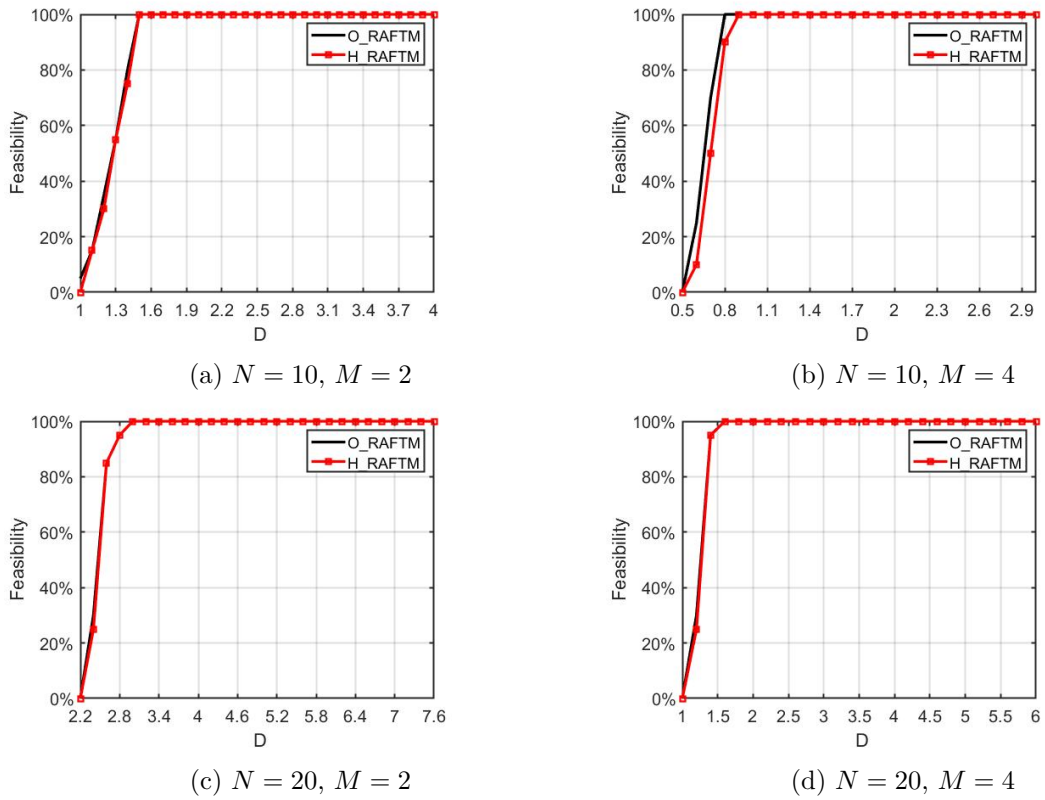
Input: Task graph (G) and set of processors (M).
Output: Application mapping (AM).

`// Phase A`
1: **for** each task τ_i in N **do**
2: $RTE_i = \{C_i^j$: C_i^j is the *j-th* configuration of $\tau_i\}$;
3: $FC_i = RTE_i$ - $\{C_i^j$: $R_i < R_i^{th}\}$;
4: $BC_i = \{FC_i$: $f_i^d = 0\}$;
5: **for** each *bc* in BC_i of task τ_i **do**
6: $PC_i = FC_i$ - $\{FC_i$: $f_i^d \neq 0$ $\}$;
7: **end for**
8: $rPC_i = \{PC_i$: $PC_i[j]$ increasing energy consumption$\}$;
9: **end for**
`// Phase B`
10: **for** each task τ_i in $\boldsymbol{N}$ **do**
11: Compute $rank_i$;
12: **end for**
13: PL-T = $\{N$: ordered in decreasing $rank_{\tau_i}\}$;
14: Obtain all frequency-to-system groups (FTS) and put them in order of frequency decreasing;
15: Start with all processor in highest frequency f_{L-1}, i.e., $FTS = \{f_{L-1}, \ldots, f_{L-1}\}$
16: **for** each task τ_i in PL-T **do**
17: Obtain available configurations (AC);
18: Compute $TM_i^{SC_i}$ (θ_m with $\min_{m \in \boldsymbol{M}} TotalET_m$);
19: **end for**
20: $AM_0 = \{\mathrm{TM}_i^{SC_i}, \mathrm{i} \in \boldsymbol{N}\}$;
21: Compute $TotalET_m^{AM_0}$ of AM_0 for each processor θ_m;
22: **if** $\exists \mathrm{TotalET}_m^{AM_0} > D$ **then**
23: Infeasible problem, algorithm stops.
24: **else if** $\forall \mathrm{TotalET}_m^{AM_0} = D$ **then**
25: $AM = AM_0$, algorithm stops.
26: **else if** $\exists \mathrm{TotalET}_m^{AM_0} < D$ **then**
27: AM relaxation (Algorithm 6);
28: **end if**

Algorithm 6 Mapping Relaxation algorithm for independent tasks under SL_DVFS scheme.

1: $AM = AM_0$, $TotalET = TotalET^{AM_0}$;
2: **while** $TotalET < D$ $(\exists\theta)$ and $|rFTS| > 1$ **do**
3: **for** each FTS in $rFTS$ **do**
4: **for** every task τ_i in PL-T **do**
5: List all available configurations (AC);
6: Compute $TM_i^{SC_i}$ $(\theta_m with \min_{m \in \boldsymbol{M}} TotalET)$
7: **end for**
8: $AM = \{TM_i^{SC_i}, i \in \boldsymbol{N}\}$;
9: Compute $TotalET_m^{AM}$ of AM for each processor θ_m;
10: **end for**
11: remove FTS from $rFTS$
12: **end while**

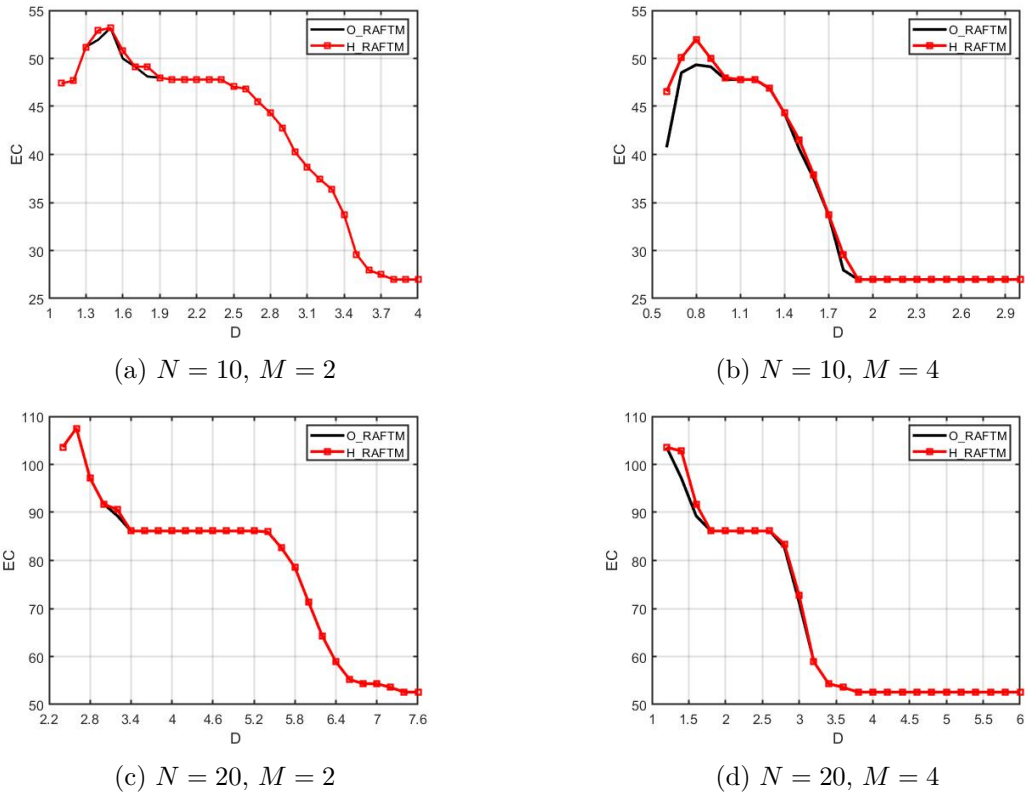
Step 2 (L. 15-25): The initial application mapping AM_0 is generated with $FTS = \{f_{L-1}, \ldots, f_{L-1}\}$, where all processors are assigned with the highest frequency f_{L-1} (L. 15). For each FTS group, there is only one available configuration (AC) for each task from the PC space (L. 17). Two available cores with least $TotalET$ are chosen to execute the original and (potential) duplicated



(a) $N = 10$, $M = 2$ (b) $N = 10$, $M = 4$

(c) $N = 20$, $M = 2$ (d) $N = 20$, $M = 4$

Figure 4.15: Feasibility of optimal and heuristic approaches for independent tasks under SL-DVFS scheme.



(a) $N = 10$, $M = 2$ (b) $N = 10$, $M = 4$

(c) $N = 20$, $M = 2$ (d) $N = 20$, $M = 4$

Figure 4.16: Energy consumption (mJ) of optimal and heuristic approaches for independent tasks under SL-DVFS scheme.

task τ_i (L. 18), as in PL-DVFS. After obtaining initial application mapping (L. 20), according to the *TotalET* of each task, we decide whether the problem is infeasible (L. 22-23) or the initial application mapping is the final result(L. 24-25).

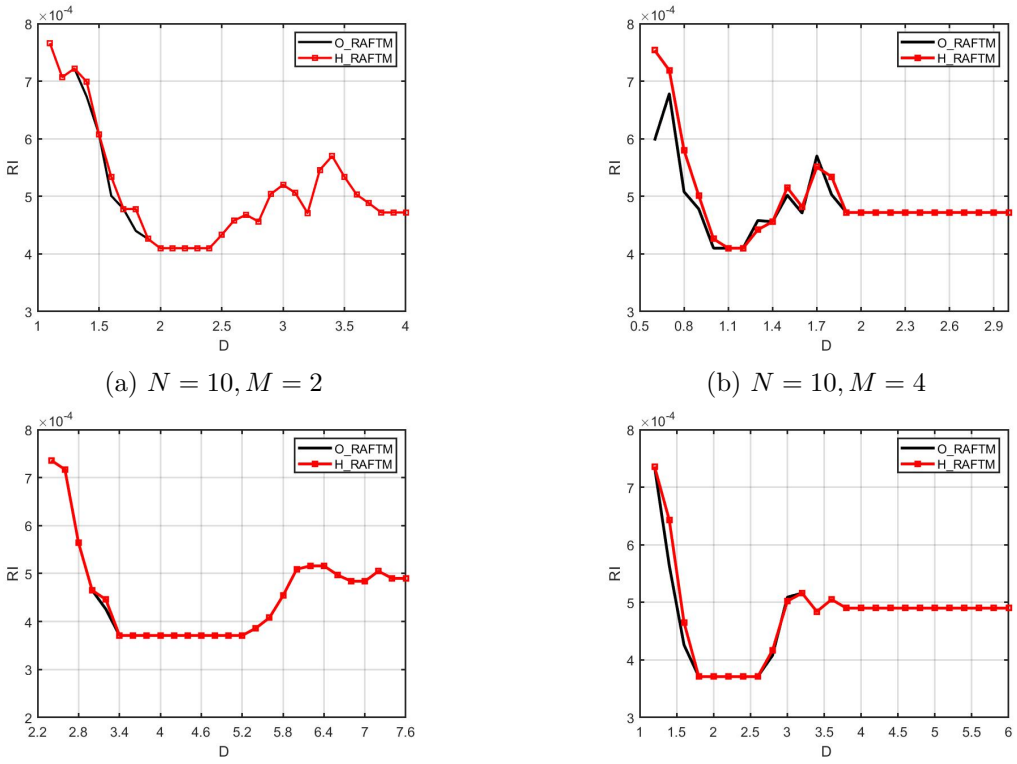
Step 3: (L. 26-28) Otherwise, if a processor has time slack (L. 26), the frequency assignment can be relaxed. Initially, the current mapping and the *TotalET* for each processor is initialised with the initial mapping (L. 1). Algorithm 6 explores every FTS group iteratively to obtain the corresponding application mapping until the end conditions are met, similar to the relaxation algorithm for PL-DVFS.

4.3.2 Evaluation results

Comparison with optimal approach

Regarding feasibility, it remains the same as TL-DVFS and PL-DVFS schemes, as shown in Fig. 4.15, since the initial task mapping starts always with the highest frequency.

Regarding energy consumption in Fig. 4.16, in general, H_RAFTM consumes slightly more energy than O_RAFTM in very strict deadlines. Compared to the behavior observed in TL-



(a) $N = 10, M = 2$ (b) $N = 10, M = 4$

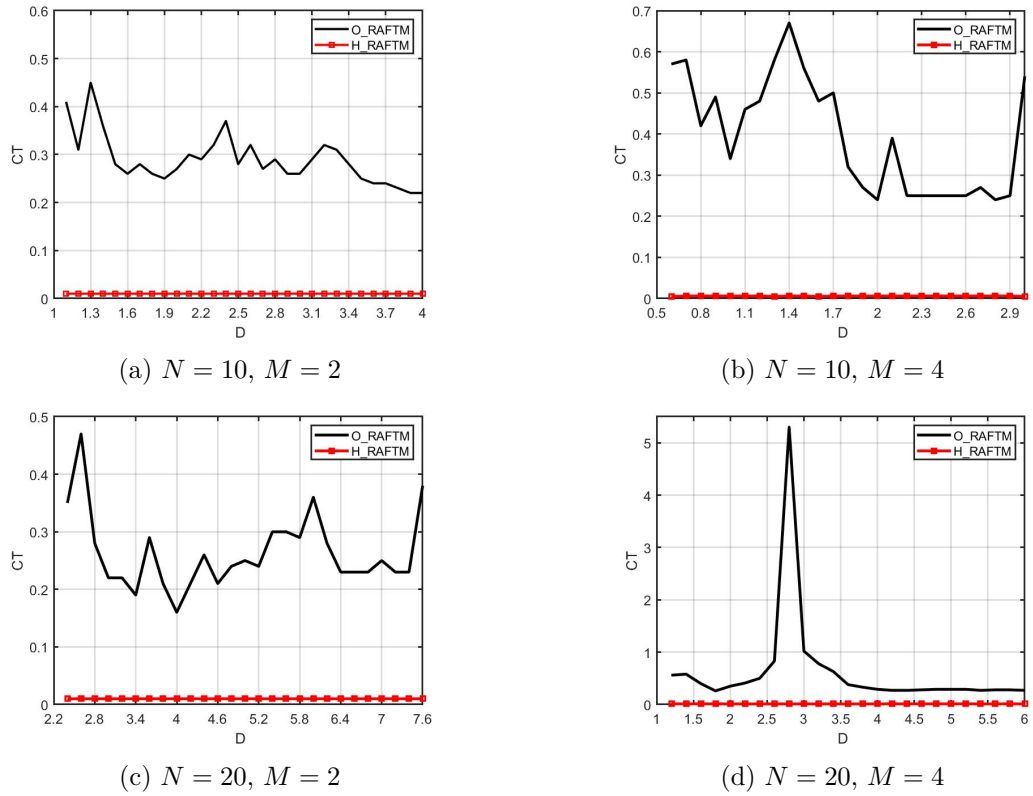
(c) $N = 20, M = 2$ (d) $N = 20, M = 4$

Figure 4.17: Reliability improvement of optimal and heuristic approaches for independent tasks under SL-DVFS scheme.

DVFS and PL-DVFS schemes, the difference between the proposed heuristic and the optimal approach remains small for SL-DVFS. This behavior is explained since SL-DVFS has the least flexibility in frequency assignment. In more details, before the EC curves overlap, H_RAFTM consumes on average 0.7% and 2.6% for $N = 10$ when $M = 2$ and $M = 4$, 0.28% and 1.1% for $N = 20$ when $M = 2$ and $M = 4$ more energy than the optimal solution. When deadline is relaxed, H_RAFTM and O_RAFTM obtain solutions with the same energy consumption.

Regarding reliability improvement, H_RAFTM can provide slightly higher reliability improvement than optimal solution at the price of consuming slightly more energy at strict deadlines, as depicted in Fig. 4.17. The difference of reliability improvement between H_RAFTM and O_RAFTM remains compared to the other two DVFS schemes as explained above.

Fig 4.18 shows the average computation time of O_RAFTM and H_RAFTM under SL-DVFS in seconds per deadline D. Similar to TL-DVFS and PL-DVFS schemes, the proposed heuristic consumes significanlty less time to find a solution than the optimal approach. Moreover, since SL-DVFS has the worst flexibility in frequency assignment, for both optimal and heuristic approaches, the computation to find a solution is largely reduced compared to the other two



(a) $N = 10$, $M = 2$ (b) $N = 10$, $M = 4$

(c) $N = 20$, $M = 2$ (d) $N = 20$, $M = 4$

Figure 4.18: Computation time (sec) of optimal and heuristic approaches for independent tasks under SL-DVFS scheme.

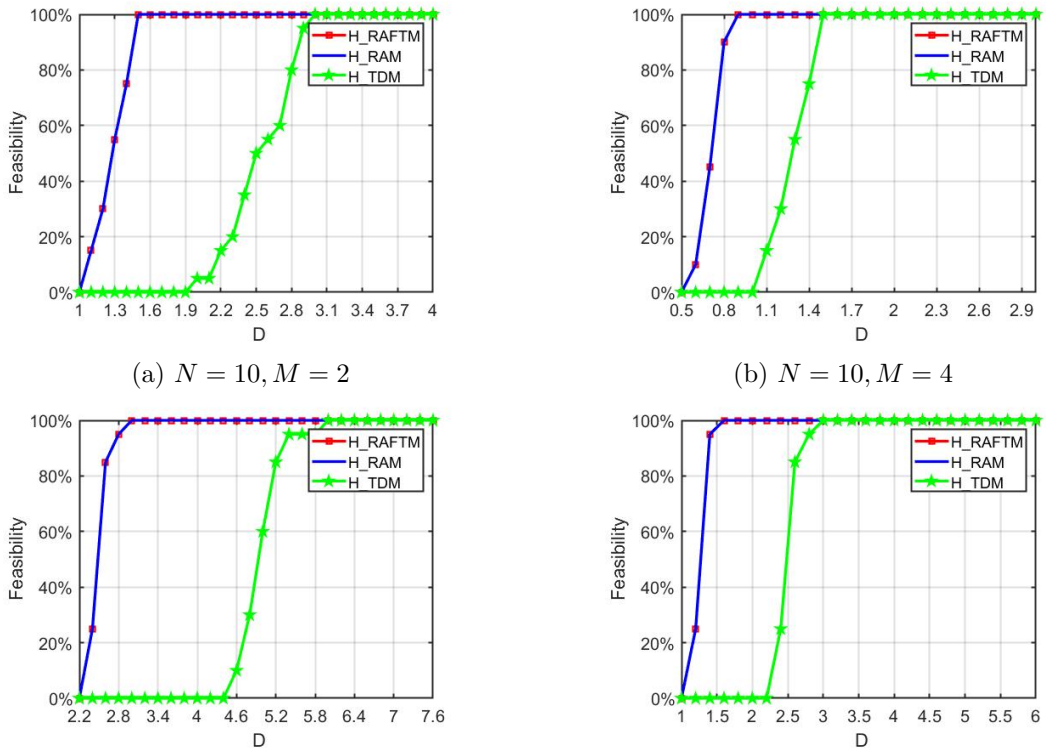
DVFS schemes.

Overall, the obtained results show that i) H_RAFTM provides near-optimal solutions, and ii) as expected, H_RAFTM takes less time to obtain the results compared to the optimal approaches, under all three DVFS schemes.

Comparison with other heuristics

The feasibility of the three heuristics under SL_DVFS is depicted in Fig. 4.19. The trend is similar to TL-DVFS and PL-DVFS. As explained previously, whether a feasible solution exists or not is similar among the different DVFS schemes.

The energy consumption is depicted in Fig. 4.20. The observations are similar to those when we compared the heuristics under TL-DVFS and PL-DVFS schemes. Overall, we observe that slightly more energy is consumed for each heuristic as the flexibility of three DVFS schemes decreases from TL-DVFS to SL-DVFS. For example, when $N = 10$ with $M = 2$, in relaxed deadlines, the final energy consumption of H_RAFTM, H_RAM and H_TDM is 27.3 mJ, 41.9 mJ and 27.3 mJ under TL-DVFS, 28.7 mJ, 43.6 mJ and 28.7 mJ under PL-DVFS, and 30.4 mJ,



(a) $N = 10, M = 2$ (b) $N = 10, M = 4$

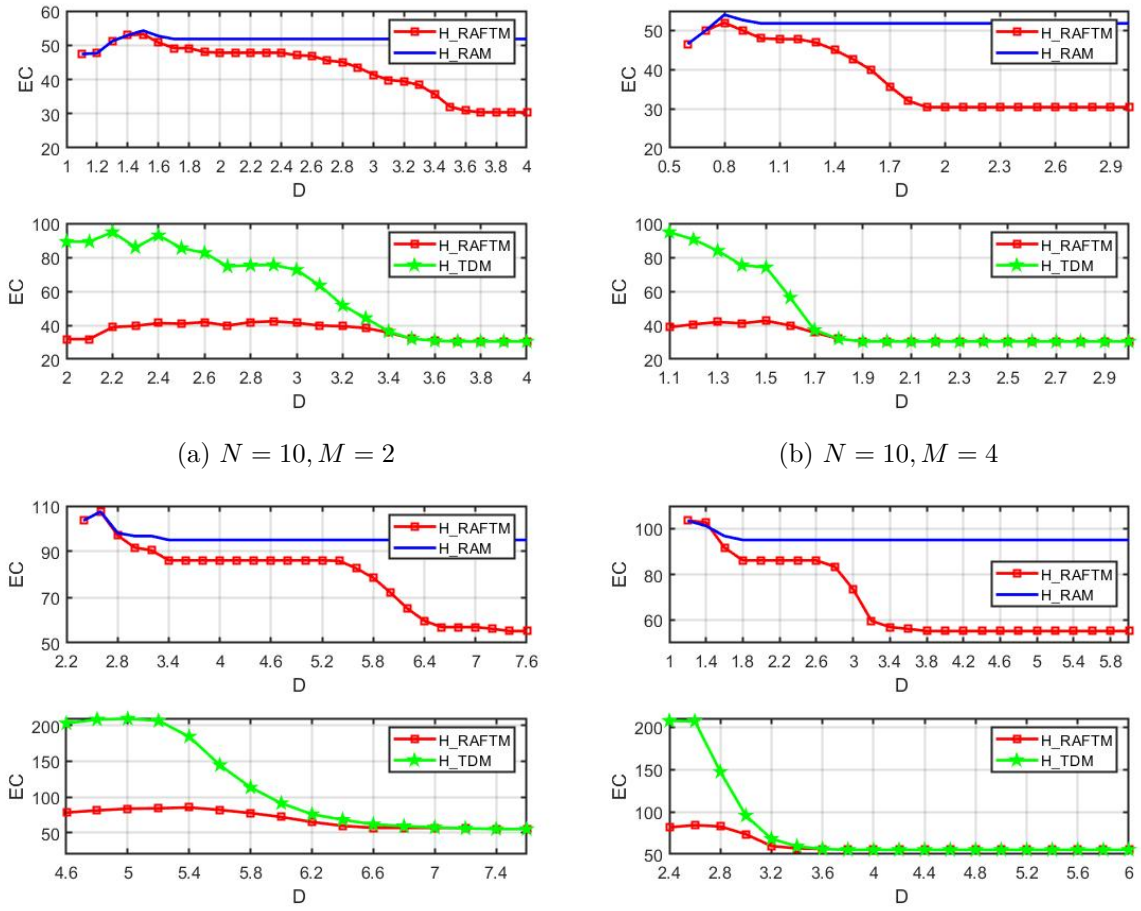
(c) $N = 20, M = 2$ (d) $N = 20, M = 4$

Figure 4.19: Feasibility of heuristics for independent tasks under SL-DVFS scheme.

51.8 mJ, and 30.4 mJ under SL-DVFS. This behavior is because with decreasing DVFS flexibility, all heuristics have more constraints in frequency assignemnt and thus energy consumption. Since SL-DVFS is the least flexible one, if a task needs high frequency to meet its reliability constraint, it forces the remaining tasks to be executed in a higher frequency than they require, increasing energy consumption.

The reliability improvement is depicted in Fig. 4.21. H_RAFTM and H_RAM achieve same reliability improvement at strict deadlines, while H_RAM achieves higher reliability improvement than H_RAFTM, when the deadline is relaxed, but it consumes more energy. We remind that as long as the reliability improvement is positive, the reliability constraint is satisfied. H_TDM achieves higher reliability than H_RAFTM before relaxed deadlines and same reliability improvement as H_RAFTM at relaxed deadlines.

The computation time of H_RAFTM, H_RAM and H_TDM heuristics under SL-DVFS is low. For each heuristic, when Phase A finishes, few possible configurations (PC) are kept. Thus, it takes less time to obtain application mapping then TL-DVFS and PL-DVFS. For instance, when $N = 20$ with $M = 4$, the time to obtain a solution for H_RAFTM, H_RAM and H_TDM is 0.01 sec, 0.003 sec and 0.01 sec, respectively.



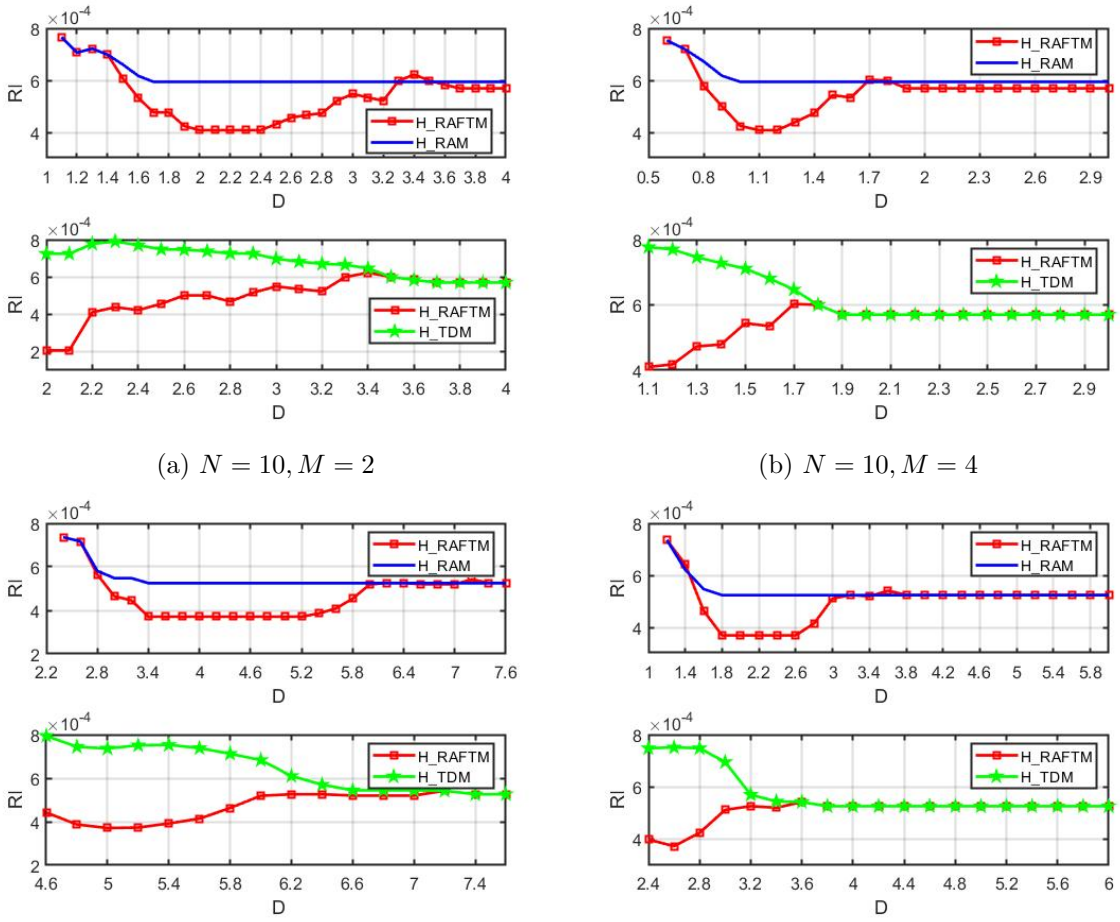
(a) $N = 10, M = 2$

(b) $N = 10, M = 4$

(c) $N = 20, M = 2$

(d) $N = 20, M = 4$

Figure 4.20: Energy consumption (mJ) of heuristics for independent tasks under SL-DVFS scheme.



(a) $N = 10, M = 2$

(b) $N = 10, M = 4$

(c) $N = 20, M = 2$

(d) $N = 20, M = 4$

Figure 4.21: Reliability improvement of heuristics for independent tasks under SL-DVFS scheme.

4.4 Dependent Tasks under Task Level DVFS

We adapt the proposed heuristic in order to solve the task mapping problem of dependent tasks, described by a DAG task graph $\boldsymbol{G}$, over M processors, with the goal of minimizing the total energy consumption by deciding the: 1) task duplication, 2) assignment of frequencies to tasks, 3) allocation of tasks to processors, 4) start time of tasks, subject to reliability, real-time and task precedence constraints. Table 4.4 summarises the main notations and definitions.

Notations	**Definitions**
τ_i^o/τ_i^d	the original/duplicated copy of task τ_i
(v_l, f_l)	the l^{th} voltage/frequency level
W_i	WCEC of task τ_i
D	the global deadline
R_i^{th}	reliability threshold of task τ_i
SL	schedule length of DAG G
EST_i	earliest start time of task τ_i
LFT_i	latest finish time of task τ_i
st_i	actual start time of task τ_i
ft_i	actual finish time of task τ_i
et_i	execution time of task τ_i
$slack_i$	time slack of task τ_i
$Pred\{\tau_i\}$	all immediate predecessors of task τ_i
$Succ\{\tau_i\}$	all immediate successors of task τ_i
$proc\{m\}$	task set that are allocated on processor θ_m
$avail\{m\}$	earliest available time of processor θ_m to execute a task
SC_i	Scheduled configuration of task τ_i in current task mapping
NC_i	New checked configuration of task τ_i to be selected to do relaxation

Table 4.4: Main notations for dependent tasks under TL-DVFS scheme.

4.4.1 Reliability-aware Fault-tolerant Task Mapping heuristic

In order to leverage the proposed heuristic for dependent tasks under TL-DVFS scheme, we extend the definitions and the constraints presented in Section 4.1.1 in order to include the precedence constraints.

Definition 5 (Task Mapping)**.** *A mapping of a task τ_i, under the task configuration C_i^j, is denoted as $TM_i^{C_i^j} = \{C_i^j, \theta_i^o, \theta_i^d, st_i^o, st_i^d\}$, where θ_i^o (θ_i^d) is the allocated processor, and st_i^o (st_i^d) is the start time of the original (duplicated) task. If a task is not duplicated, then $f_i^d = 0$, the duplicated task takes no execution time, i.e., $st_i^d = ft_i^d$.*

Definition 6 (Application Mapping)**.** *The mapping of the application (AM) is given by the set of mappings of N original tasks and $S \subseteq N$ duplicated tasks. The mapping is valid if task precedence and real-time constraints are satisfied.*

Constraint 3 (Precedence constraints)**.** *Based on the dependencies defined by the task graph, a task τ_i can start execution only when all task predecessors (including duplicated tasks) are completed. Then, the Earliest Start Time (EST) of τ_i is*

$$EST_i = \begin{cases} 0, & if\ \tau_i = \tau_{entry} \\ \max\limits_{\tau_j \in Pred\{\tau_i\}} \{EFT_j\}, & else \end{cases} \tag{4.3}$$

where $Pred\{\tau_i\}$ is the set of τ_i's predecessors, and $EFT_j = EST_j + et_j$ is the Earliest Finish Time (EFT) of task τ_j.

Constraint 4 (Deadline constraint)**.** *The application must finish before the deadline D. The schedule length SL of task graph G, under a given application mapping AM, is determined by the latest finish time of exist tasks:*

$$SL_{AM} = \max\{ft_{\tau_{exit}}\} \leq D. \tag{4.4}$$

Due to precedence constraints, the start time of a task is $st_i \geq EST_i$. Our goal is to exploit the available time slack to save energy, thus, we initially consider that tasks start execution as soon as possible, i.e., $st_i = EST_i, \forall \tau_i$, in order to increase the probability of executing all tasks without exceeding the deadline.

Constraint 5 (Non-overlapping constraint)**.** *Only a single task should be executed on a processor at a given time instance. Taking into account the earliest available time, $avail[m]$, when processor θ_m is ready to execute a task, EST_i is modified as*

$$EST_i = \begin{cases} 0, if\ \tau_i = \tau_{entry} \\ \max \left\{ \begin{array}{l} \max\limits_{\tau_j \in Pred\{\tau_i\}} \{EFT_j\}, \\ avail[m], \end{array} \right\}, else \end{cases} \tag{4.5}$$

Algorithm 7 describes the two phases of the proposed heuristic.

Phase A: Task configurations under reliability constraint.

`Phase A` (L. 1-9) in Algorithm 7 is same as Phase A of Section 4.1.1 under TL-DVFS scheme, where we consider only the reliability constraint.

Phase B: Application mapping under precedence and real-time constraints.

`Phase B` consists of three steps (L. 10-26):

Step 1 (L. 10-13): Priorities are given to tasks for task allocation based on the upward rank value [24], $rank_i$, which is common for original and duplicated tasks (L. 10-12):

Algorithm 7 Proposed H_RAFTM algorithm for dependent tasks under TL-DVFS scheme.

Input: Task graph (G) and set of processors (M).
Output: Application mapping (AM).

`// Phase A`
1: **for** each task τ_i in $\boldsymbol{N}$ **do**
2: $RTE_i = \{C_i^j$: C_i^j is a configuration of $\tau_i\}$;
3: $FC_i = RTE_i$ - $\{C_i^j$: $R_i < R_i^{th}\}$;
4: $BC_i = \{FC_i$: $f_i^d = 0\}$;
5: **for** each configuration bc in BC_i **do**
6: $PC_i = FC_i$ - $\{FC_i$: $f_i^d \neq 0 \wedge \min\{et_i^o, et_i^d\} \geq et^{bc} \wedge \sum\{E_i^o, E_i^d\} > E^{bc}\}$;
7: **end for**
8: $rPC_i = \{PC_i$: PC_i decreasing energy consumption$\}$;
9: **end for**
`// Phase B`
10: **for** each task τ_i in $\boldsymbol{N}$ **do**
11: Compute $rank_i$ (Eq. (4.6));
12: **end for**
13: PL-T = $\{N$: ordered in decreasing $rank_i\}$;
14: **for** each task τ_i in PL **do**
15: $SC_i = rPC_i[0]$;
16: Compute $TM_i^{SC_i}$ ($st_i = EST_i$ in Eq. (4.5));
17: **end for**
18: $AM_0 = \{TM_i^{SC_i}\}$;
19: Compute SL_{AM_0};
20: **if** $SL_{AM_0} > D$ **then**
21: Infeasible problem, algorithm stops.
22: **else if** $SL_{AM_0} = D$ **then**
23: $AM = AM_0$, algorithm stops.
24: **else if** $SL_{AM_0} < D$ **then**
25: AM relaxation (Algorithm 8);
26: **end if**

$$rank_i = \begin{cases} \overline{et}_i, & if\ \tau_i = \tau_{exit} \\ \overline{et}_i + \max\limits_{\tau_j \in Succ\{\tau_i\}} \{rank_j\}, & else \end{cases} \tag{4.6}$$

where $\overline{et}_i = (\sum_{l=1}^{L} W_i/f_l)/L$ is the average computation time of τ_i and $Succ\{\tau_i\}$ the immediate successors of τ_i. The Priority List of tasks (PL-T) is ordered in decreasing rank value (L. 13).

Step 2 (L. 14-23): The initial application mapping AM_0 is generated to check if the problem is feasible and time slack is available. For all task, AM_0 uses the first configuration in rPC_i as the Selected Configuration SC_i (L. 15). Setting $st_i = EST_i$ to Equation 4.5, we obtain the task mapping ($TM_i^{SC_i}$) per task (L. 16). The set of all task mappings provides the AM_0 (L. 18) and its schedule length SL_{AM_0} is obtained (L. 19). If it is higher than the deadline, the problem is

Algorithm 8 Mapping Relaxation Algorithm for dependent tasks under TL-DVFS scheme.

1: $AM = AM_0$, $SL = SL_{AM_0}$;
2: **while** $SL < D$ **and** $rPC_i > 1 (\forall \tau_i)$ **do**
3: **for** each task τ_i in $\boldsymbol{N}$ **do**
4: $NC_i^A = rPC_i[0]$;
5: $NC_i^B = rPC_i[j]$ with $\max \left\{ \sum_{k \in \{o,d\}} \left(ES^j_{\tau_i^k} / TI^j_{\tau_i^k} \right) \right\}$;
6: $NC_i = (E^{NC_i^A} \leq E^{NC_i^B} ? NC_i^A : NC_i^B)$;
7: Compute $TM_i^{NC_i}, i \in \boldsymbol{N}$ ($st_i = EST_i$ in Eq. (4.5));
8: $AM_i = \left\{ TM_i^{NC_i} \right\}$;
9: Compute SL_{AM_i};
10: $SLI_i = SL_{AM_i} - SL$;
11: $Gain_i = \frac{\sum_{k \in \{o,d\}} \left(ES^{NC_i}_{\tau_i^k} / TI^{NC_i}_{\tau_i^k} \right)}{SLI_i}$;
12: **end for**
13: **for** each task τ_i in $\boldsymbol{N}$ **do**
14: Compute $slack_{AM_i}$ (Eqs. (4.4), (4.5), (4.8));
15: **end for**
16: $\tau_{rel} = \tau_i$ with $\arg\max_{i \in \boldsymbol{N}} (Gain_i) \wedge (TI_i^{NC_i} \leq slack_{AM_i})$;
17: $AM = AM_{\tau_{rel}}$ and $SL = SL_{AM_{\tau_{rel}}}$;
18: **for** each configuration pc in $rPC_{\tau_{rel}}$ **do**
19: $rPC_{\tau_{rel}} = rPC_{\tau_{rel}}$ - $\{ rPC_{\tau_{rel}} : E^{pc} \geq E^{SC_{\tau_{rel}}} \}$;
20: **end for**
21: **end while**

infeasible (L. 20-21), and the algorithm stops. If it is equal to the deadline, the initial application mapping is the final mapping and the algorithm stops (L. 22-23).

Step 3: (L. 25-26) If the initial schedule length is less than the deadline (L. 24), time slack exists. Different task configurations and different tasks that can be relaxed, leading to energy savings. Algorithm 8 decides the task to be relaxed and its configuration. Initially, the current mapping (schedule length) is initialised with the initial mapping (schedule length) (L. 1). The algorithm is applied iteratively, until the schedule length reaches the deadline or all tasks reach their configuration with the least energy consumption (L. 2). First, an inner search decides a new configuration per task (L. 3-14). We combine two criteria to select a potential New Configuration (NC_i) for a task. NC_i^A explores rPC_i sequentially, by selecting always the first configuration. NC_i^B selects the configuration in rPC_i with the highest value $(ES^j_{\tau_i^o} / TI^j_{\tau_i^o}) + (ES^j_{\tau_i^d} / TI^j_{\tau_i^d})$, where $ES^j_{\tau_i^o}$ ($ES^j_{\tau_i^d}$) is the energy savings and $TI^j_{\tau_i^o}$ ($TI^j_{\tau_i^d}$) the time increase of task τ_i in configuration j, compared to the current selected configuration SC_i. The final selected configuration NC_i is the one with the minimum energy consumption (L. 6). After selecting a new task configuration, all task mappings are updated accordingly (L. 7-9). The new application mapping AM_i (L. 8) and its schedule length SL_{AM_i} (L. 9) are obtained. The difference of SL_{AM_i} with the schedule length of the current mapping SL provides the Schedule Length Increase (SLI_i), when task τ_i

changes its current configuration SC_i to NC_i (L. 10). With this information, the overall gain ($Gain_i$), considering both energy and time, that this configuration modification will bring to the overall mapping, is computed (L. 11). After the inner search finishes, the global decision takes place (L. 13-21). The time slack of each task in the current mapping ($slack_{AM_i}$) based on task mobility through Eqs. (4.5), (4.7), (4.8) (L. 13-15):

$$slack_{AM_i} = LFT_i - EST_i - et_i. \tag{4.7}$$

The Latest Finish Time (LFT) of τ_i is:

$$LFT_i = \begin{cases} D, if\ \tau_i = \tau_{exit} \\ \min \left\{ \begin{array}{l} \min\limits_{\tau_p \in Proc\{\tau_i\}} \{LST_p\}, \\ \min\limits_{\tau_j \in Succ\{\tau_i\}} \{LST_j\}, \end{array} \right\}, else \end{cases} \tag{4.8}$$

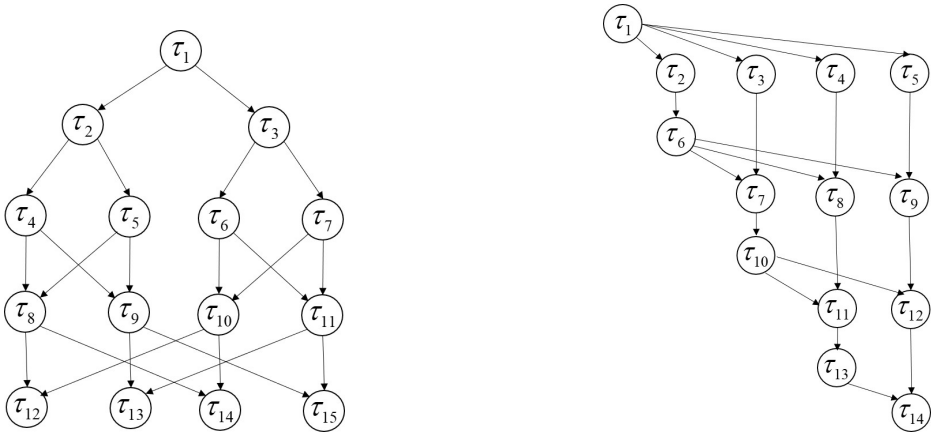
where $LST_i = LFT_i - et_i$ is the Latest Start Time (LST) of task τ_i. $Proc\{\tau_i\}$ is the processor where the task is allocated and τ_p are the tasks, scheduled after task τ_i, on the same processor. The task (and its corresponding configuration) to be relaxed (τ_{rel}) is the task with highest overall gain, whose time increase in this new configuration is not larger than its available slack in the current mapping (L. 16). Last, the selected configuration for the relaxed task, the application mapping and its schedule length are updated (L. 16-17) and all configurations that have a higher energy consumption than the selected one are removed from $rPC_{\tau_{rel}}$ (L. 18-20).

4.4.2 Evaluation results

This section evaluates the proposed heuristic (H_RAFTM) with i) the optimal approach (O_RAFTM) in Section 3.3.4 and ii) two SoA heuristics H_RAM and H_TDM. The parameters for the set-up are provided by Table 3.3 in Section 3.2.4. To compare with optimal solutions, experiments are performed with the number of processors $M = 2$ and $M = 4$ for $N = 10$ original tasks. To compare with other heuristics, we used randomly generated task graphs and graphs obtained both from real-world kernels, i.e., Fast Fourier Transformation (FFT) ($N = 15$) and Gaussian Elimination (GE) ($N = 14$) [24, 62]. Fig 4.22 depicts the shape of parallelism obtained by FFT and GE DAGs. Random generation is used for comparison with optimal approach ($N = 10$) and for evaluation of the scalability of the heuristics (100). A large and diverse set of experiments is performed, by tuning the:

1. Number of processors ($M = 2, 4, 6$).
2. Size of task set ($N = 10, 14, 15, 100$).
3. For each application task graph, a number of experiments (denoted as NE) is performed, each time with different task characteristics (W_i and R_i^{th}).

The Feasibility, Energy Consumption (EC), Reliability Improvement (RI) and Computation time (CT) are presented.



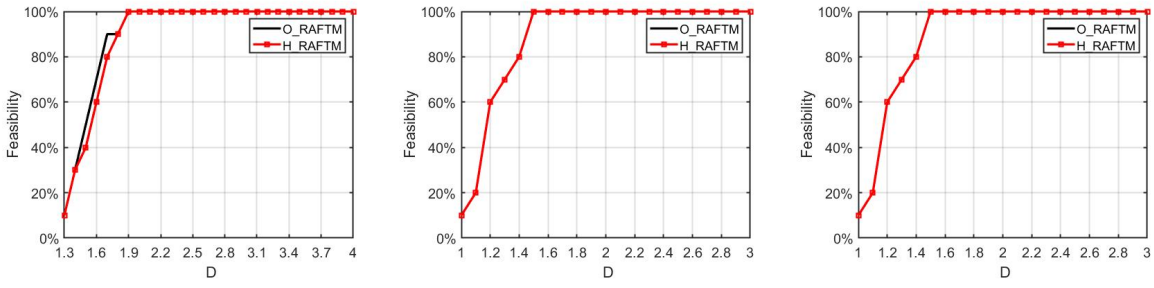
(a) Fast Fourier Transform (FFT) (b) Gaussian Elimination (GE)

Figure 4.22: DAG obtained from real code kernels.

Comparison with optimal approach

We evaluate the obtained solutions when solved by the proposed heuristic (H_RAFTM) and the optimal approach (O_RAFTM) for $NE = 10$ experiments considering random graphs with $N = 10$ tasks, $M = 2$, $M = 4$ and $M = 6$ processors.

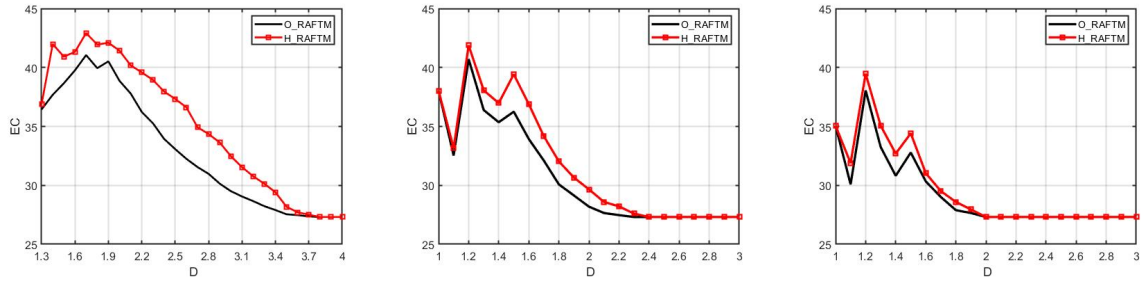
Regarding feasibility, when $M = 2$, the H_RAFTM feasibility is very close to the optimal feasibility, as shown in Fig. 4.23, except for a few cases when the deadline is strict. The average difference before achieving 100% feasibility is 3.3%. With the number of processors increasing to $M = 4$ (Fig. 4.23b) and $M = 6$ (Fig. 4.23c), H_RAFTM and O_RAFTM achieve the same feasibility, since more resources are available to execute the original and potentially duplicated



(a) $N = 10, M = 2$ (b) $N = 10, M = 4$ (c) $N = 10, M = 6$

Figure 4.23: Feasibility of optimal and heuristic approaches for dependent tasks under TL-DVFS scheme.

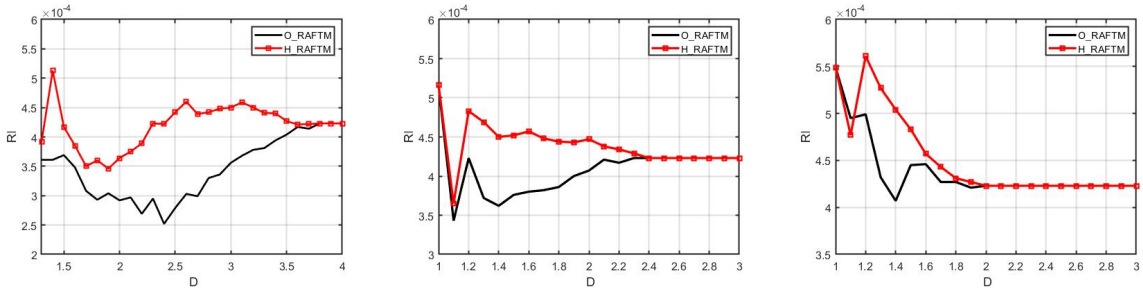
tasks.



(a) $N = 10, M = 2$ (b) $N = 10, M = 4$ (c) $N = 10, M = 6$

Figure 4.24: Energy consumption (mJ) of optimal and heuristic approaches for dependent tasks under TL-DVFS scheme.

Regarding energy consumption in Fig. 4.24, H_RAFTM generally consumes slightly more energy than O_RAFTM. When deadline is relaxed, H_RAFTM and O_RAFTM obtain solutions with the same energy consumption. H_RAFTM consumes on average 6.5% ($M = 2$), 2.9% ($M = 4$) and 1.6% ($M = 6$) more energy than the optimal solutions. With the processor number increasing, the energy consumption of both proposed heuristic and optimal solution flattens at earlier deadlines, since there are more processors available to perform the task mapping, and thus, more opportunities to start the tasks earlier.



(a) $N = 10, M = 2$ (b) $N = 10, M = 4$ (c) $N = 10, M = 6$

Figure 4.25: Reliability improvement of optimal and heuristic approaches for dependent tasks under TL-DVFS scheme.

Regarding reliability improvement, H_RAFTM provides more reliability improvement than optimal solutions at the price of consuming slightly more energy under the same deadlines, as depicted in Fig. 4.25.

The average computation time of O_RAFTM and H_RAFTM is computed over the number of experiments that a feasible solution is found over all NE experiments. Table 4.5 shows the results in seconds per deadline D. It can be observed that although few tasks and processors are used, the time to obtain the optimal solution is very long, on average $\times 10^4$ more than the

Table 4.5: Computation time (sec) of optimal and heuristic approaches for dependent tasks under TL-DVFS scheme.

$N = 10,\ M = 2$									
D	**1.3**	**1.4**	**1.5**	**1.6**	**1.7**	**1.8**	**1.9**	**2.0**	**2.1**
O_RAFTM	424.8	523.4	537.1	275.5	432.2	739.7	832.7	1,645.6	2,349.0
H_RAFTM	0.15	0.13	0.14	0.17	0.17	0.20	0.22	0.23	0.29
D	**2.2**	**2.3**	**2.4**	**2.5**	**2.6**	**2.7**	**2.8**	**2.9**	**3.0**
O_RAFTM	3,358.1	5,368.3	4,543.9	9,335.6	11,974.2	13,038.4	19,364.5	27,312.9	19,378.6
H_RAFTM	0.32	0.31	0.36	0.33	0.36	0.35	0.41	0.42	0.42
D	**3.1**	**3.2**	**3.3**	**3.4**	**3.5**	**3.6**	**3.7**	**3.8-4**	
O_RAFTM	13,155.9	21,493.1	24,228.4	77,477.5	5,472.0	4,868.6	4,096.3	2.5	
H_RAFTM	0.49	0.52	0.47	0.52	0.56	0.58	0.59	0.63	

$N = 10,\ M = 4$											
D	**1.0**	**1.1**	**1.2**	**1.3**	**1.4**	**1.5**	**1.6**	**1.7**	**1.8**	**1.9**	**2.0**
O_RAFTM	14.9	61.0	74.9	138.2	202.1	256.9	327.3	219.0	1039.3	1054.7	126.8
H_RAFTM	0.19	0.26	0.27	0.30	0.35	0.27	0.33	0.43	0.47	0.50	0.57
D	**2.1**	**2.2**	**2.3**	**2.4**	**2.5**	**2.6**	**2.7**	**2.8**	**2.9**	**3.0**	-
O_RAFTM	58.7	123.1	4.9	1.9	1.4	0.6	1.8	1.0	1.8	2.3	-
H_RAFTM	0.59	0.66	0.64	0.70	0.68	0.67	0.68	0.69	0.66	0.78	-

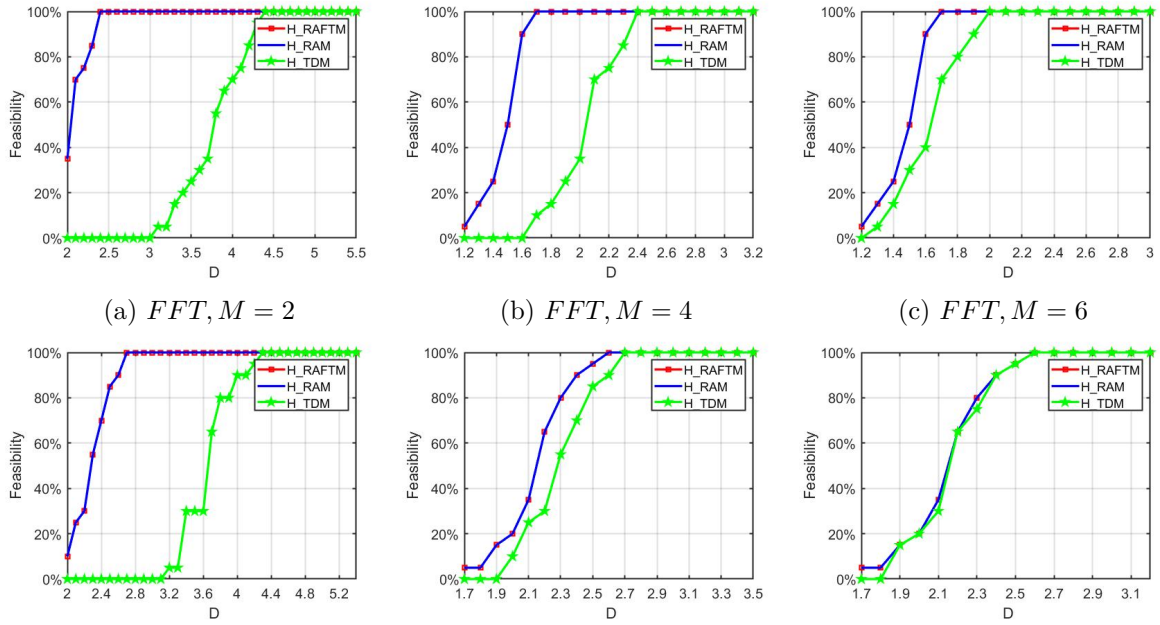
$N = 10,\ M = 6$											
D	**1.0**	**1.1**	**1.2**	**1.3**	**1.4**	**1.5**	**1.6**	**1.7**	**1.8**	**1.9**	**2.0**
O_RAFTM	0.83	58.10	26.95	89.89	81.87	109.82	58.60	62.77	59.77	14.69	2.49
H_RAFTM	0.32	0.27	0.33	0.48	0.54	0.50	0.58	0.68	0.69	0.70	0.67
D	**2.1**	**2.2**	**2.3**	**2.4**	**2.5**	**2.6**	**2.7**	**2.8**	**2.9**	**3.0**	-
O_RAFTM	0.74	0.65	0.63	0.66	0.66	0.64	0.64	0.63	0.60	0.60	-
H_RAFTM	0.76	0.74	0.76	0.80	0.84	0.77	0.76	0.76	0.71	0.72	-

proposed H_RAFTM.

Overall, the conclusions for dependent tasks under TL-DVFS are that i) H_RAFTM provides near-optimal solutions, and the solutions tend to converge to the optimal ones with the number of processors increasing, and ii) as expected, H_RAFTM takes significantly less time to obtain the results compared to the optimal approaches.

Comparison with other heuristics

i) Real code DAGs: The feasibility of the three heuristics for FFT and GE benchmarks with $NE = 20$ experiments is depicted in Fig. 4.26. Comparing to H_TDM, the proposed H_RAFTM can find solutions in significantly more experiments than H_TDM, especially when the deadline is not fully relaxed or number of cores is reduced. When tasks meet their reliability constraint, H_RAFTM does not need to duplicate these tasks. However, H_TDM duplicates all tasks, and thus, it is able to find solutions only when the deadline is relatively relaxed or several processors exist to run the tasks in parallel. Before obtaining 100% feasibility for both approaches, on average, H_RAFTM finds a solution in more experiments than H_TDM, i.e., 70.2% for FFT and 59.4% for GE ($M = 2$), 47.5% for FFT and 14.5% for GE ($M = 4$) and 19.7% for FFT and 2.9% for GE ($M = 6$). Note that, H_RAFTM and H_RAM have the same feasibility. This behavior is explained as follows: when H_RAM finds a solution, it means the reliability



(a) $FFT, M = 2$ (b) $FFT, M = 4$ (c) $FFT, M = 6$

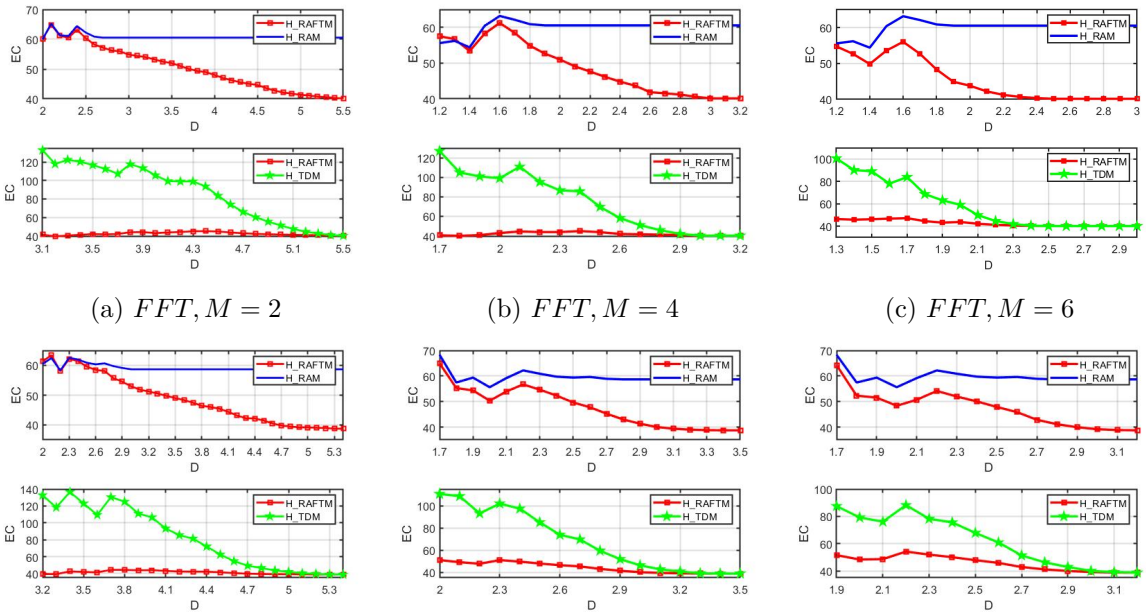
(d) $GE, M = 2$ (e) $GE, M = 4$ (f) $GE, M = 6$

Figure 4.26: Feasibility of heuristics (real-world DAGs) for dependent tasks under TL-DVFS scheme.

constraint of all tasks can be met by executing only the original task with a high frequency. In this case, H_RAFTM is also able to find this solution.

The energy consumption obtained by the solutions of the heuristics for FFT and GE is depicted in Fig. 4.27. Comparing H_RAFTM and H_RAM, we observe that they consume similar energy at very strict deadlines, when the number of processors is small. In this case, H_RAFTM behaves similarly to H_RAM, i.e., mainly executing the original tasks with the frequency required to achieve the reliability constraint. With deadline relaxing, H_RAFTM starts to consume less energy than H_RAM. H_RAFTM achieves this gain by exploring the available time slack to duplicate tasks in order to save energy, e.g., up to ∼50.9% for FFT at relaxed deadlines. Similarly, when more processors are available, H_RAFTM can take advantage of these resources and execute duplicated task in parallel. Comparing H_RAFTM and H_TDM, as H_TDM applies task duplication for every task, it cannot find solutions in very strict deadlines. H_RAFTM consumes significantly less energy than H_TDM. H_RAFTM selects the task configuration, if exists, with only the original task, meeting the reliability constraint and consuming less energy than configurations with duplicated tasks. Since H_TDM duplicates all tasks, its energy consumption can be significant, when it finds a solution. In relaxed deadlines, H_RAFTM behaves similar to H_TDM, i.e. duplicates the tasks when less energy is consumed.

The reliability improvement obtained by the solutions of the three heuristics is depicted in



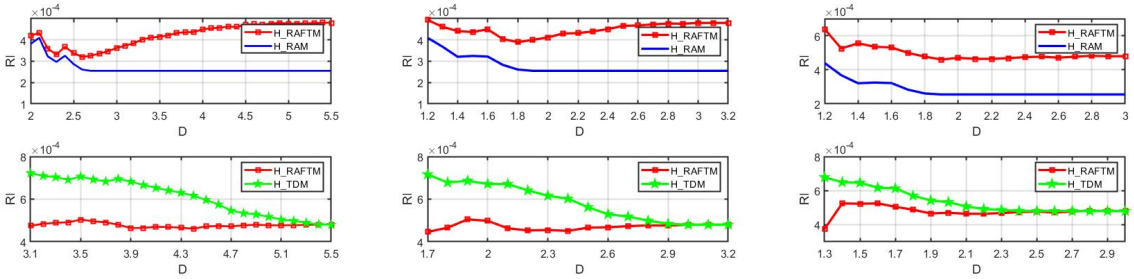
(a) $FFT, M = 2$ (b) $FFT, M = 4$ (c) $FFT, M = 6$

(d) $GE, M = 2$ (e) $GE, M = 4$ (f) $GE, M = 6$

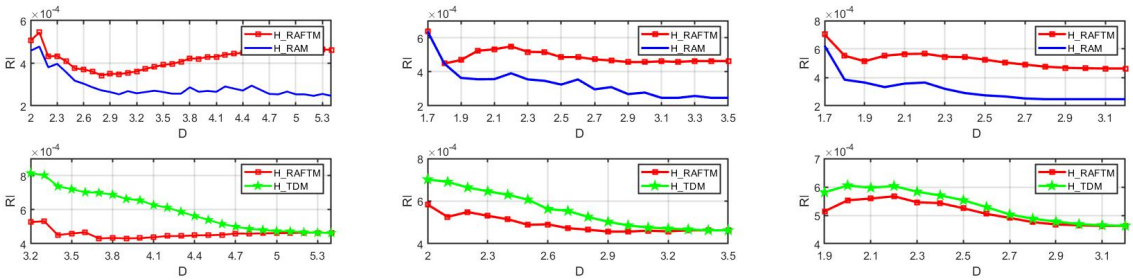
Figure 4.27: Energy consumption (mJ) of heuristics (real-world DAGs) for dependent tasks under TL-DVFS scheme.

Fig. 4.28. H_RAFTM achieves higher reliability than RAM, except in very strict deadlines when H_RAFTM behaves similar to H_RAM without task duplication. As explained above, this is because most of the tasks are executed with only their original copy when there is not available time slack to perform duplication at strict deadlines. Compared to H_TDM, H_RAFTM provides lower reliability for tight deadlines, as it duplicates only a part of the task-set. The same reliability improvement can be achieved in relaxed deadlines, since both H_RAFTM and H_TDM duplicate tasks similarly.

The computation time of H_RAFTM, H_RAM and H_TDM heuristics is depicted in Fig. 4.29. Overall, when the deadline increases, the trend of computation time for H_RAM is to remain stable, for H_RAFTM to slightly increase and for H_TDM to increase with a higher factor. The computation time to obtain a feasible solution increases with deadline relaxing, due to the fact that the proposed heuristic explores the PC space for each task, based on the deadline constraints. Therefore, the more relaxed the deadline is, the larger is the PC space to be explored per task, and thus, more time is needed. Note that, H_TDM is the most expensive approach in terms of computation time. This behavior is due to the fact that all tasks are duplicated, which increases the total number of tasks to be scheduled and the number of PCs in each task PC space, and thus, the time to find a solution. For H_RAM, it only executes original tasks, thus it has a reduced number of PCs in the PC space, taking the least time

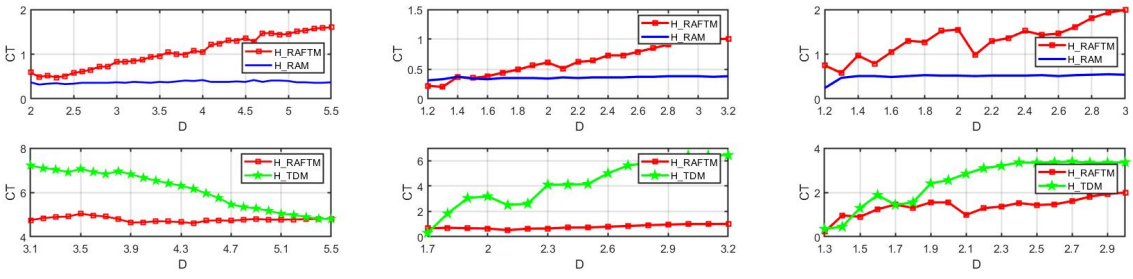


(a) $FFT, M = 2$ (b) $FFT, M = 4$ (c) $FFT, M = 6$

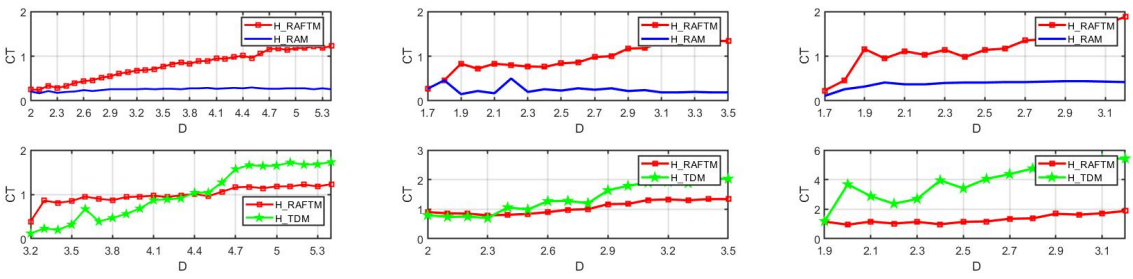


(d) $GE, M = 2$ (e) $GE, M = 4$ (f) $GE, M = 6$

Figure 4.28: Reliability improvement of heuristics (real-world DAGs) for dependent tasks under TL-DVFS scheme.

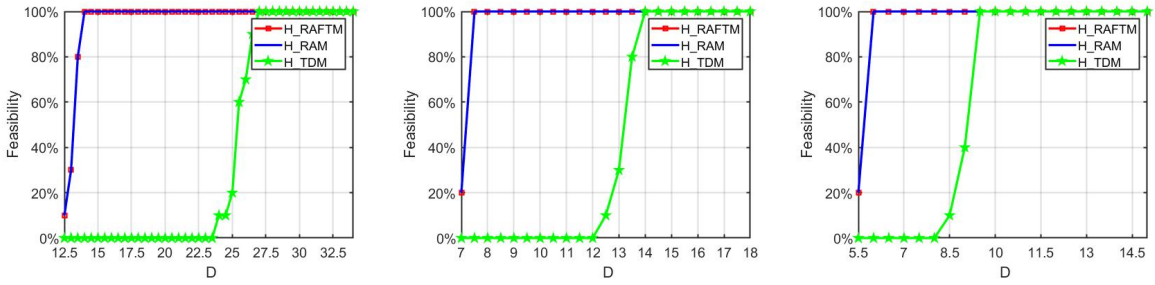


(a) $FFT, M = 2$ (b) $FFT, M = 4$ (c) $FFT, M = 6$



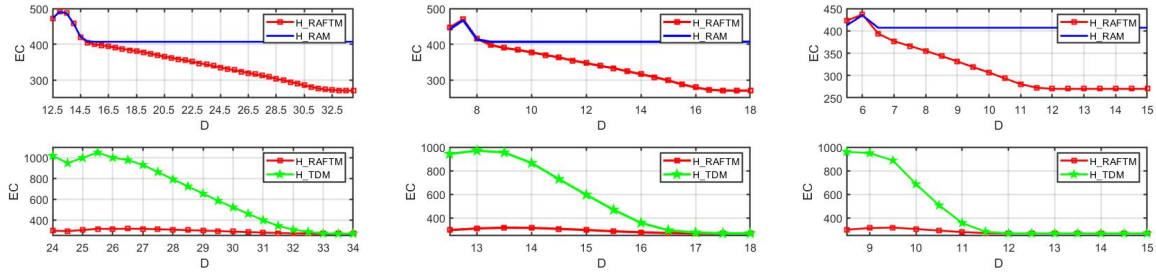
(d) $GE, M = 2$ (e) $GE, M = 4$ (f) $GE, M = 6$

Figure 4.29: Computation time (sec) of heuristics (real-world DAGs) for dependent tasks under TL-DVFS scheme.



(a) $M = 2$ (b) $M = 4$ (c) $M = 6$

Figure 4.30: Feasibility of heuristics (large randomly generated DAG, $N = 100$) for dependent tasks under TL-DVFS scheme.



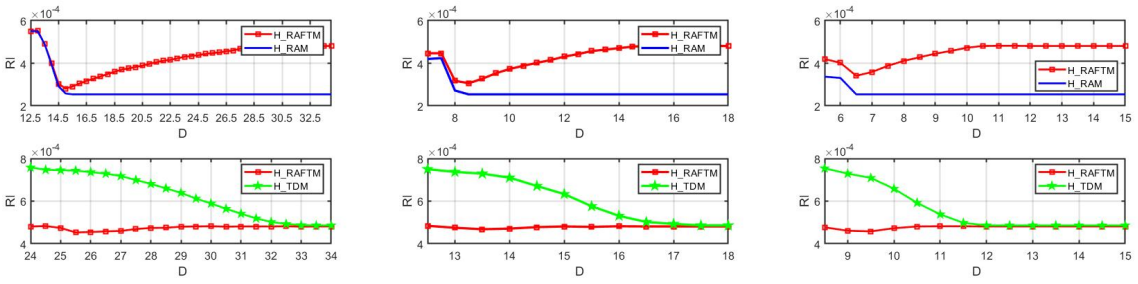
(a) $M = 2$ (b) $M = 4$ (c) $M = 6$

Figure 4.31: Energy consumption (mJ) of heuristics (large randomly generated DAG, $N = 100$) for dependent tasks under TL-DVFS scheme.

to obtain a solution. However, it provides less energy savings as explained above, especially at relaxed deadlines.

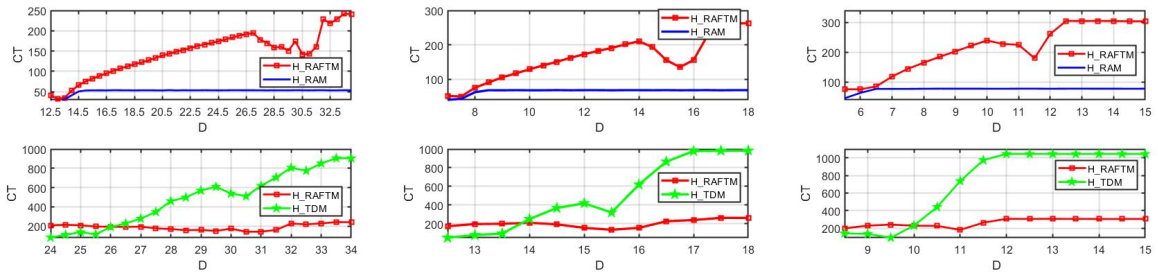
ii) Large random generated DAGs: We evaluate the quality of solutions obtained by H_RAFTM, H_RAM and H_TDM heuristics for a large randomly generated task graph with $N = 100$, $M = 2$, $M = 4$ and $M = 6$ processors, and $NE = 10$ experiments. The obtained results verify the previous observations regarding feasibility, energy consumption, reliability improvement and computation time, for the three heuristics. For instance, as the deadline is not fully relaxed or number of cores is reduced H_RAFTM has increased feasibility compared to H_TDM, i.e., 84.8% for $M = 2$, 85.7% for $M = 4$ and 83.8% for $M = 6$. Moreover, as the number of processors is increased the energy consumption is reduced as lower frequencies can be selected and partial task duplication can be applied by H_RAFTM, as long as the reliability constraint is met.

Regarding computation time, it is increased when the number of tasks increases as expected, but still remains low compared to the prohibited computation time required for the optimal approach. For a small randomly generated task with $N = 10$ (Table 4.5 when $M = 2$ and



(a) $M = 2$ (b) $M = 4$ (c) $M = 6$

Figure 4.32: Reliability improvement of heuristics (large randomly generated DAG, $N = 100$) for dependent tasks under TL-DVFS scheme.



(a) $M = 2$ (b) $M = 4$ (c) $M = 6$

Figure 4.33: Computation time (sec) of heuristics (large randomly generated DAG, $N = 100$) for dependent tasks under TL-DVFS scheme.

$M = 4$), the proposed heuristic takes less than 1 sec to find a solution for all experiments whereas for a large randomly generated task graph with $N = 100$, the average computation time (considering all experiments and deadlines) in Fig 4.33 is 144 sec ($M = 2$), 159 sec ($M = 4$) and 212 sec ($M = 6$). Comparing the computation time of the three heuristics for large DAG $N = 100$, the average computation time for H_RAM is 52 sec ($M = 2$), 66 sec ($M = 4$) and 75 sec ($M = 6$). For H_TDM, the average computation time is 488 ($M = 2$), 501 sec ($M = 4$) and 718 sec ($M = 6$). Overall, we observe that the H_TDM is the more time consuming approach, and H_RAM the least time consuming approach. However, H_TDM is not able to always find solutions, whereas H_RAFTM finds solutions with always same or less energy consumption compared to H_RAM and H_TDM.

Algorithm 9 Proposed H_RAFTM algorithm for dependent tasks under PL-DVFS scheme.

Input: Task graph (G) and set of processors (M).
Output: Application mapping (AM).

`// Phase A`
1: **for** each task τ_i in N **do**
2: $RTE_i = \{C_i^j$: C_i^j is a configuration of $\tau_i\}$;
3: $FC_i = RTE_i$ - $\{C_i^j$: $R_i < R_i^{th}\}$;
4: $BC_i = \{FC_i$: $f_i^d = 0\}$;
5: **for** each bc in BC_i **do**
6: $PC_i = FC_i$ - $\{FC_i$: $f_i^d \neq 0$ $\}$;
7: **end for**
8: $rPC_i = \{PC_i$: PC_i in increasing energy consumption$\}$;
9: **end for**
`// Phase B`
10: **for** each task τ_i in N **do**
11: Compute $rank_{\tau_i}$ (Eq. (4.6));
12: **end for**
13: $PL\text{-}T = \{N$: ordered in decreasing $rank_{\tau_i}\}$;
14: Obtain all possible frequency-to-processor groups (FTP) and put them in sum of frequency index decreasing order to get $rFTP$;
15: Start with $FTP = \{f_{L-1}, ..., f_{L-1}\}$
16: **for** each task τ_i in PL-T **do**
17: List all available configurations (AC);
18: Compute $TM_i^{SC_i}$ ($st_i = EST_i$ in Eq. (4.5));
19: **end for**
20: $AM_0 = \{TM_i^{SC_i}, i \in \boldsymbol{N}\}$;
21: Compute SL_{AM_0} (Eq. (4.4));
22: **if** $SL_{AM_0} > D$ **then**
23: Infeasible problem, algorithm stops.
24: **else if** $SL_{AM_0} = D$ **then**
25: $AM = AM_0$, algorithm stops.
26: **else if** $SL_{AM_0} < D$ **then**
27: AM relaxation (Algorithm 10);
28: **end if**

4.5 Dependent Tasks under Processor Level DVFS

4.5.1 Reliability-aware Fault-tolerant Task Mapping heuristic

Algorithm 9 depicts the proposed H_RAFTM under PL-DVFS scheme. The definition of variables used in Section 4.2.1 and Section 4.4.1 are valid also for this part.

Phase A: Task configurations, under reliability constraint.

`Phase A` (L. 1-9) is applied per task as in Phase A described in Section 4.2.1 under PL-DVFS scheme.

Phase B: Application mapping, under precedence and real-time constraints.

`Phase B` uses `Phase A` task configurations and performs the application mapping, subject to the same precedence and real-time constraints in Equations (4.3), (4.4), (4.5). Similarly, tasks start execution as soon as possible, i.e., $st_i = EST_i$, $i \in \boldsymbol{N}$ as well. We explore the frequency-to-processor (FTP) combinations to do relaxation if available time slack exists, similarly as in Section 4.2.1. `Phase B` consists of three steps (L. 10-28):

Step 1 (L. 10-14): The Priority List of tasks (PL-T) is obtained similar as in Section 4.4.1 (L. 13). All possible Frequency-to-processor (FTP) assignment groups are listed and ordered in decreasing sum of frequency index in order to obtain the ranked rFTC space, same as in Section 4.2.1 (L. 14).

Step 2 (L. 15-25): Similarly, the initial application mapping AM_0 is generated to check if the problem is feasible and time slack is available where all processors are assigned with the highest frequency f_{L-1} (L. 15). For each FTP group, we list all available configurations (AC) for each task from the PC space (L. 17) and decide task mapping for each task (including original and potential duplicated tasks) as in Section 4.2.1, except that we also consider task dependencies in Equation (4.5) when selecting the processors to execute the task. After initial task mapping AM_0 and its schedule length SL_{AM_0} are obtained (L. 20-21), we check if relaxation is possible.

Step 3 (L. 26-28): If available time slack exists (L. 26), Algorithm 10 is applied to relax the frequency assignment for energy savings by exploring different FTP groups iteratively as in Section 4 but taking into account task dependencies during task mapping (L. 6 in Algorithm 10). The algorithm ends based on the same condition as in Section 8.

Algorithm 10 Mapping Relaxation Algorithm for dependent tasks under PL-DVFS scheme.

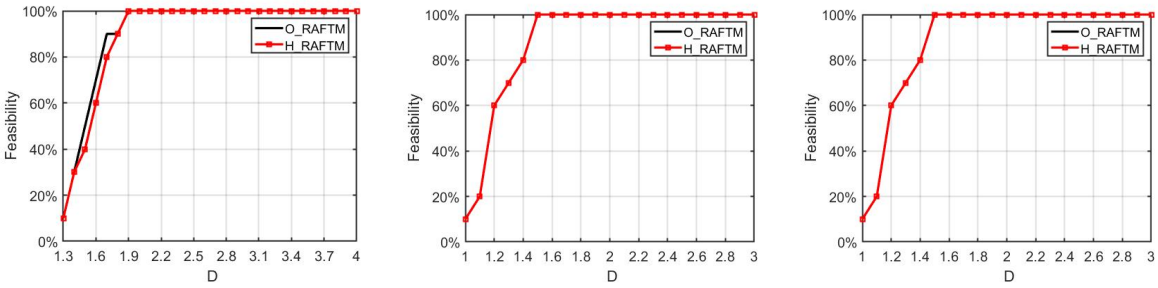
1: $AM = AM_0$, $SL = SL_{AM_0}$;
   `// iteration of all available FTP groups:`
2: **while** $SL < D$ and $|rFTC| > 1$ **do**
3: **for** each FTP in $rFTP$ **do**
4: **for** every task τ_i in $PL\text{-}T$ **do**
5: List all available configurations (AC)
6: Compute $TM_i^{SC_i}$ ($st_i = EST_i$ in Eq. (4.5))
7: **end for**
8: $AM = \{TM_i^{SC_i}, i \in \boldsymbol{N}\}$;
9: Compute SL (Eq. (4.4));
10: **end for**
11: remove FTP from $rFTP$
12: **end while**

4.5.2 Evaluation Results

The experimental set-up for mapping dependent tasks under PL-DVFS is the same as TL-DVFS scheme, presented in Section 4.4.2.

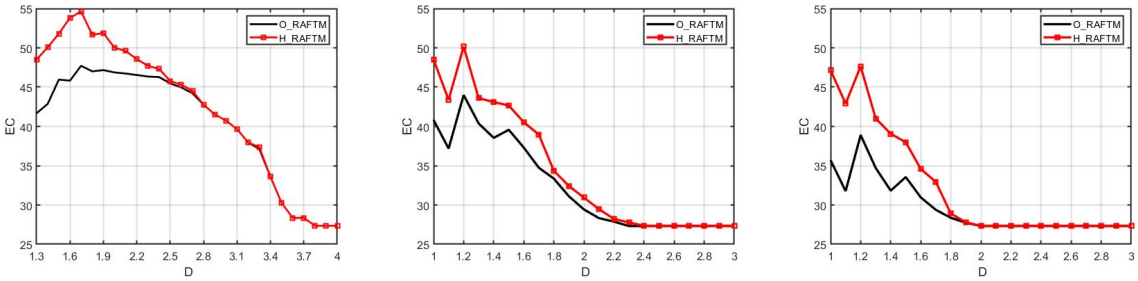
Comparison with optimal approach

Regarding feasibility when we compare the heuristic and optimal approaches under PL-DVFS, simialr observations are obtained to TL-DVFS scheme. The feasibility stays same among different DVFS schemes also for dependent tasks, since the initial application mapping is based



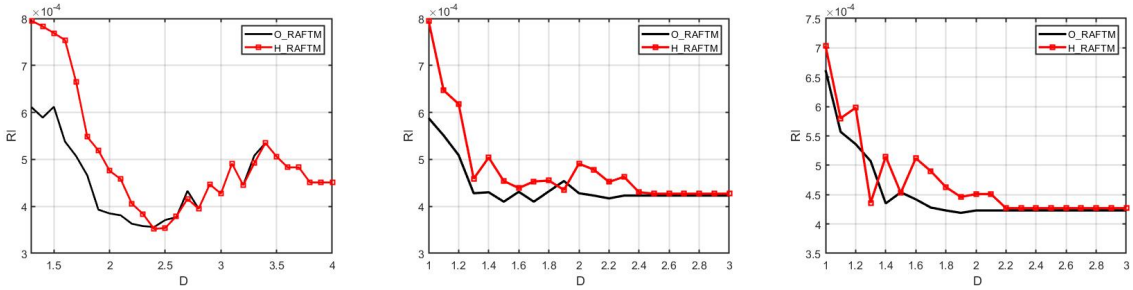
(a) $N = 10, M = 2$ (b) $N = 10, M = 4$ (c) $N = 10, M = 6$

Figure 4.34: Feasibility of optimal and heuristic approaches for dependent tasks under PL-DVFS scheme.



(a) $N = 10, M = 2$ (b) $N = 10, M = 4$ (c) $N = 10, M = 6$

Figure 4.35: Energy consumption (mJ) of optimal and heuristic approaches for dependent tasks under PL-DVFS scheme.



(a) $N = 10, M = 2$ (b) $N = 10, M = 4$ (c) $N = 10, M = 6$

Figure 4.36: Reliability improvement of optimal and heuristic approaches for dependent tasks under PL-DVFS scheme.

Table 4.6: Computation time (sec) of optimal and heuristic approaches for dependent tasks under PL-DVFS scheme.

$N = 10$, $M = 2$									
D	**1.3**	**1.4**	**1.5**	**1.6**	**1.7**	**1.8**	**1.9**	**2.0**	**2.1**
O_RAFTM	424.84	523.42	526.67	259.41	416.04	739.68	832.69	1645.61	2348.98
H_RAFTM	~ 0.02								
D	**2.2**	**2.3**	**2.4**	**2.5**	**2.6**	**2.7**	**2.8**	**2.9**	**3.0**
O_RAFTM	3358.08	5368.34	4543.89	9335.59	11974.15	13038.35	19364.46	27312.94	19378.63
H_RAFTM	~ 0.02								
D	**3.1**	**3.2**	**3.3**	**3.4**	**3.5**	**3.6**	**3.7**	**3.8**	**3.9-4.0**
O_RAFTM	13155.91	21493.13	24228.42	77477.48	5471.98	4868.59	4096.25	1.82	$\sim$2.8
H_RAFTM	~ 0.02								

$N = 10$, $M = 4$							
D	**1.0**	**1.1**	**1.2**	**1.3**	**1.4**	**1.5**	**1.6**
O_RAFTM	228.63	662.25	529.92	1092.20	1814.96	4665.41	15722.03
H_RAFTM	~ 0.17		~ 0.18				
D	**1.7**	**1.8**	**1.9**	**2.0**	**2.1**	**2.2**	**2.3**
O_RAFTM	4675.72	33294.03	70465.02	24238.97	24054.44	24072.35	154.44
H_RAFTM	~ 0.18						
D	**2.4**	**2.5**	**2.6**	**2.7**	**2.8**	**2.9**	**3.0**
O_RAFTM	137.59	114.27	118.70	111.94	116.85	125.94	93.17
H_RAFTM	~ 0.18						

$N = 10$, $M = 6$							
D	**1.0**	**1.1**	**1.2**	**1.3**	**1.4**	**1.5**	**1.6**
O_RAFTM	114.19	697.34	730.76	661.47	859.47	824.04	1090.08
H_RAFTM	0.89	0.87	0.91	0.92	0.92	0.94	0.94
D	**1.7**	**1.8**	**1.9**	**2**	**2.1**	**2.2**	**2.3**
O_RAFTM	503.30	569.25	188.64	95.91	82.54	75.01	36.54
H_RAFTM	0.94	0.93	0.93	0.93	0.94	0.94	0.94
D	**2.4**	**2.5**	**2.6**	**2.7**	**2.8**	**2.9**	**3**
O_RAFTM	40.27	32.15	18.56	13.26	5.26	3.21	2.80
H_RAFTM	0.94	0.93	0.93	0.93	0.93	0.93	0.93

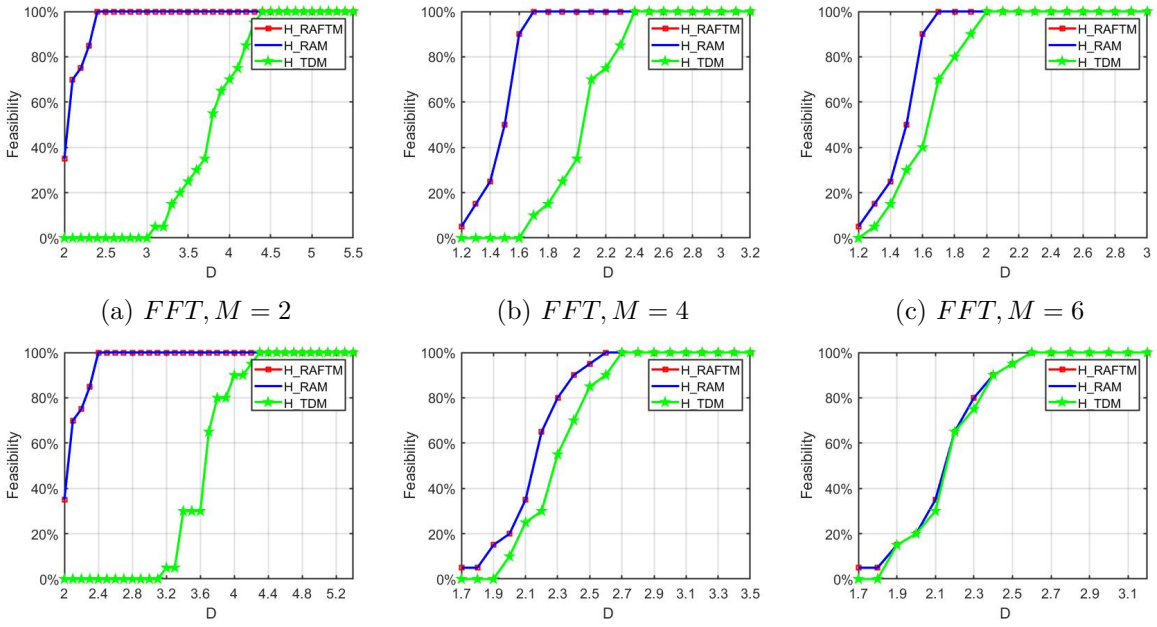
on the highest frequency.

Regarding energy consumption in Fig. 4.35, similar trends can be found as TL-DVFS, H_RAFTM generally consumes slightly more energy than O_RAFTM under PL-DVFS when deadlines are not relaxed enough while the difference of performance between O_RAFTM and H_RAFTM under PL-DVFS is slightly bigger than TL-DVFS. More details are that H_RAFTM consumes on average 4.4% ($M = 2$), 5.6% ($M = 4$) and 8.1% ($M = 6$) more energy than the optimal solutions under PL-DVFS.

Regarding reliability improvement, we observe similar trends, i.e., the H_RAFTM provides more reliability improvements than optimal solutions when deadlines are strict, as depicted in Fig. 4.36.

The average computation time of O_RAFTM and H_RAFTM is shown in Table 4.6. The computation complexity is largely reduced when the heuristic is used to solve the problem.

Overall, similar are the conclusions for dependent tasks under PL-DVFS, i.e., H_RAFTM provides near-optimal solutions with largely reduced computation complexity compared to optimal approach.



(a) $FFT, M = 2$ (b) $FFT, M = 4$ (c) $FFT, M = 6$

(d) $GE, M = 2$ (e) $GE, M = 4$ (f) $GE, M = 6$

Figure 4.37: Feasibility of heuristics (real-code DAGs) for dependent tasks under PL-DVFS scheme.

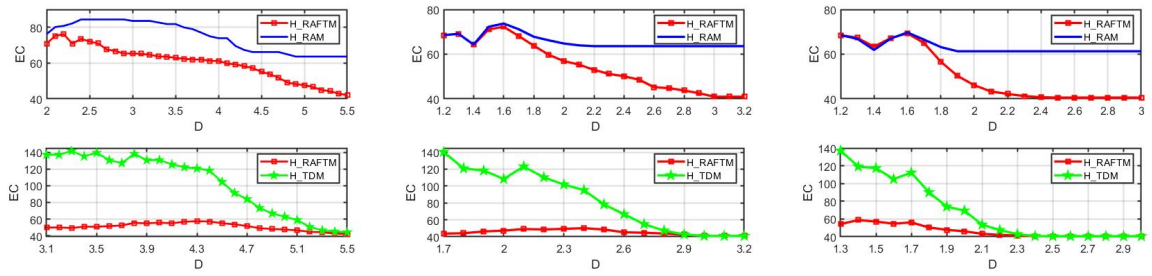
Comparison with other heuristics

i) Real-code DAGs:

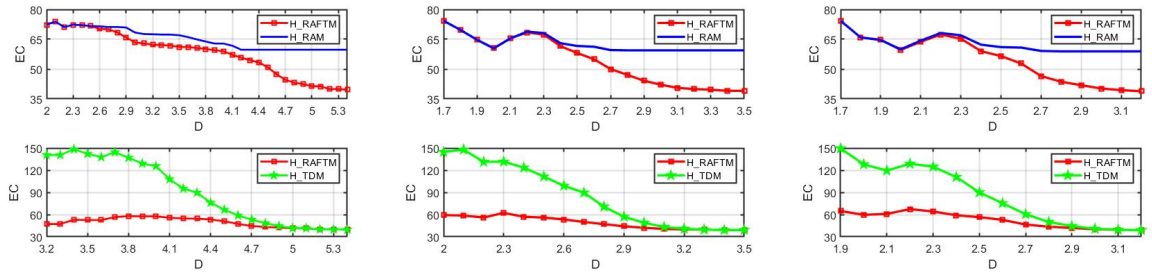
The feasibility under PL-DVFS of the heuristics is depicted in Fig. 4.37. Overall, the observations for PL-DVFS scheme are similar to TL-DVFS scheme.

The energy consumption obtained the three heuristics for FFT and GE is depicted in Fig. 4.38. Comparing the proposed heuristic with H_RAM and H_TDM, we observe similar trends regarding the energy consumption as in TL-DVFS scheme in Section 4.4.2. Comparing PL-DVFS with TL-DVFS, slightly more energy is consumed at strict deadlines for all the three heuristics. For example, for the FFT graph with $M = 4$, under TL-DVFS in Fig. 4.27c, the energy consumption of H_RAFTM, H_RAM and H_RAFTM, H_TDM at the first deadline is 54.7 mJ, 55.6 mJ and 46.5 mJ, 100.6 mJ, respectively, while under PL-DVFS in Fig. 4.38c the corresponding values are 68.5 mJ, 68.5 mJ, and 54.3 mJ, 137.0 mJ, respectively. This is explained based on the fact that PL-DVFS has less flexibility in frequency assignment compared to TL-DVFS, and thus, a lower capability to achieve energy savings.

The reliability improvement obtained by the three heuristics is depicted in Fig. 4.39. H_RAFTM generally achieves higher reliability than RAM, except in a few deadlines. Compared to H_TDM, H_RAFTM provides lower reliability for tight deadlines, as it duplicates only a part of the task-set. The same reliability improvement can be achieved in relaxed deadlines, when both

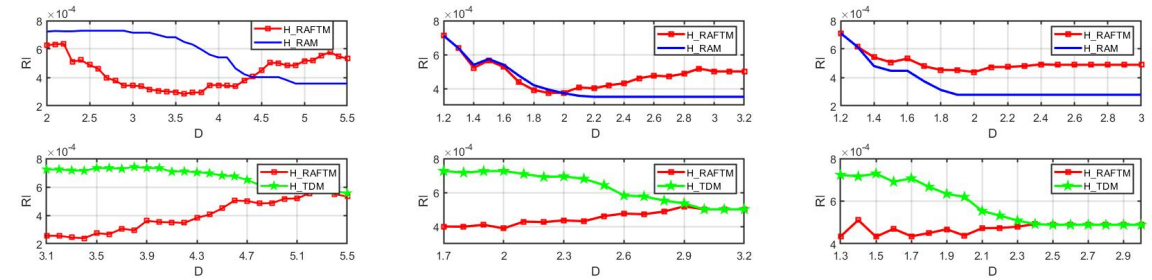


(a) $FFT, M = 2$ (b) $FFT, M = 4$ (c) $FFT, M = 6$

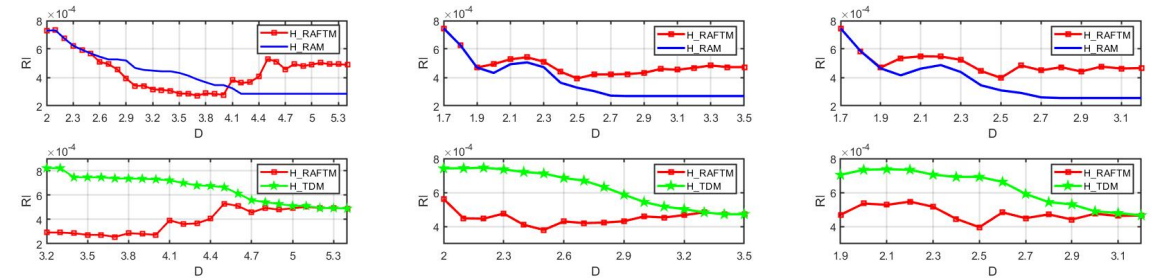


(d) $GE, M = 2$ (e) $GE, M = 4$ (f) $GE, M = 6$

Figure 4.38: Energy consumption (mJ) of heuristics (real-code DAGs) for dependent tasks under PL-DVFS scheme.

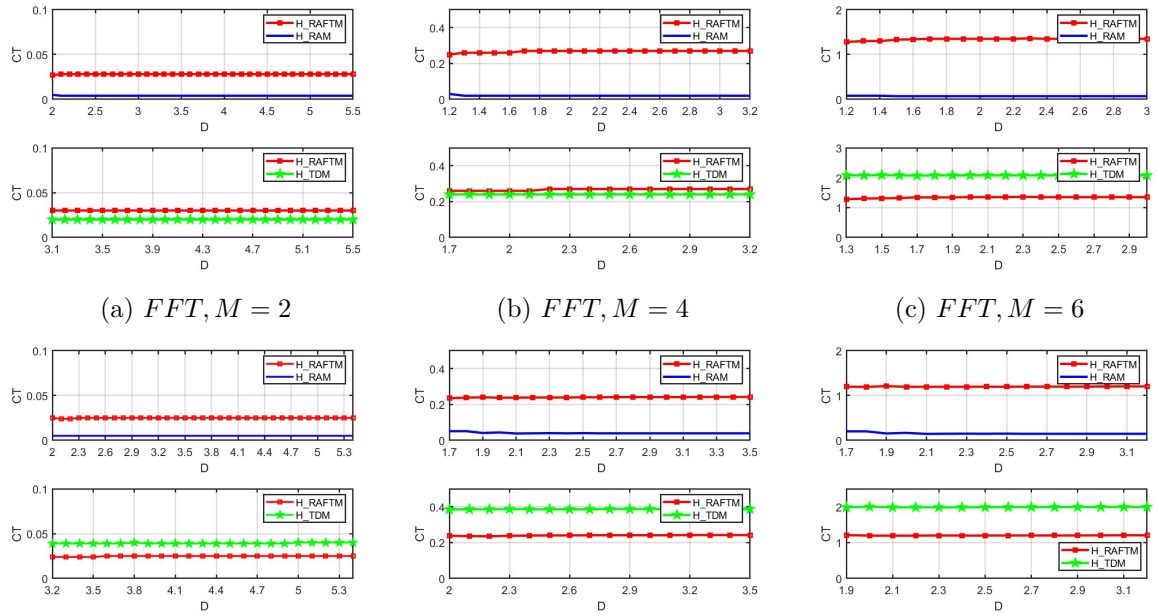


(a) $FFT, M = 2$ (b) $FFT, M = 4$ (c) $FFT, M = 6$



(d) $GE, M = 2$ (e) $GE, M = 4$ (f) $GE, M = 6$

Figure 4.39: Reliability improvement of heuristics (real-code DAGs) for dependent tasks under PL-DVFS scheme.



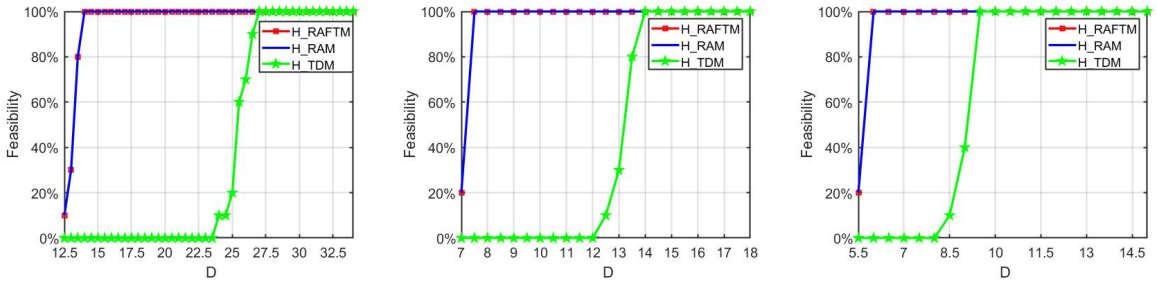
(a) $FFT, M = 2$ (b) $FFT, M = 4$ (c) $FFT, M = 6$

(d) $GE, M = 2$ (e) $GE, M = 4$ (f) $GE, M = 6$

Figure 4.40: Computation time (sec)of heuristics (real-code DAGs) for dependent tasks under PL-DVFS scheme.

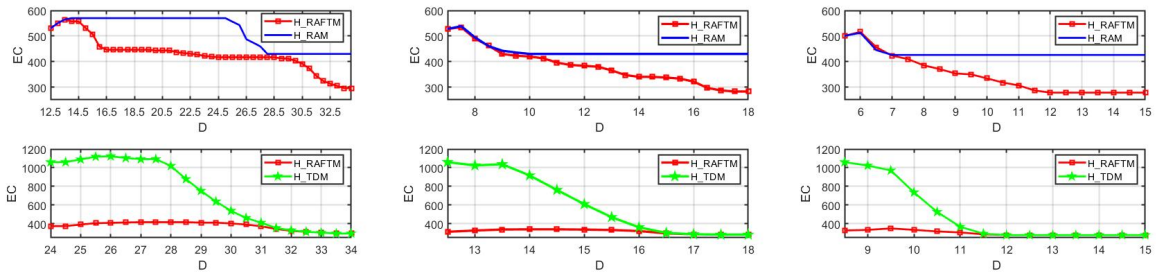
H_RAFTM and H_TDM duplicate tasks in a similar way.

The computation time of H_RAFTM, H_RAM and H_TDM heuristics is depicted in Fig. 4.40. Note that, in general, similar to Section 4.2.2 under PL-DVFS, H_TDM is the most expensive approach in terms of computation time. For H_RAM, as it only executes the original tasks, it has a reduced number of PCs in the PC space, taking the least time to obtain a solution. All experiments for these two real-code graphs take within 0.04 seconds or approximate 2 seconds when $M = 6$ to obtain solutions.



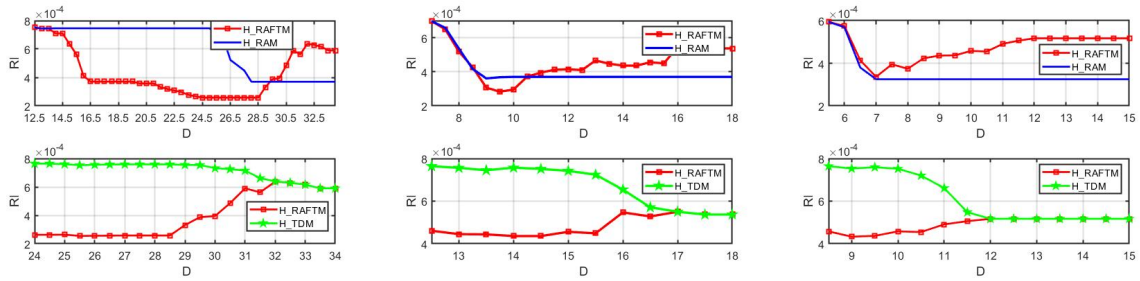
(a) $M = 2$ (b) $M = 4$ (c) $M = 6$

Figure 4.41: Feasibility of heuristics (large randomly generated DAG, $N = 100$) for dependent tasks under PL-DVFS scheme.



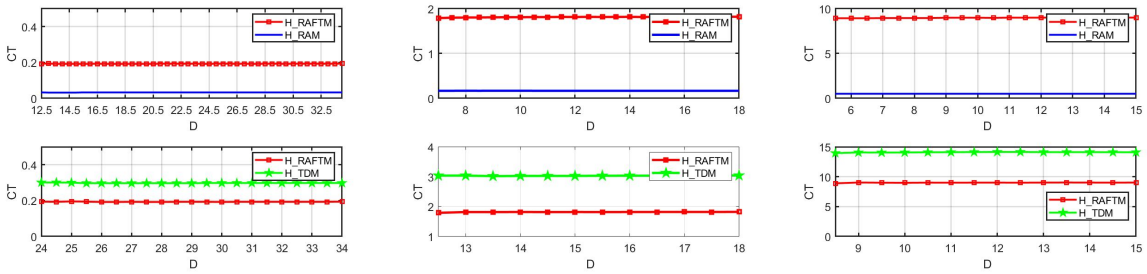
(a) $M = 2$ (b) $M = 4$ (c) $M = 6$

Figure 4.42: Energy consumption (mJ) of heuristics (large randomly generated DAG, $N = 100$) for dependent tasks under PL-DVFS scheme.



(a) $M = 2$ (b) $M = 4$ (c) $M = 6$

Figure 4.43: Reliability improvement of heuristics (large randomly generated DAG, $N = 100$) for dependent tasks under PL-DVFS scheme.



(a) $M = 2$ (b) $M = 4$ (c) $M = 6$

Figure 4.44: Computation time (sec) of heuristics (large randomly generated DAG, $N = 100$) for dependent tasks under PL-DVFS scheme.

ii) Large random generated DAGs: We compare the quality of three heuristics for a large randomly generated task graph with $N = 100$, $M = 2$, $M = 4$ and $M = 6$ processors, and $NE = 10$ experiments. Previous observations regarding feasibility, energy consumption, reliability improvement and computation time for the three heuristics are also verified by the experiments with the large randomly generated DAG.

Regarding computation time, it is increased when the number of processors and tasks increases as expected, but still remains low compared to the prohibited computation time required for the optimal approach. For a small randomly generated task with $N = 10$ (Table 4.6 when $M = 2$, $M = 4$ and $M = 6$), the proposed heuristic takes less than 0.2 seconds to find a solution at strict deadlines, whereas for a large randomly generated task with $N = 100$, the average computation time (considering all experiments and deadlines) is ~ 0.2 seconds ($M = 2$), ~ 1.8 sec ($M = 4$) and ~ 9 seconds ($M = 6$). Comparing the computation time of the three heuristics for large DAGs $N = 100$, the average computation time for H_RAM is ~ 0.03 ($M = 2$), ~ 0.17 ($M = 4$) and ~ 0.51 seconds ($M = 6$). For H_TDM, the average computation time is ~ 0.3 ($M = 2$), ~ 3 ($M = 4$) and ~ 14 seconds ($M = 6$). Overall, we observe that the H_TDM is the more time consuming approach, and H_RAM the least time consuming approach. However, H_TDM is not able to always find solutions, whereas H_RAFTM finds solutions with always same or less energy consumption compared to H_RAM and H_TDM.

4.6 Dependent Tasks under System Level DVFS

4.6.1 Reliability-aware Fault-tolerant Task Mapping heuristic

The proposed heuristic under SL-DVFS is depicted in Algorithm 11.

Phase A: Task configurations, under reliability constraint.

`Phase A` (L. 1-9) is applied per task in the same way as Phase A described in Section 4.3.1.

Phase B: Application mapping, under precedence and real-time constraints.

`Phase B` uses `Phase A` task configurations and performs the application mapping, subject to the same precedence and real-time constraints in Equations (4.3), (4.4), (4.5). Similarly, we set the tasks to start execution as soon as possible, i.e., $st_i = EST_i$, $i \in \boldsymbol{N}$ and explore all the possible frequency-to-system (FTS) combinations to do relaxation, as in Section 4.3.1. `Phase B` consists of three steps (L. 10-28):

Step 1 (L. 10-14): The Priority List of tasks (PL-T)(L. 13) and all possible ranked rFTS space are obtained as for TL-DVFS scheme (Section 4.4.1).

Algorithm 11 Proposed H_RAFTM algorithm for dependent tasks under SL-DVFS scheme.

Input: Task graph (G) and set of processors (M).
Output: Application mapping (AM).

```
    // Phase A
```
1: **for** each task τ_i in $\boldsymbol{N}$ **do**
2: $RTE_i = \{C_i^j$: C_i^j is a configuration of $\tau_i\}$;
3: $FC_i = RTE_i$ - $\{C_i^j$: $R_i < R_i^{th}\}$;
4: $BC_i = \{FC_i$: $f_i^d = 0\}$;
5: **for** each bc in BC_i **do**
6: $PC_i = FC_i$ - $\{FC_i$: $f_i^d \neq 0$ $\}$;
7: **end for**
8: $rPC_i = \{PC_i$: PC_i in increasing energy consumption$\}$;
9: **end for**

```
    // Phase B
```
10: **for** each task τ_i in $\boldsymbol{N}$ **do**
11: Compute $rank_{\tau_i}$ (Eq. (4.6));
12: **end for**
13: $PL\text{-}T = \{N$: ordered in decreasing $rank_{\tau_i}\}$;
14: Obtain all possible frequency-to-system groups (FTS) and put them in sum of frequency index decreasing order to get $rFTS$;
15: Start with the system run in highest frequency, i.e., $FTS = \{f_{L-1}, \ldots, f_{L-1}\}$
16: **for** each task τ_i in PL-T **do**
17: List all available configurations (AC);
18: Compute $TM_i^{SC_i}$ ($st_i = EST_i$ in Eq. (4.5));
19: **end for**
20: $AM_0 = \{TM_i^{SC_i}, i \in \boldsymbol{N}\}$;
21: Compute SL_{AM_0} (Eq. (4.4));
22: **if** $SL_{AM_0} > D$ **then**
23: Infeasible problem, algorithm stops.
24: **else if** $SL_{AM_0} = D$ **then**
25: $AM = AM_0$, algorithm stops.
26: **else if** $SL_{AM_0} < D$ **then**
27: AM relaxation (Algorithm 12);
28: **end if**

Step 2 (L. 15-25): After the initial task mapping AM_0 and its schedule length SL_{AM_0} are obtained (L. 20-21), we checked whether the relaxation is possible.

Step 3: (L. 26-28) If available time slack exists, Algorithm 12 is applied to explore other FTS frequency assignments to save energy, similar to the heuristic for the independent tasks (Algorithm 6) except that task dependencies are taken into account when task mapping ie performed.

4.6.2 Evaluation results

The experimental set-up for mapping dependent tasks under SL-DVFS is the same as TL-DVFS scheme, presented in Section 4.4.2.

Algorithm 12 Mapping Relaxation Algorithm for dependent tasks under SL-DVFS scheme.

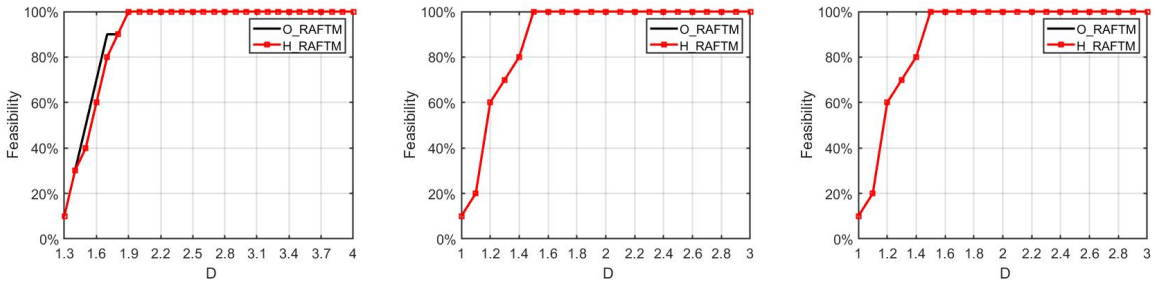
$AM = AM_0$, $SL = SL_{AM0}$;
// iteration of all available FTS groups:
while $SL < D$ and $|rFTS| > 1$ **do**
  for each FTS in $rFTC$ **do**
    for every task τ_i in $PL\text{-}T$ **do**
      Compute $TM_i^{SC_i}$ ($st_i = EST_i$ in Eq. (4.5))
    end for
    $AM = \{TM_i^{SC_i}, i \in \boldsymbol{N}\}$;
    Compute SL (Eq. (4.4));
  end for
  remove FTS from $rFTS$
end while

Comparison with optimal approach

We present the results of the proposed heuristic (H_RAFTM) and the optimal solution (O_RAFTM) for $NE = 10$ considering random graphs with $N = 10$ tasks, $M = 2$, $M = 4$ and $M = 6$ processors.

Regarding feasibility under SL-DVFS, the observations for TL-DVFS and PL-DVFS schemes hold also in this configuration.

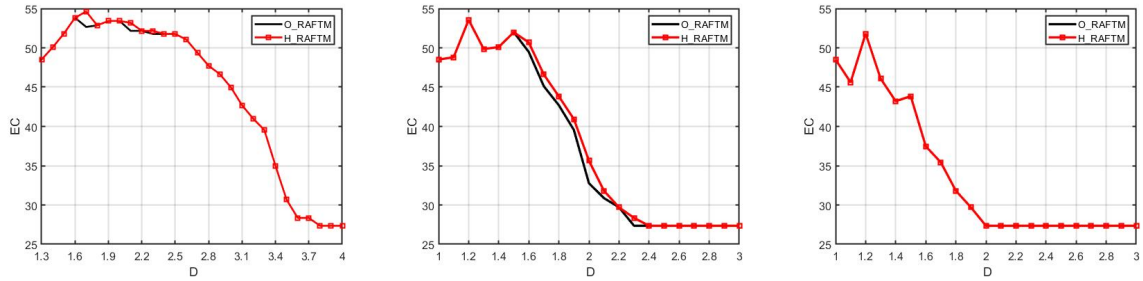
Regarding energy consumption in Fig. 4.46, H_RAFTM generally consumes slightly more energy than O_RAFTM when deadlines are strict and the same energy at relaxed deadlines. For dependent tasks, compared to TL-DVFS and PL-DVFS, the energy consumption difference between H_RAFTM and O_RAFTM is the the smallest ones under SL-DVFS. This is because SL-DVFS has the least flexibility in frequency assignment. Under SL-DVFS scheme, H_RAFTM consumes on average 0.23% ($M = 2$), 1.3% ($M = 4$) and 0% ($M = 6$) more energy than the optimal solutions.



(a) $N = 10, M = 2$ (b) $N = 10, M = 4$ (c) $N = 10, M = 6$

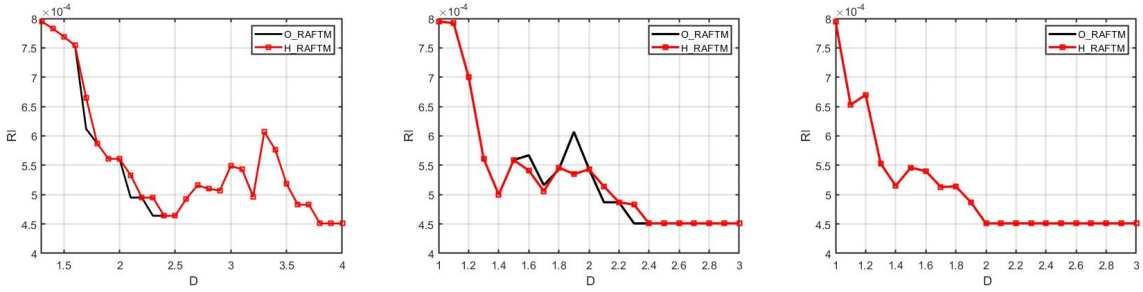
Figure 4.45: Feasibility of optimal and heuristic approaches for dependent tasks under SL-DVFS scheme.

Regarding reliability improvement, similarly, H_RAFTM provides comparative reliability improvement compared to optimal solutions depicted in Fig. 4.47.



(a) $N = 10, M = 2$ (b) $N = 10, M = 4$ (c) $N = 10, M = 6$

Figure 4.46: Energy consumption (mJ) of optimal and heuristic approaches for dependent tasks under SL-DVFS scheme.



(a) $N = 10, M = 2$ (b) $N = 10, M = 4$ (c) $N = 10, M = 6$

Figure 4.47: Reliability improvement of optimal and heuristic approaches for dependent tasks under SL-DVFS scheme.

Table 4.7: Computation time (sec) of optimal and heuristic approaches for dependent tasks ($N = 10$) under SL-DVFS scheme.

$N = 10,\ M = 2$									
D	**1.3**	**1.4**	**1.5**	**1.6**	**1.7**	**1.8**	**1.9**	**2.0**	**2.1**
O_RAFTM	3.92	3.71	2.70	4.89	8.91	8.25	111.49	209.70	864.68
H_RAFTM	~ 0.005								
D	**2.2**	**2.3**	**2.4**	**2.5**	**2.6**	**2.7**	**2.8**	**2.9**	**3.0**
O_RAFTM	1970.62	2216.28	1762.50	5956.50	3431.95	2781.67	0.22	0.20	0.28
H_RAFTM	~ 0.005								
D	**3.1**	**3.2**	**3.3**	**3.4**	**3.5**	**3.6**	**3.7**	**3.8**	**3.9-4.0**
O_RAFTM	0.20	0.12	0.14	0.17	0.18	0.23	0.16	0.20	∼0.18
H_RAFTM	~ 0.005								

$N = 10,\ M = 4$							
D	**1.0**	**1.1**	**1.2**	**1.3**	**1.4**	**1.5**	**1.6**
O_RAFTM	0.53	0.30	5.76	14.14	21.69	98.22	91.88
H_RAFTM	~ 0.005						
D	**1.7**	**1.8**	**1.9**	**2.0**	**2.1**	**2.2**	**2.3**
O_RAFTM	183.39	1301.28	585.03	65.29	28.56	15.08	9.00
H_RAFTM	~ 0.005						
D	**2.4**	**2.5**	**2.6**	**2.7**	**2.8**	**2.9**	**3.0**
O_RAFTM	9.71	6.24	5.79	1.82	2.20	0.63	0.50
H_RAFTM	~ 0.005						

$N = 10,\ M = 6$							
D	**1.0**	**1.1**	**1.2**	**1.3**	**1.4**	**1.5**	**1.6**
O_RAFTM	0.36	0.33	1.35	12.09	12.60	23.52	19.20
H_RAFTM	0.005	~ 0.006					
D	**1.7**	**1.8**	**1.9**	**2**	**2.1**	**2.2**	**2.3**
O_RAFTM	19.71	16.12	21.28	5.90	4.71	5.57	4.79
H_RAFTM	~ 0.006						
D	**2.4**	**2.5**	**2.6**	**2.7**	**2.8**	**2.9**	**3**
O_RAFTM	1.26	3.21	0.63	0.50	0.35	0.23	0.25
H_RAFTM	~ 0.006						

Table 4.7 shows the average computation time of O_RAFTM and H_RAFTM in seconds per deadline D. Similarly we observe that H_RAFTM can largely reduce the computation complexity compared to the optimal approach.

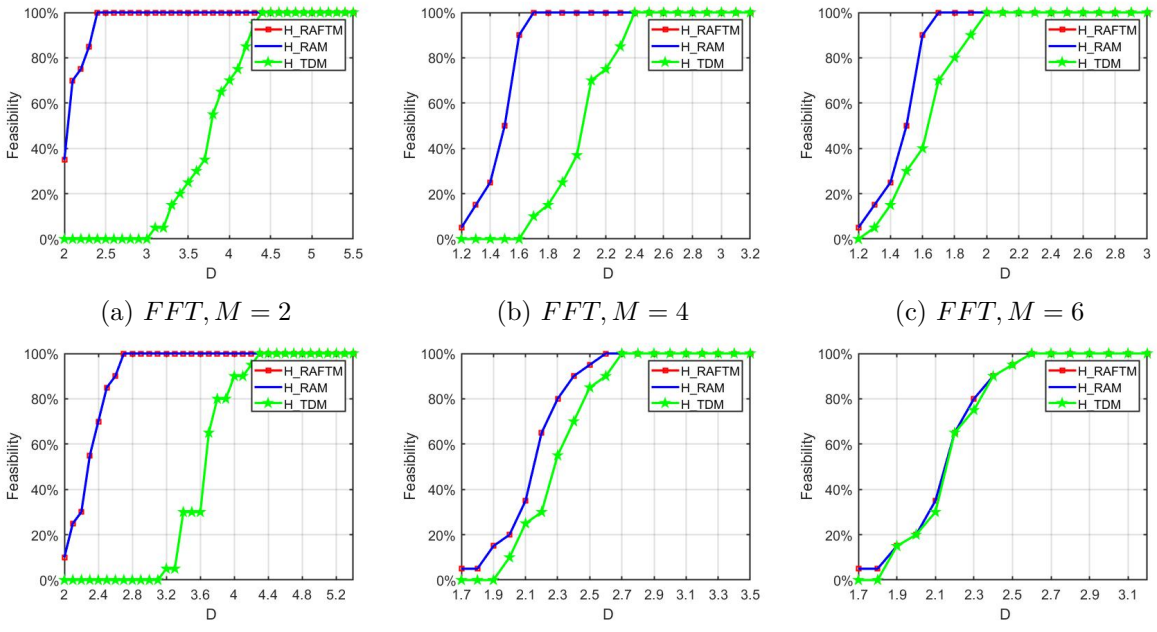
Overall, same conclusion for dependent tasks under SL-DVFS can be made that H_RAFTM provides near-optimal solutions with largely reduced computation complexity compared to optimal approach.

Comparison with other heuristics

i) Real-word DAGs: The feasibility under SL-DVFS of the three heuristics is depicted in Fig. 4.48. Same observation can be found as under TL- and PL-DVFS, we do not repeat here.

The energy consumption obtained by the solutions of the three heuristics for FFT and GE is depicted in Fig. 4.49. Similar observation can be concluded that H_RAFTM and H_RAM have similar trend of energy consumption at very strict deadlines except few cases for FFT when $M = 2$. This is caused by the least flexibility SL-DVFS has in scaling frequency and not enough processors ($M = 2$), if one task needs high frequency to mee H_RAFTM consumes significantly less energy than H_TDM at strict deadlines and similar energy when relaxed deadlines compared to H_TDM, same as explained before.

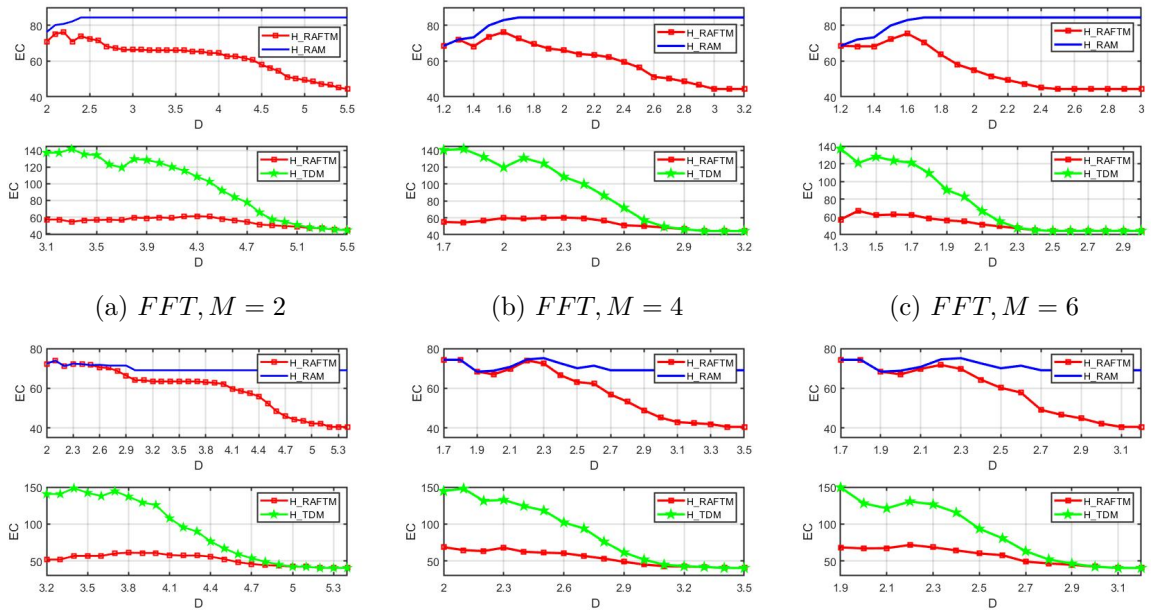
The reliability improvement obtained by the three heuristics is depicted in Fig. 4.50, leading



(a) $FFT, M = 2$ (b) $FFT, M = 4$ (c) $FFT, M = 6$

(d) $GE, M = 2$ (e) $GE, M = 4$ (f) $GE, M = 6$

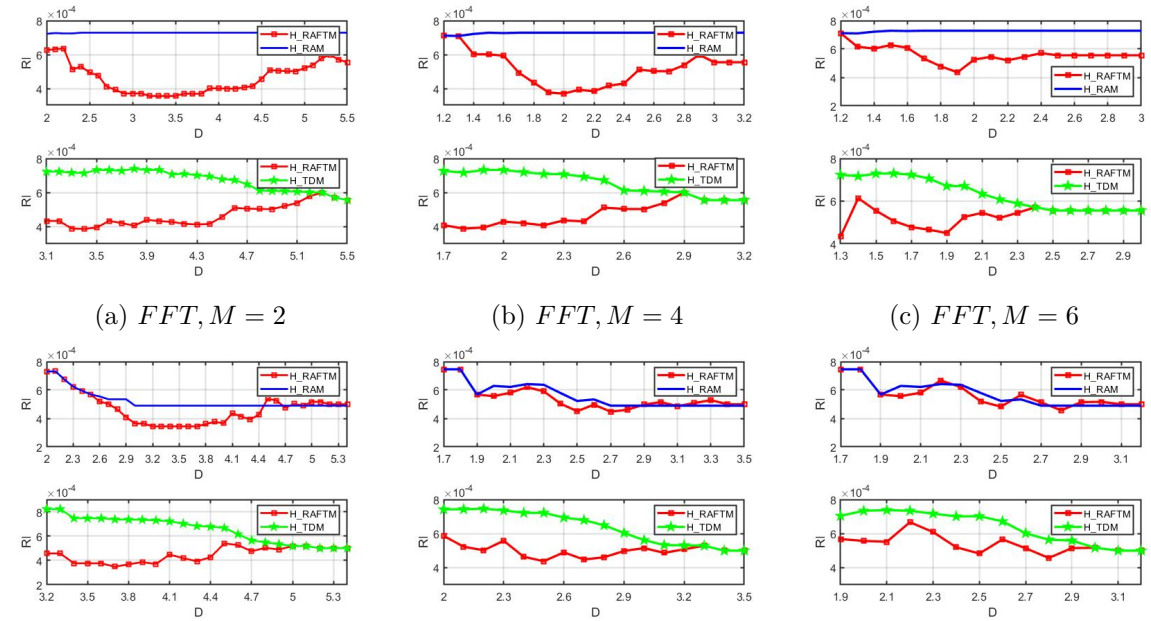
Figure 4.48: Feasibility of heuristics (real-code DAGs) for dependent tasks under SL-DVFS scheme.



(a) $FFT, M = 2$ (b) $FFT, M = 4$ (c) $FFT, M = 6$

(d) $GE, M = 2$ (e) $GE, M = 4$ (f) $GE, M = 6$

Figure 4.49: Energy consumption (mJ) of heuristics (real-code DAGs) for dependent tasks under SL-DVFS scheme.



(a) $FFT, M = 2$ (b) $FFT, M = 4$ (c) $FFT, M = 6$

(d) $GE, M = 2$ (e) $GE, M = 4$ (f) $GE, M = 6$

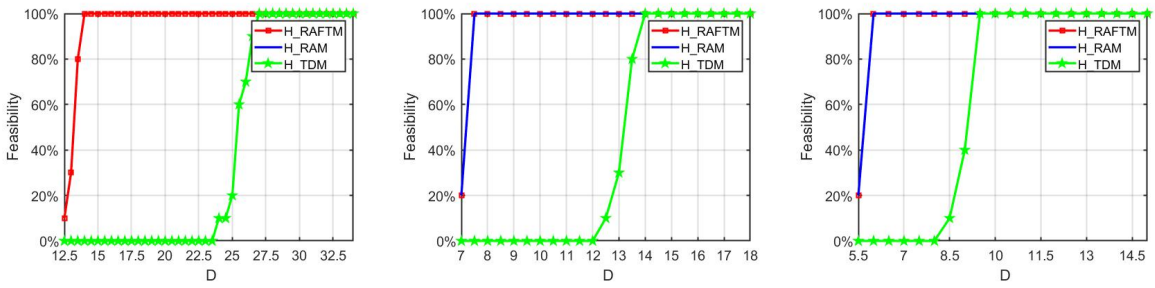
Figure 4.50: Reliability improvement of heuristics (real-code DAGs) for dependent tasks under SL-DVFS scheme.

to similar results obtained when the independent task set is considered in Section 4.3.2.

The computation time of H_RAFTM, H_RAM and H_TDM heuristics is low, within 0.02 seconds for any experiment.

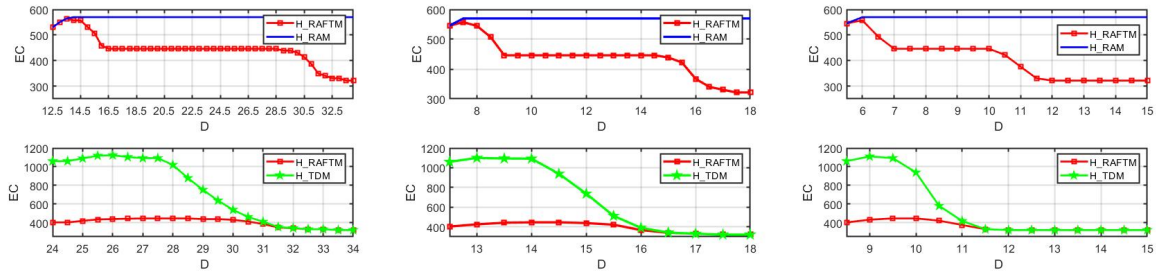
ii) Large random generated DAGs: We compare the quality of solutions obtained by H_RAFTM, H_RAM and H_TDM heuristics for a large randomly generated task graph with $N = 100$, $M = 2$, $M = 4$ and $M = 6$ processors, and $NE = 10$ experiments. Previous observations regarding feasibility, energy consumption, reliability improvement and computation time for the three heuristics are also verified by these experiments.

Comparing the computation time of the three heuristics for large DAGs, the average computation time is 0.1 seconds for all experiments which is largely reduced compared to the other two DVFS schemes. The reason is that less possible configurations (PC) are kept compared to TL-DVFS and PL-DVFS after Phase A is finished. Thus, the time to obtain application mapping is largely reduced.



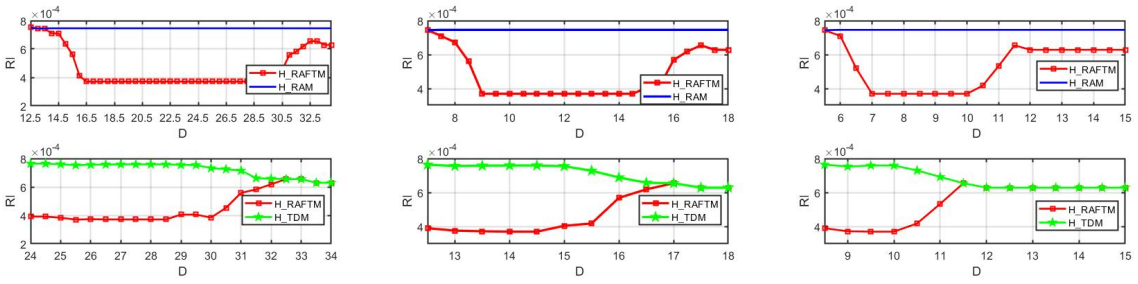
(a) $M = 2$ (b) $M = 4$ (c) $M = 6$

Figure 4.51: Feasibility of heuristics (large randomly generated DAG, $N = 100$) for dependent tasks under SL-DVFS scheme.



(a) $M = 2$ (b) $M = 4$ (c) $M = 6$

Figure 4.52: Energy consumption (mJ) of heuristics (large randomly generated DAG, $N = 100$) for dependent tasks under SL-DVFS scheme.



(a) $M = 2$ (b) $M = 4$ (c) $M = 6$

Figure 4.53: Reliability improvement of heuristics (large randomly generated DAG, $N = 100$) for dependent tasks under SL-DVFS scheme.

4.7 Conclusion

In this chapter, we have performed task mapping with partial redundancy of both independent tasks and dependent tasks on multicore platforms under three DVFS schemes. We have proposed heuristic algorithms based on list scheduling to obtain near-optimal solutions, with significantly reduced complexity.

Because the different DVFS schemes are applied in a different way regarding frequency assignment, the heuristics we propose are each time adapted to the specific DVFS scheme. The core idea for all DVFS schemes, when deciding the processor to execute a task, is based on Earliest-Start-Time (EST) first policy. Another important part in the proposed heuristics is that a relaxation algorithm is applied to explore available time slack which selects the task and configuration to do a relaxation that leads to energy saving.

Then, we provided simulation-based evaluations for our proposed heuristics. A large number of experiments are done considering with both real-code task graphs and randomly generated graphs. To evaluate the performance of the proposed heuristics, we first compare the solutions obtained by the proposed heuristics with the optimal solutions. The proposed heuristics are also compared with two SoA approaches which solve the same problem. Experimental results show that our proposed heuristics achieve closed performance on feasibility and energy consumption with optimal solutions while largely decrease the computation time. Compared to the other two SoA heuristics, and as it could have been expected from chapter 3, the proposed approach is able to provide better energy savings, and at the same time, higher feasibility even when existing approaches may fail to find a solution, without violating timing and reliability constraints.

Chapter 5

Conclusions and Perspectives

5.1 Summary

With the adventure of multicore systems and the increasing needs for high performance computing, energy consumption, reliable execution, and real-time guarantees, they have become important but conflicting concerns when designing efficient task mapping methodologies. Efficient task mapping approaches are of major interests in order to achieve low energy consumption, reliable and real-time execution at same time. Dynamic Voltage and Frequency Scaling (DVFS) technique is important to manage the optimization problems of energy-reliability-timeliness task mapping. In general, most recent works utilize DVFS technique at task level and only few approaches consider processor level. In this PhD thesis, we consider and evaluated three DVFS levels as explained in Section 3.2 . We proposed a series of task mapping methodologies that can be categorized into two groups: 1) optimal algorithms which provide the optimal solutions, and 2) heuristic-based algorithms which provide near-optimal, but much less time consuming solutions (see Fig. 5.1).

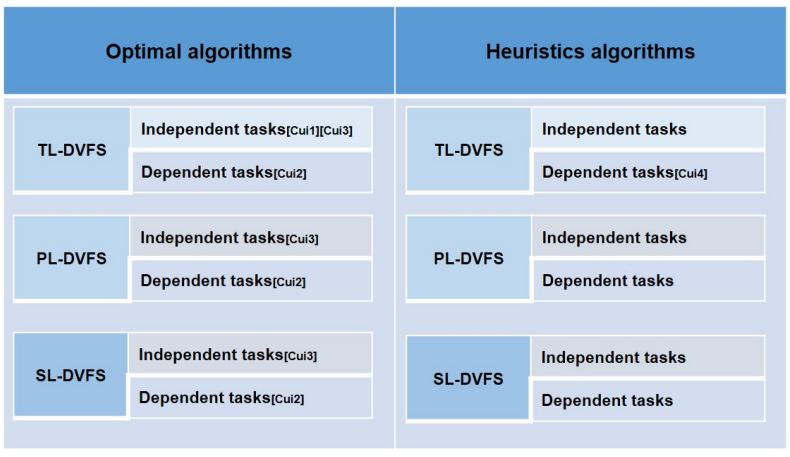


Figure 5.1: Proposed task mapping approaches under different DVFS levels and task models.

Targeting the studied problems, Fig. 5.2 depicts the general idea of the proposed Reliability-Aware Fault-Tolerant Task Mapping (RAFTM) approach based on partial duplication technique

which provides fault tolerance. Task replication is widely used as a fault tolerance technique which replicate multiple copies for each task. In the proposed approach we set the maximum number of replicas for each task to two, and select a part of the task set to do duplication. One can notice the constraints are not always exactly taken into account as it is shown in Fig 5.2 (for example reliability constraint is taken as a primary input for the heuristic-based approaches), however this figure shows the big picture of the proposed approaches.

First the optimal algorithms for independent and dependent task models under three DVFS schemes are studied in Chapter 3, by using a *variable replacement* method to safely and equivalently transfer the original MINLP problems into MILP forms, since MILP problems can be solved using optimization solvers, such as Gurobi and CPLEX tools. However, the time to obtain a solution with such optimal approaches rapidly becomes too large unless the application has very few tasks. We thus extend the proposed approach to heuristic algorithms in Chapter 4. Focusing on each DVFS schemes, we propose the corresponding heuristics for both independent and dependent task models. Finally, we conduct a large number of experiments for both randomly generated task graphs and real-world task graphs to evaluate our proposed approaches. For optimal solutions, we compare our approaches with two other SoA approaches. Experimental results show that our proposed approaches achieve better energy saving and ability to obtain feasible solutions. For heuristic-based approaches, we first compare our proposed heuristics with optimal solutions to analyze the gap of performance. Furthermore, experiments are done to evaluate the proposed heuristic algorithms against two other SoA heuristic algorithms. In conclusion, our proposed heuristic algorithms perform closed to the optimal algorithms with a large reduction of computational complexity, and outperform the SoA heuristics in energy saving and finding feasible solutions.

5.2 Future work and perspectives

Several possible extensions of this thesis can be interesting for future work.

In this thesis, we focus on homogeneous platforms where all processors share the same architecture and micro-architecture resources. Each core is identical in this system. The first extension for future work is that to replace the homogeneous platform by a heterogeneous one, for example a platform where two or more types of cores exist which differ in architecture or micro-architecture. Such an example of a simple heterogeneous multicore system is the combination of a microprocessor core with a micro-controller class core (for example, mix of Cortex-A, Cortex-M or DSP cores).

Moreover, when formulating the studied problem for dependent tasks, we first assume that the communication cost between tasks when they are executed on different processors is included into the Worst-Case-Execution-Time cost. Another assumption is the communication between

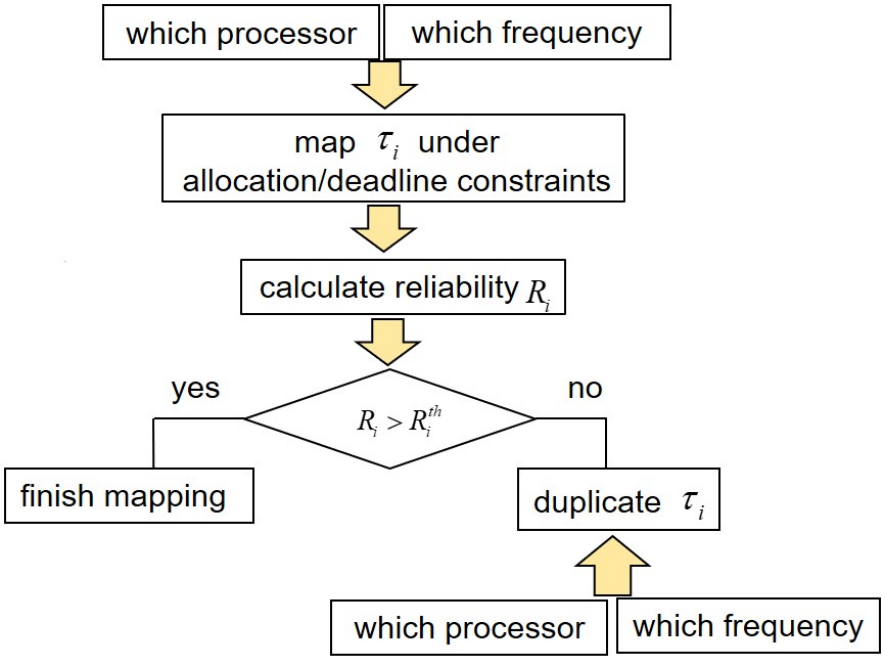


Figure 5.2: General overview of the proposed reliability-aware fault-tolerant task mapping approach

tasks are considered as reliable. By removing these two assumptions, we can take communication cost and communication reliability into consideration to get a more realistic solution and guarantee a reliable application execution.

Another interesting extension of this work is individual deadlines for each task. Actually we mainly focus on the global deadline for the task graph since we use frame-based task models in this thesis. Worst case execution cycles of each task is also considered in this thesis as we focus on design-time phase when designing task mapping methodologies, which can be too pessimistic if the worst-case execution cycles of a task is much longer than the average. It may be worthy to find a way to take into account the real execution cycles in the run-time phase to further save resource consumption.

The energy-reliability-timeliness multi-criteria are the conflicting but important concerns in modern task mapping on multicore platforms. Besides aiming at energy minimization under reliability and deadline constraints as we have studied in this thesis, one of our future work is taking reliability maximization or schedule length minimization into consideration under energy/deadline and energy/reliability constraints, and studying related task mapping approaches.

To end this thesis, fault-tolerant has been a very active domain in recent years. Although there is still a long journey for solving more practical and complicated task mapping problems with the adventure of new techniques, we hope our solutions proposed in this thesis will be helpful to people who work on such related problems.

My Publications

[1] M. Cui, L. Mo, A. Kritikakou, and E. Casseau, "Energy-aware Partial-Duplication Task Mapping under Real-Time and Reliability Constraints," in *SAMOS 2020-International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, Samos / Virtual, Greece, 2020.

[2] M. Cui, A. Kritikakou, L. Mo, and E. Casseau, ""fault-tolerant mapping of real-time parallel applications under multiple dvfs schemes," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021, pp. 387–399.

[3] ——, "Near-optimal energy-efficient partial-duplication task mapping," in *26th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2022)*, 2022.

[4] M. Cui, A. Kritikakou, L. Mo, and E. Casseau, "Energy-efficient partial-duplication task mapping under multiple dvfs schemes," *International Journal of Parallel Programming(IJPP), Springer, Published online 16 February 2022, https://doi.org/10.1007/s10766-022-00724-7*, pp. 1–28, 2022.

[5] M. Cui, A. Kritikakou, L. Mo, and E. Casseau, "Near-optimal energy-efficient partial-duplication task mapping," *Journal of System Architecture (JSA), Elsevier,* **Under Review**, 2022.

BIBLIOGRAPHY

[1] Executive summary report. Mixed-critical systems. In *European Commission Workshop on Mixed-critical systems*, 2012.

[2] A. Kritikakou, C. Pagetti, M. Roy, et al. Distributed run-time wcet controller for concurrent critical tasks in mixed-critical systems. In *22nd International Conference on Real-Time Networks and Systems*, page 139, 2014.

[3] N. Navet and F. Simonot-Lion. Fault tolerant services for safe in-car embedded systems. In *The Embedded Systems Handbook, CRC Press*, 2005.

[4] D. P. Siewiorek and P. Narasimhan. Fault-tolerant architectures for space and avionics applications. 2005.

[5] S. Girbal, M. Moretó, A. Grasset, et al. On the convergence of mainstream and mission-critical markets. In *Design Automation Conference (DAC)*, pages 1–10, 2013.

[6] F. Lemonnier, G. M. Almeida P. Millet, et al. Towards future adaptive multiprocessor soc: an innovative approach for flexible architectures. In *International Conference on Embedded Computer Systems (SAMOS)*, pages 228–235, 2012.

[7] Where does big.little fit in the world of dynamiq? In *https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/where-does-big-little-fit-in-the-world-of-dynamiq*, 2017.

[8] R. Kumar, D.M. Tullsen, and P. Ranganathan. Single-isa heterogeneous multi-core architectures for multithreaded workload performance. *IEEE/ACM Annual International Symposium on Computer Architecture*, 32(2), 2004.

[9] C. Baun. Mobile clusters of single board computers: an option for providing resources to student projects and researchers. *Springer Plus*, 5(1):article 360, 2016.

[10] T. Guan, Y. wang, L. Duan, et al. On-device mobile landmark recognition using binarized descriptor with multifeature fusion. *ACM Trans. on Intelligent Systems and Technology*, 7(1):article 12, 2015.

[11] M. Zhu and K. Shen. Energy discounted computing on multicore smartphones. In *International Conference on Advanced and Trusted Computing (ATC)*, 2016.

[12] M. Psarakis D. Gizopoulos, S. V. Adve, et al. Architectures for online error detection and recovery in multicore processors. In *Design Automation and Test Europe Conference (DATE)*, 2011.

[13] M. Pignol. Dmt and dt2 : two fault-tolerant architectures developed by cnes for cots-based spacecraft supercomputers. In *International On-Line Testing Symposium (IOLTS)*, 2006.

[14] J. Zhou, J. Sun, X. Zhou, et al. Resource management for improving soft-error and lifetime reliability of real-time MPSoCs. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 38(12):2215–2228, 2019.

[15] S. Kamdar and N. Kamdar. big. little architecture: Heterogeneous multicore processing. *International Journal of Computer Applications*, 119:35–38, 06 2015.

[16] C. Jalier, D. Lattard, A. Jerraya, et al. Heterogeneous vs homogeneous mpsoc approaches for a mobile lte modem. In *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, pages 184–189, 2010.

[17] C. Duursma, O. Olsson, and U. Sundin. Task model definition and task analysis proces. *An Advanced and Comprehensive Methodology for Integrated KBS Development*, 1998.

[18] S. Zhao, X. Dai, I. Bate, et al. Dag scheduling and analysis on multiprocessor systems: Exploitation of parallelism and dependency. In *2020 IEEE Real-Time Systems Symposium (RTSS)*, pages 128–140, 2020.

[19] M. Qamhieh. *Scheduling of Parallel Real-time DAG Tasks on Multiprocessor Systems*. PhD thesis, 2015.

[20] C. Xian, Y. H. Lu, and Z. Li. Dynamic voltage scaling for multitasking real-time systems with uncertain execution time. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 27(8):1467–1478, 2008.

[21] A. Saifullah, S. Fahmida, V. Modekurthy, et al. Cpu energy-aware parallel real-time scheduling. In *The 32nd Euromicro Conference on Real-Time Systems (ECRTS '20)*, volume 165, pages 1–24, 2020.

[22] G. C. Buttazzo. Hard real-time computing systems: Predictable scheduling algorithms and applications. In *Real-Time Systems Series, Springer*, 2011.

[23] F. Poursafaei, M. Bazzaza Morteza, M. Kafshdooz, et al. Slack clustering for scheduling frame-based tasks on multicore embedded systems. *Microelectronics Journal, Elsevier*, 81:144–153, 2015.

[24] G. Xie, Y. Chen, X. Xiao, et al. Energy-efficient fault-tolerant scheduling of reliable parallel applications on heterogeneous distributed embedded systems. *IEEE Trans. on Sustainable Computing*, 3(3):167–181, 2018.

[25] Gang Chen, Kai Huang, and Alois Knoll. Energy optimization for real-time multiprocessor system-on-chip with optimal dvfs and dpm combination. *ACM Trans. Embedded Computing Systems*, 13(3s), 2014.

[26] B. Zhao, H. Aydin, and D. Zhu. Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints. *ACM Trans. on Design Automation of Electronic Systems*, 18(2), 2013.

[27] A. Guliani and M. M. Swift. Per-application power delivery. In *The Fourteenth EuroSys Conference 2019,*.

[28] D. Hackenberg, R. Schöne, T. Ilsche, et al. An energy efficiency feature survey of the intel haswell processor. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, pages 896–904, 2015.

[29] AMD Inc., R. Schöne, T. Ilsche, et al. Processor programming reference (ppr) for amd family 17h model 01h, revision b1 processors. 2017.

[30] K. Huang, X. Jiang, X. Zhang, et al. Energy-efficient fault-tolerant mapping and scheduling on heterogeneous multiprocessor real-time systems. *IEEE Access*, 6:57614–57630, 2018.

[31] M. Salehi, A. Ejlali, and B. M. AI-Hashimi. Two-phase low-energy n-modular redundancy for hard real-time multi-core systems. *IEEE Trans. on Parallel and Distributed Systems*, 27(5):1497–1510, 2016.

[32] M. A. Haque, H. Aydin, and D. Zhu. On reliability management of energy-aware real-time systems through task replication. *IEEE Trans. on Parallel and Distributed Systems*, 28(3):813–825, 2017.

[33] S. Safari, M. Ansari, G. Ershadi, et al. On the scheduling of energy-aware fault-tolerant mixed-criticality multicore systems with service guarantee exploration. *IEEE Trans. on Parallel and Distributed Systems*, 30(10):2338–2354, 2019.

[34] D. Li and J. Wu. Energy-aware scheduling for frame-based tasks on heterogeneous multiprocessor platforms. *IEEE 2012 41st International Conference on Parallel Processing*, 2012.

[35] D. Li and J. Wu. Minimizing energy consumption for frame-based tasks on heterogeneous multiprocessor platforms. *IEEE Trans. on Parallel and Distributed Systems*, 26(3):810–823, 2015.

[36] D. Zhu, R. Melhem, and D. Mosse. The effects of energy management on reliability in real-time embedded systems. *IEEE/ACM International Conference on Computer Aided Design(ICCAD)*, pages 35–40, 2004.

[37] S. Tosun. Energy- and reliability-aware task scheduling onto heterogeneous MPSoC architectures. *Journal of Supercomputing, Springer*, 62(1), 2012.

[38] R. C. Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Trans. Device and Materials Reliability*, 5(3):305–316, 2005.

[39] E. Dubrova. Fault tolerant design: An introduction. In *Springer*, 2008.

[40] J. C. Laprie. Dependable computing and fault tolerance: Concepts and terminology. *IEEE Computer Society*, (9):2–11, 1985.

[41] Dakai Zhu. Reliability-aware dynamic energy management in dependable embedded real-time systems. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, pages 397–407, 2006.

[42] M. Ebrahimi, A. Evans, M. B. Tahoori, et al. Comprehensive analysis of alpha and neutron particle-induced soft errors in an embedded processor at nanoscales. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–6, 2014.

[43] S. Rehman. *Reliable Software for Unreliable Hardware – A Cross-Layer Approach*. PhD thesis, 2015.

[44] C. Gou, A. Benoit, M. Chen, et al. Reliability-aware energy optimization for throughput-constrained applications on MPSoC. In *IEEE 24th International Conference on Parallel and Distributed Systems*, pages 1–10, 2018.

[45] L. Han, L. C. Canon, J. Liu, et al. Improved energy-aware strategies for periodic real-time tasks under reliability constraints. In *2019 IEEE Real-Time Systems Symposium (RTSS)*, pages 17–29, 2019.

[46] Q. Zheng, B. Veeravalli, and C. K. Tham. On the design of fault-tolerant scheduling strategies using primary-backup approach for computational grids with low replication costs. *IEEE Trans. on Computers*, 58(3):380–393, 2009.

[47] A. Benoit, M. Hakem, and Y. Robert. Fault tolerant scheduling of precedence task graphs on heterogeneous platforms. In *2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8, 2008.

[48] J. Choi, B. Jung, and Y. Choi. An adaptive and integrated low-power framework for multicore mobile computing. In *J. Mobile Information Systems*, 2017.

[49] I. Ripoll and R. Ballester-Ripoll. Period selection for minimal hyperperiod in periodic task systems. *IEEE Trans. on Computers*, 62(9):144–153, 2013.

[50] K. S. Chatha and R. Vemuri. Hardware-software partitioning and pipelined scheduling of transformative applications. *IEEE Trans. Very Large Scale Integr. Syst.*, 10:193–208, 2002.

[51] Z. Deng, D. Cao, H. Shen, et al. Reliability-aware task scheduling for energy effciency on heterogeneous multiprocessor systems. *The Journal of Super computing*, 2021.

[52] D. Rossi, M. Omana, F. Toma, et al. Multiple transient faults in logic: an issue for next generation ics? *20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'05)*, 2005.

[53] J. R. Azambuja, F. Kastensmidt, and J. Becker. Hybrid fault tolerance techniques to detect transient faults in embedded processors. *Springer*, 2014.

[54] Omer Qadir. *Hardware Architecture for a Bi-directional Protein Processor Associative Memory.* PhD thesis, 11 2011.

[55] D. Zhu and H. Aydin. Energy management for real-time embedded systems with reliability requirements. In *2006 IEEE/ACM International Conference on Computer Aided Design*, pages 528–534, 2006.

[56] B. Zhao, H. Aydin, and D. Zhu. Generalized reliability-oriented energy management for real-time embedded applications. In *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 381–386, 2011.

[57] L. Leung, C. Tsui, and W. Ki. Simultaneous task allocation, scheduling and voltage assignment for multiple-processors-core systems using mixed integer nonlinear programming. In *2003 International Symposium on Conference: Circuits and Systems (ISCAS '03).*

[58] L. Leung, C. Tsui, and W. Ki. Minimizing energy consumption of multiple-processors-core systems with simultaneous task allocation, scheduling and voltage assignment. *IEEE Asia and South Pacific Design Automation Conference(ASP-DAC)*, 2004.

[59] D. Li and J. Wu. Energy-efficient contention-ware application mapping and scheduling on noc-based mpsocs. *Journal of Parallel and Distributed Computing, Elsevier*, 2016.

[60] M. Qiu and E. H. M. Sha. Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems. *ACM Trans. on Design Automation of Electronic Systems*, 14(2), 2009.

[61] M. W. Convolbo and J. Chou. Cost-aware dag scheduling algorithms for minimizing execution cost on cloud resources. *Journal of Supercomputing, Springer*, 72(3):985–1012, 2016.

[62] G. Xie, Y. Chen, Y. Liu, et al. Resource consumption cost minimization of reliable parallel applications on heterogeneous embedded systems. *IEEE Trans. on Industrial Informatics*, 13(4):1629–1640, 2017.

[63] L. Zhang, K.Li, C. Li, et al. Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems. *Information Sciences,Springer*, 379:241–256, 2017.

[64] T. Wei, P. Mishra, K. Wu, et al. Quasi-static fault-tolerant scheduling schemes for energy-efficient hard real-time systems. *Journal of Systems and Software, Elsevier*, 85(6):1386–1399, 2012.

[65] Zheng Li, Li Wang, Shuhui Li, et al. Reliability guaranteed energy-aware frame-based task set execution strategy for hard real-time systems. *Journal of Systems and Software, Elsevier*, 86(12):3060–3070, 2013.

[66] L. Zhang, K. Li, K. Li, et al. Joint optimization of energy efficiency and system reliability for precedence constrained tasks in heterogeneous systems. *Int. Journal of Electrical Power &Energy Systems*, 78:499–512, 2016.

[67] P. Pop, K. H. Poulsen, and V. Izosimovand others. Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems. In *2007 5th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 233–238, 2007.

[68] S. Hua, P. R. Pari, and G. Qu. Dual-processor design of energy efficient fault-tolerant system. In *IEEE 17th International Conference on Application-specific Systems, Architectures and Processors (ASAP'06)*, pages 239–244, 2006.

[69] R. Sridharan and R. Mahapatra. Reliability aware power management for dual-processor real-time embedded systems. In *Design Automation Conference(DAC)*, pages 819–824, 2010.

[70] Y. Gao, L. Han, J. Liu, et al. Minimizing energy consumption for real-time tasks on heterogeneous platforms under deadline and reliability constraints. Research Report RR-9403, Inria - Research Centre Grenoble – Rhône-Alpes, 2021.

[71] M. Cui, L. Mo, A. Kritikakou, and E. Casseau. Energy-aware Partial-Duplication Task Mapping under Real-Time and Reliability Constraints. In *SAMOS 2020-International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*, Samos / Virtual, Greece, 2020.

[72] M. Cui, A. Kritikakou, L. Mo, and E. Casseau. Fault-tolerant mapping of real-time parallel applications under multiple dvfs schemes. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 387–399, 2021.

[73] M. Cui, A. Kritikakou, L. Mo, and E. Casseau. Energy-efficient partial-duplication task mapping under multiple dvfs schemes. *International Journal of Parallel Programming(IJPP), Springer*, 2021.

[74] N. Rana, Muhammad Shafie Abd Latiff, and Shafi'i Muhammad Abdulhamid et al. Whale optimization algorithm: a systematic review of contemporary applications, modifications and developments. In *Neural Computing and Applications volume*, page 16245–16277, 2020.

[75] B. Zhao, H. Aydin, and D. Zhu. On maximizing reliability of real-time embedded applications under hard energy constraint. *IEEE Trans. on Industrial Informatics*, 6(3):316–328, 2010.

[76] J. J. Dongarra, E. Jeannot, E. Saule, et al. Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In *Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures, ACM*, SPAA '07, page 280–288, 2007.

[77] L. Zhang, K. Li, W. Zheng, et al. Contention-aware reliability efficient scheduling on heterogeneous computing systems. *IEEE Trans. on Sustainable Computing*, 3(3):182–194, 2018.

[78] S. Wang, K. Li, J. Mei, et al. A reliability-aware task scheduling algorithm based on replication on heterogeneous computing systems. *Journal of Grid Computing, Springer*, 15(1):23–39, 2017.

[79] K. H. Chen, G. von der Brüggen, and J. J. Chen. Reliability optimization on multi-core systems with multi-tasking and redundant multi-threading. *IEEE Trans. on Computers*, 67(4):484–497, 2018.

[80] A. Das, A. Kumar, B. Veeravalli, et al. Combined dvfs and mapping exploration for lifetime and soft-error susceptibility improvement in mpsocs. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–6, 2014.

[81] H. Topcuoglu, S. Hariri, and M. Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. on Parallel and Distributed Systems*, 13(3):260–274, 2002.

[82] H. Kanemitsu, M. Hanada, and H. Nakazato. Clustering-based task scheduling in a large number of heterogeneous processors. *IEEE Trans. on Parallel and Distributed Systems*, 27(11):3144–3157, 2016.

[83] R. Mayer, C. Mayer, and L. Laich. The tensorflow partitioning and scheduling problem: It's the critical path! In *Proceedings of the 1st Workshop on Distributed Infrastructures for Deep Learning*, DIDL '17, page 1–6. Association for Computing Machinery, 2017.

[84] Z. Quan, Z. Wang, T. Ye, et al. Task scheduling for energy consumption constrained parallel applications on heterogeneous computing systems. *IEEE Trans. on Parallel and Distributed Systems*, 31(5):1165–1182, 2020.

[85] K. He, X. Meng, Z. Pan, et al. A novel task-duplication based clustering algorithm for heterogeneous computing environments. *IEEE Trans. on Parallel and Distributed Systems*, 30(1):2–14, 2019.

[86] Z. Zong, A. Manzanares, X. Ruan, et al. Ead and pebd: Two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters. *IEEE Trans. on Computers*, 60(3):360–374, 2011.

[87] A. Abdi, A. Girault, and H. Zarandi. Erpot: A quad-criteria scheduling heuristic to optimize execution time, reliability, power consumption and temperature in multicores. *IEEE Trans. on Parallel and Distributed Systems*, 30(10):2193–2210, 2019.

[88] K. Cao, J. Zhou, P. Cong, et al. Affinity-driven modeling and scheduling for makespan optimization in heterogeneous multiprocessor systems. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 38(7):1189–1202, 2019.

[89] M. Salehi, M. K. Tavana, S. Rehman, et al. DRVS: Power-efficient reliability management through dynamic redundancy and voltage scaling under variations. In *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 225–230, 2015.

[90] G. Quan and V. Chaturvedi. Feasibility analysis for temperature-constraint hard real-time periodic tasks. *IEEE Trans. on Industrial Informatics*, 6(3):329–339, 2010.

[91] Matthew Guthaus, Jeffrey Ringenberg, Daniel Ernst, Todd Austin, Trevor Mudge, and Richard Brown. Mibench: A free, commercially representative embedded benchmark suite. In *International Workshop on Workload Characterization*, pages 3–14, 01 2002.

[92] S. Rokicki, D. Pala, J. Paturel, et al. What you simulate is what you synthesize: Designing a processor core from c++ specifications. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2019.

[93] Deverge J. F and I. Puaut. Safe measurement-based wcet estimation. *OpenAccess Series in Informatics*, 1, 01 2005.

[94] Yifeng Guo, Dakai Zhu, and Hakan Aydin. Reliability-aware power management for parallel real-time applications with precedence constraints. In *2011 International Green Computing Conference and Workshops*, pages 1–8, 2011.

école
normale
supérieure
rennes

Titre : Ordonnancement de tâches sur architectures multicoeurs avec des contraintes d'énergie, de temps réel et de tolérance aux fautes

Mot clés : architectures multicoeurs, ordonnancement temps réel, tolérance aux fautes, consommation d'énergie

Résumé : Le contexte de cette thèse est l'ordonnancement de tâches sur architectures multiprocesseurs et avec prise en compte de la tolérance aux fautes. Dans ce contexte, la technique de DVFS (Dynamic Voltage and Frequency Scaling) est généralement utilisée pour économiser l'énergie des processeurs. Malheureusement, lorsque la fréquence et/ou la tension est réduite, l'énergie diminue mais la fiabilité diminue également. A l'inverse, l'utilisation de fréquences et ou tensions plus élevées permet d'augmenter la fiabilité mais au dépend de l'augmentation de la consommation d'énergie.

Dans le cadre de cette thèse, pour minimiser la consommation d'énergie tout en respectant les contraintes de temps réel et de fiabilité, le principe retenu est de combiner la technique du DVFS pour limiter la consommation d'énergie et la réplication de certaines tâches pour satisfaire la contrainte de fiabilité.

La méthode proposée a d'abord été formalisés sous la forme d'un problème de programmation non linéaire mixte en nombre entier, problème ensuite transformé en un problème équivalent de programmation linéaire mixte en nombres entiers pour sa résolution. Afin de réduire le temps nécessaire pour trouver une solution, une technique de type heuristique est ensuite proposée. Les expérimentations montrent que les heuristiques proposées permettent d'obtenir des résultats quasi optimaux, avec un temps de calcul faible par rapport à ceux obtenus par des solveurs, et, en comparaison avec d'autres approches heuristiques de la littérature, permettent d'obtenir une consommation d'énergie plus faible tout en étant capable d'aboutir plus souvent à des solutions.

Title: Energy-Quality-Time Fault Tolerant Task Mapping on Multicore Architectures

Keywords: multicore architectures, real-time scheduling, fault tolerance, energy consumption

Abstract: The context of this thesis is the mapping of tasks on multicore architectures and taking fault tolerance into account. In this context, the technique of DVFS (Dynamic Voltage and Frequency Scaling) is generally used to save energy. Unfortunately, when frequency and/or voltage is reduced, energy decreases but reliability also decreases. Conversely, the use of higher frequencies and/or voltages increases the reliability but at the expense of increased energy consumption.

In the context of this thesis, to minimize energy consumption while respecting real-time and reliability constraints, the principle we adopted is to combine the DVFS technique to limit energy consumption and the replication of certain tasks to satisfy the reliability constraint.

The proposed method was first formalized as a mixed integer nonlinear programming problem, then transformed into an equivalent mixed integer linear programming problem for its resolution. In order to reduce the time needed to find a solution, a heuristic-based technique is then proposed. Experiments show that the proposed heuristics make it possible to obtain almost optimal results, with a low computation time compared to those obtained by solvers, and, in comparison with other heuristic-based approaches of the literature, make it possible to obtain a lower energy consumption while being able to come up with solutions more often.