Fault tolerance through self-configuration in the future nanoscale multiprocessors

Piotr Zajac

THÈSE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'**Institut National des Sciences Appliquées de Toulouse**

École Doctorale : Génie Électrique, Électronique, Télécommunications
Discipline : Conception des Circuits Microélectroniques et Microsystèmes

présentée et soutenue

par

Piotr Zając

le 30 juin 2008

Fault Tolerance through Self-configuration in the Future Nanoscale Multiprocessors

Directeurs de thèse:

M. Jacques Henri COLLET et M. Andrzej NAPIERALSKI

JURY

M. Jean-Marie DILHAC, Président
M. Christian LANDRAULT
M. Andrzej NAPIERALSKI
M. Michael NICOLAIDIS
M. Stanislaw PIESTRAK
M. Jacques Henri COLLET

Acknowledgements

This work has been carried out at the Laboratory of Analysis and Architecture of Systems of the French National Research Center (LAAS-CNRS). I wish to express my gratitude to the successive directors of the LAAS-CNRS, Mr. Malik Ghallab and Mr. Raja Chatila, for the facilities provided in the laboratory.

This thesis was conducted as a part of a cotutelle agreement between Technical University of Lodz and Institut National des Sciences Appliquées of Toulouse. It was partially financed by the Cotutelle PhD Scholarship from the French Ministry of Foreign Affairs.

I owe my sincere thanks to my thesis supervisor, Mr. Jacques Henri Collet, without whom this thesis would not be possible. I appreciate his vast knowledge and I am most grateful to him for offering me the opportunity of working in the interesting area of research, for his guidance, advice and time he devoted to read my thesis.

I am also deeply grateful to my second supervisor, Mr. Andrzej Napieralski, for his many helpful suggestions and constant encouragement during the course of this work and I want to thank him for his expertise, understanding and patience.

I wish to express my gratitude to Mr. Christian Landrault and Mr. Michael Nicolaidis for examining my thesis, to Mr. Jean-Marie Dilhac for granting me the honor of presiding over my thesis jury and to Mr. Stanislaw Piestrak for his valuable remarks concerning the manuscript.

I want to thank all members of the TSF group for accepting me in their midst and providing a friendly atmosphere during my work at LAAS-CNRS. I would especially like to acknowledge Mr. Yves Crouzet and Mr. Jean Arlat for their insightful comments on my work.

I have also benefited from many valuable discussions with my colleagues, Cezary Maj and Michal Olszacki.

Finally, I would like to thank my family: my parents, my grandparents and my sister for their love and constant support in all my endeavors.

RESUMÉ DE THÈSE

Table des matières

F1. Introduction

Depuis trente ans, la puissance du calcul des processeurs a continûment augmenté pour les raisons suivantes :

- la miniaturisation de la taille des transistors a permis une augmentation du nombre de transistors intégrés sur les puces et l'accroissement de la fréquence d'horloge.
- les changements dans l'architecture (ajout de plusieurs niveaux de cache, parallélisme d'instructions, prédiction de branchement, etc..) ont permis de contrebalancer l'allongement du temps d'accès a la mémoire RAM exprimé en cycle processeur.
- les nouveaux modèles d'exécution (hyperthreading, multithreading) ont aussi contribué à augmenter la puissance du calcul par l'exploitation du parallélisme.

En ce qui concerne la miniaturisation, la question se pose de savoir si la diminution de la taille des transistors continuera indéfiniment ou, si par contre, des barrières physiques (en particulier la granularité de matière) rendront impossible la fabrication de transistors plus petits. Aujourd'hui, les concepteurs et les fabricants de puces conviennent généralement que la taille des transistors continuera à diminuer jusque des dimensions nanométriques et qu'ils seront capables de fabriquer des nanopuces (c'est-à-dire, les puces avec les transistors nanométriques) dans la prochaine décennie. Alors, le nombre de transistors sur une puce atteindra des centaines de milliards. Est-ce que pour autant la conception des puces ne rencontra plus aucun problème et que l'augmentation constante de la performance des processeurs est garantie ? En dépit des pronostics optimistes, il faut tenir compte que des défis apparaissent plus nombreux qu'a n'importe quel moment de l'histoire de la conception des processeurs. Dans cette thèse, nous nous concentrons sur les trois problèmes suivants :

1. **L'extensibilité**. L'architecture uniprocesseur ne garantit plus l'augmentation de la performance parce qu'il y a une crise au niveau d'une « utilisation » efficace du nombre augmenté de transistors. Autrement dit, il n'y a aucune nouvelle solution architecturale pour monoprocesseur qui garantirait une augmentation de la performance désirée. La tendance actuelle consiste à l'augmentation du nombre de cœurs dans le cadre d'architectures multiprocesseurs symétriques. Pourtant, cette architecture ne permet pas le passage à l'échelle et ce n'est donc pas une solution viable à long terme.

2. **La puissance dissipée**. La course aux hautes fréquences a été arrêtée parce la dissipation d'énergie devient prohibitivement grande aux fréquences élevées.
3. **La sûreté du fonctionnement dans les technologies massivement défectueuses**. La longueur du canal d'un transistor nanométrique devient si petite qu'elle est de l'ordre de quelques couches atomiques ce qui entraîne une dispersion plus grande des caractéristiques de transistors et, par conséquent un accroissement de la fraction d'éléments défectueux (et potentiellement défaillants).

Dans cette thèse nous présentons des solutions au niveau architectural pour améliorer la sûreté de fonctionnement des futures puces en augmentant progressivement la performance. Nous proposons une nouvelle méthodologie de conception d'une puce multiprocesseur pour résoudre au moins partiellement les problèmes mentionnés dans le paragraphe précédent. Notre approche est basée sur deux idées principales :

- En raison de l'augmentation de la complexité, les futures puces devraient évoluer vers des architectures régulières et réplicatives.
- Les mécanismes de test de et de configuration (nécessaire pour tolérer les blocs défectueux) devraient être exécutés par la puce elle-même, avec un contrôle externe réduit au minimum. Autrement dit, les futurs nanochips fabriqués dans les technologies massivement défectueuses devraient mettre en oeuvre des mécanismes auto-immunitaires, comme les tissus biologiques, pour maintenir leur capacité de traitement.

Dans cette thèse, nous considérons comme mécanismes auto-immunes l'auto-diagnostique du chip par tests mutuels entre cœurs adjacents et l'auto-configuration des communications. Nous étudions la limite de l'efficacité de ces mécanismes en fonction de la fraction de cœurs et liens défectueux. A l'exécution, nous décrivons et étudions l'implémentation de mécanismes d'exécution redondants afin d'éliminer les erreurs induites par les fautes transitoires ou permanentes.

F2. Auto-configuration

F2.1 Architecture réplicative multi-cœur

Nous étudions dans cette thèse des architectures comportant des centaines de cœurs basées sur des topologies d'interconnexion régulières. L'idée d'architectures multi-cœurs massivement parallèles et régulières est fondée sur les raisons présentées dans la partie précédente, c'est-à-dire que :

- Seules des architectures permettant le passage à l'échelle (c'est-à-dire dont le réseau peut être étendu sans remettre en cause la connectivité de chaque nœud) peuvent fournir des solutions garantissant l'accroissement constant de la puissance de traitement.
- Les technologies à l'échelle nanométrique seront inévitablement défectueuses.
- Le taux élevé de défauts au niveau du transistor et le grand nombre de transistors sur une puce rendront la production des puces sans défauts pratiquement impossible. En conséquence, toute puce comportera une fraction de blocs de base défectueux.

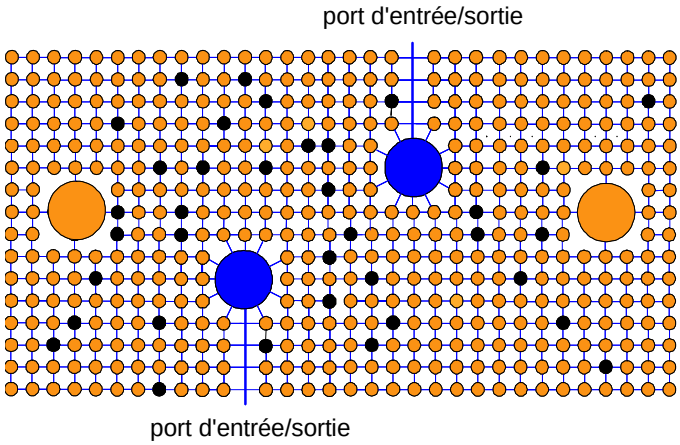


Figure 1. Modèle de l'architecture réplicative

La figure ci-dessus présente un exemple d'architecture multi-cœurs réplicative. Les cercles représentent des nœuds qui dans notre exemple sont organisés comme un réseau carré. Bien sûr, d'autres topologies peuvent aussi être considérées et nous étudions dans le mémoire des topologies dont la connectivité de chaque nœud s'échelonne de 3 (réseau héxagonal) à 5 (réseau carré à 2 couches). Comme nous considérons des technologies défectueuses, les cercles noircis représentent des nœuds défectueux qui ne peuvent pas participer au traitement.

Les méthodes décrites dans notre travail ont pour but essentiel de faire que ce type d'architectures multi-cœurs tolère les nœuds défectueux, c'est-à-dire qu'une fraction de cœurs défectueux provoque un affaiblissement de la puissance de traitement de la puce et non une défaillance globale. Notons également que les positions des nœuds défectueux sont a priori inconnues. Chacun des nœuds comprend un cœur et un router assurant la communication avec des cœurs voisins. En général, la complexité des cœurs peut être variable : les grands cercles représentent des cœurs plus puissants que les autres. Cependant, dans ce travail nous considérons un réseau homogène ou tous les cœurs sont identiques. Le port d'entrée/sortie (PES) est un circuit spécial responsable de la communication avec l'extérieur, par exemple avec de la mémoire RAM.

F2.2 Principes d'auto-configuration

Les idées principales que nous proposons concernant l'auto-configuration d'une puce incluent :

- L'auto-diagnostique du chip basée sur des tests mutuels entre cœurs adjacents.
- L'auto-désactivation des cœurs isolés.
- L'auto-configuration des routes de communication.
- L'allocation dynamique et l'exécution redondante de tâches.

Ainsi, ce travail peut être considéré comme la présentation d'une méthodologie de l'auto-configuration qui permet de tolérer de défauts permanents et des fautes transitoires dans les puces multi-cœurs massivement défectueuses. Nous étudions également l'efficacité des mécanismes d'auto-configuration proposés en fonction de la fraction de nœuds défectueux Typiquement, on considère que l'architecture multicoeurs peut comporter jusque 40% de coeurs défectueux. Il faut souligner que ce paramètre ne dépend pas seulement de la technologie, mais aussi de la complexité du coeur. Il faut considérer la probabilité $p_{f,N}$ du coeur d'être défectueux comme un paramètre résultant d'un compromis entre la puissance recherchée du coeur (plus le coeur est complexe et donc puissant, moins les contraintes de parallélisation des applications sur plusieurs cœurs sont fortes) et la fraction acceptable de coeurs défectueux dans le réseau, qui n'est rien d'autre en moyenne que $p_{f,N}$.

F2.3 Auto-diagnostique du chip basé sur le test mutuel entre cœurs adjacents

L'idée de l'autotest résulte de l'évidence que le diagnostique externe des nano-puces très complexe devient très difficile à mettre en œuvre, voire impossible. L'autotest est fondé sur l'idée que tout le processus de diagnostic des cœurs défaillants est exécuté par le chip lui-même. De plus, les mécanismes du test doivent être associés à des actions de configuration visant à isoler les cœurs défectueux pour assurer malgré tout un fonctionnement correct du circuit même si la puissance de traitement est réduite.

L'autotest logiciel est une méthode relativement nouvelle basée sur un test fonctionnel. Il est surtout adapté au diagnostique des processeurs. Son avantage important est que chaque processeur joue le rôle d'un circuit de test. Toutes les phases du test, c'est-à-dire la génération des vecteurs de test, le test lui-même ainsi que la collection de réponses, sont exécutées par le processeur lui-même. Autrement dit, le processeur est testé au moyen de l'exécution d'un programme de test.

Il faut souligner que dans la méthodologie proposée, la découverte des cœurs défectueux n'est exécutée qu'une seule fois, au démarrage. Les résultats du test initial sont gardés dans la mémoire non volatile des cœurs pour ne pas disparaître quand le chip n'est pas alimenté.

Dans la méthode la plus simple, chaque nœud s'auto-diagnostique. Cependant, cette approche est dangereuse puisque la sûreté de fonctionnement repose sur des actions décidés justement par les nœuds défectueux, donc au comportement imprévisible. En particulier, un nœud défaillant peut se déclarer non-défaillant. Dans ce travail, nous étudions les tests mutuels qui éliminent ce problème. Plus précisément, chaque nœud compare les résultats de son propre test avec ceux de ses voisins adjacents. Puis, il exécute une action simple : il arrête toute communication avec chaque voisin ayant produit des résultats différents. En d'autres termes, les liens vers ces nœuds adjacents sont logiquement déconnectés. Il faut souligner que l'on suppose que les nœuds défaillants peuvent opérer de façon aléatoire alors ils peuvent (mais ne doivent pas forcément) déconnecter leurs voisins.

Globalement, il résulte que :

- les bons nœuds seront toujours connectés entre eux.
- les bons nœuds seront toujours déconnectés des nœuds défaillants.
- les nœuds défaillants peuvent être connectes ou déconnectés mais cela est sans importance. Ce qu'il est capital de souligner, c'est que le mécanisme d'isolation basé sur les tests réciproques dépend **uniquement** des actions des bons nœuds sans faire aucune hypothèse sur les actions des nœuds défaillants.

La figure suivante est un exemple qui montre le résultat des tests adjacents pour une architecture en grille comportant 9x7 coeurs. Cette architecture comporte 4 ports d'entrée sortie (marqués N,W, S, E) et 14 cœurs défectueux représentés par des carrés de couleur noir. La figure montre que tous les nœuds défaillants ont été isolés et n'exercent aucune influence sur les autres nœuds. Nous voyons aussi que quelques bons nœuds (en particulier ceux en haut à gauche) ont été isolés de la zone principale comprenant les PES si bien qu'ils ne pourront pas être contactés pour participer au traitement. Dans les chapitres suivants nous étudions l'impact de la fraction des nœuds défaillants sur l'accessibilité des nœuds dans le réseau multi-cœur.

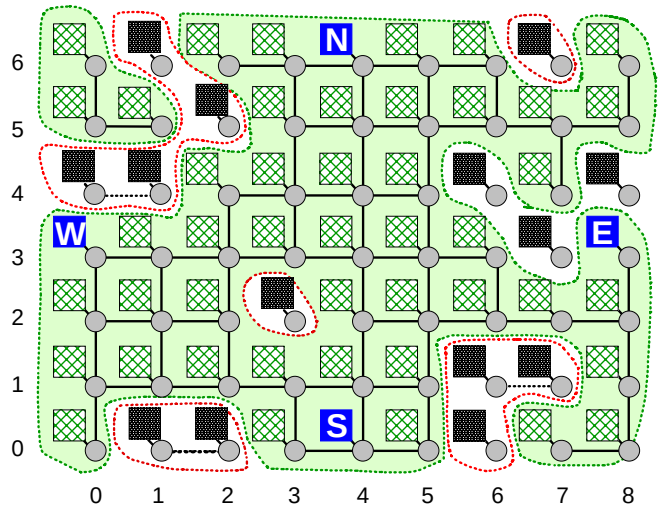


Figure 2. Exemple d'auto-partitionnement de la puce multi-cœur résultant de tests réciproques.

Globalement, la figure montre que les tests entre premiers voisins découpent le réseau en zones simplement connexes de cœurs valides interconnectés. Le problème posé cependant est d'assurer les communications entre les cœurs valides dans une topologie comportant des trous (i.e., des cœurs diagnostiqués comme défaillants) en positions indéterminées.

F2.4 Auto-configuration des routes de communication

Les protocoles de communication doivent garantir la communication sans échec entre les nœuds. Comment transmettre les messages en présence des nœuds défectueux et isolés? La plupart des mécanismes de découverte de trajets sont basés sur la diffusion (broadcast) d'un ou plusieurs messages. Cependant, diffuser fréquemment des messages peut causer une augmentation significative (voire une congestion) du trafic conduisant à l'accroissement des latences de communication et la réduction de la puissance de traitement de la puce. La solution que nous considérons pour éviter ce problème peut être décomposée en trois phases comme suit :

1) Au démarrage, chaque port d'entrée/sortie (PES) diffuse un message de *Route* (MR) à tous les nœuds. Ce message comporte un champ de route qui est mis à jour pendant la propagation, c'est-à-dire que chaque nœud traversé ajoute à ce champ le routage exécuté localement. Par exemple on peut coder le routage local sur 2 bits si la connectivité du nœud est 4. Autrement dit, le message stocke la route suivie durant sa propagation de sorte qu'un nœud le recevant possède une route complète vers le PES l'ayant émis.
2) Chaque nœud stocke le trajet vers le PES dans sa mémoire locale. Il envoie ensuite un message de *Confirmation* (MC) au PES en incluant le trajet. Notons que le message de confirmation n'est pas diffusé de façon isotrope mais qu'au contraire, il suit en sens inverse le trajet indiqué par le MR.
3) Le PES reçoit tous les messages de confirmation et stocke dans sa mémoire locale toutes les routes découvertes durant cette phase et menant vers tous les nœuds accessibles. Les routes ainsi stockées dans la mémoire peuvent être utilisés ultérieurement pour permettre les communications dans une puce en opération.

Il faut souligner que l'espace mémoire nécessaire pour stocker les routes est assez petit dès lors que l'on n'utilise que quelques bits de codage par nœud traversé. Bien sûr, le mécanisme de découverte des routes pourrait être répété périodiquement en opération pour accroître la sûreté de fonctionnement.

F2.5 Etude de la découverte des routes

Le nombre de nœuds accessibles sur une puce est un paramètre important que l'on peut utiliser pour décider de valider un circuit lors de sa fabrication. Pour étudier ce paramètre, nous avons utilisé le simulateur MASS. Brièvement, ce logiciel multiagent, écrit au LAAS en langage C++, permet de simuler la communication dans une puce multi-cœur. Le mécanisme de découverte des routes suit les trois phases décrites dans la section précédente. Nous avons étudié l'accessibilité (c'est-à-dire le nombre de cœurs accessibles par le PES) en fonction de la fraction de cœurs ou de liens défectueux dans des architectures uni- ou multi-ports, et pour différentes topologies d'interconnexion. Nous avons étudié l'accessibilité pour:

1) Le réseau hexagonal de connectivité C=3 (chaque nœud possède trois voisins, sauf pour les nœuds sur les bords).
2) Le réseau carré de connectivité C=4 (chaque nœud possède quatre voisins, sauf pour les nœuds sur les bords).
3) Le tore de connectivité C=4 (chaque nœud possède quatre voisins). Cette topologie est repliée et ne comporte plus de bords.
4) Le réseau carré à deux couches de connectivité C=5 (chaque nœud possède cinq voisins, sauf pour les nœuds sur les bords).

F2.5.1 Architecture uniport

Les résultats des simulations d'accessibilité présentés dans ce résumé de thèse ne représentent qu'une très faible fraction des résultats reportés dans la thèse qui comporte environ 140 pages. Les figures 3 et 4 ci-après présentent la réduction de l'accessibilité du PES dans une architecture uniport en fonction de la fraction de cœurs défectueux (et isolés dans le réseau) pour les quatre topologies d'interconnexions. L'accessibilité est décrite de la manière suivante : l'axe x représente la fraction accessible de cœurs de la puce (f_C) alors que l'axe y représente la probabilité $p(f_C)$ qu'au moins la fraction f_C soit accessible, donc disponible pour

exécuter des tâches. La probabilité $p(f_C)$ a été calculée en tant que moyenne statistique à partir d'un grand nombre de simulations. Cette façon de présenter les résultats de simulation permet en fait de déterminer le pourcentage des puces fabriquées qui auront une accessibilité plus grande qu'un seuil f_C défini par l'utilisateur. Considérons par exemple la figure 3. Elle correspond à un réseau comportant 20% de cœurs défaillants, de sorte que de toute façon, l'accessibilité (moyenne) du PES ne peut dépasser 0.8. Cela explique la présence de la zone grisée inaccessible sur la partie droite de la figure. Maintenant, considérons le point B situé sur la courbe C=3, c'est-à-dire pour un réseau hexagonal. Ce point signifie que la probabilité que le PES atteigne 75% des cœurs n'est que de 0,6. Donc, statistiquement, si on choisit de sélectionner les puces qui permettent de joindre au moins 75% des coeurs sur 80 % possibles, le rendement de production ne sera que de 60 %. Similairement, le point A montre que la probabilité de contacter au moins 77 % des cœurs (sur 80% non-défectueux) est égale à 0.94 pour le réseau carré.

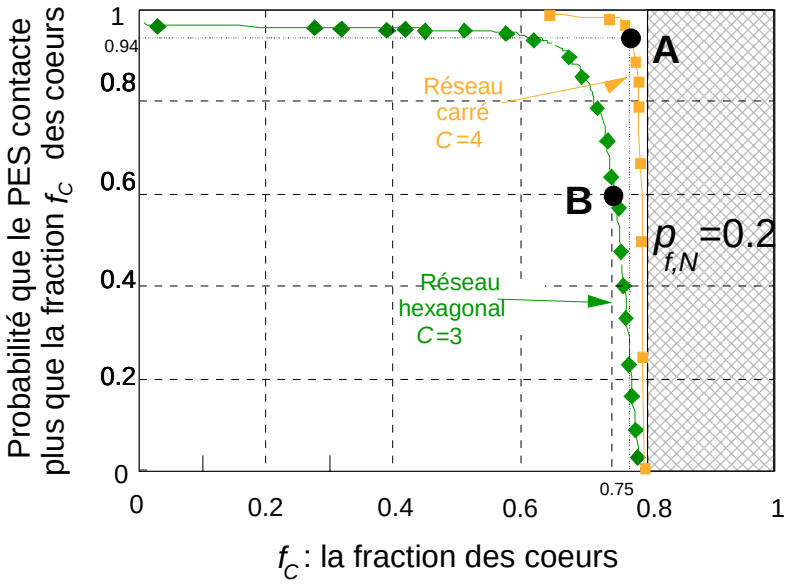


Figure 3. Simulations d'accessibilité dans un chip comportant 20% de cœurs défaillants

La figure 4 suivante montre que l'accessibilité se dégrade radicalement quand il y a 40% de cœurs défectueux dans le réseau. En effet, supposons que l'on exige que le PES puisse contacter au moins 53% des cœurs (parmi 60% non-défectueux). La figure montre que cela est quasi-impossible dans un réseau hexagonal ou un réseau carré! Les résultats pour le tore (représentés par le point B) sont légèrement meilleurs, sans être satisfaisants puisque

seulement 20% des puces réaliseraient la condition mentionnée. Seule la topologie avec la plus haute connectivité C=5 (c'est-à-dire le réseau carré à 2 couches) fournit des résultats satisfaisants, puisque la probabilité de contacter au moins 53% des cœurs est égale à 0.9.

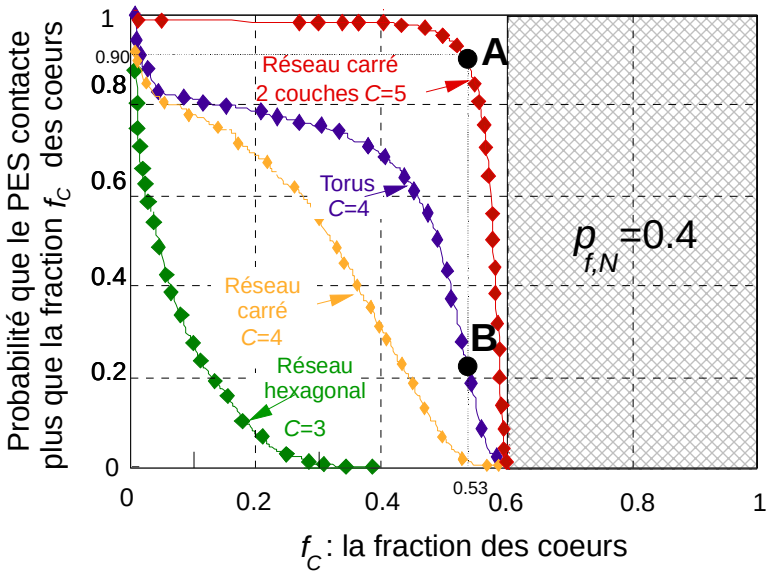


Figure 4. Simulations d'accessibilité dans un chip comportant 40% de cœurs défaillants

Ces deux figures illustrent un résultat général de la thèse qui est que la topologie carrée (donc de connectivité 4) peut être considérée comme une solution satisfaisante aussi longtemps que la probabilité que le nœud soit défectueux ne dépasse pas 0.2. Au-delà de ce seuil, la connectivité du nœud doit être augmentée pour maintenir l'accessibilité à des valeurs élevées, disons typiquement supérieure à 80-90 %.

Nous avons aussi étudié la réduction de l'accessibilité due à la présence de liens défectueux dans le réseau (section 4.2.2 de la thèse). En fait, les interconnexions défectueuses sont mieux tolérées que les cœurs du point de vue du maintien de l'accessibilité, ce qui n'est pas une réelle surprise. Nous avons conclu que pour la plupart des topologies étudiées (réseau hexagonal, carré, en tore ou à deux couches), une fraction R de liens défectueux réduit approximativement deux fois moins l'accessibilité que le même pourcentage des cœurs défectueux. Cette conclusion peut être utilisée pour équilibrer la protection que l'on décide d'implémenter pour protéger la puce à la fois contre les liens et les cœurs défectueux. Les

résultats de simulation ont permis d'estimer le rendement de production en fonction de la probabilité de défaillance du nœud ou de l'interconnexion selon la topologie du réseau.

Comme la découverte des routes basée sur la diffusion isotrope (broadcast) peut provoquer des congestions dans le réseau résultant de la corrélation des trajets dans un réseau asynchrone, nous avons développé et simulé une nouvelle méthode pour équilibrer le trafic (section 4.2.4 de la thèse).

F2.5.2 Architecture multi-port

Les architectures multiports ont aussi été étudiées extensivement. Nous avons :

- Calculé le surcoût en circuiterie (i.e., en redondance d'implémentation) nécessaire pour tolérer les PES défectueux (section 4.4.1 de la thèse). Il ne faut pas oublier que les PES sont des éléments particulièrement critiques de la puce puisqu'ils sont des points de passage obligés de ses communications avec son environnement. Il faut donc implémenter des redondances pour tolérer à la fois l'occurrence des fautes de fabrication et des fautes dynamiques en opération dans les PES. On a conclu que la protection des ports devient extrêmement difficile si la probabilité de défaillance du nœud dépasse 0.3, surtout si la taille du PES est comparable avec la taille du nœud.

- Etudié le meilleur placement des ports dans le réseau carré comportant des cœurs défectueux du point de vue du partage des cœurs et de l'accessibilité des PES (section 4.4.2 de la thèse). La figure 5 illustre cette étude. Elle représente un réseau carré avec 100 nœuds et quatre PES qui peuvent être positionnés dans les places indiquées par les lettes A, B, C ou D. La figure à droite donne le partage des cœurs, c'est-à-dire la fraction de cœurs qui sont soit défaillants (■), soit accessibles par 1 à 4 PES (symboles ▧, □, ▩, ▒), soit inaccessibles (▩) en fonction de la fraction de nœuds défaillants dans le réseau. La figure montre clairement que l'accessibilité par plusieurs PES se dégrade quand plus de 25% des cœurs sont défaillants dans le réseau. La meilleure accessibilité aux cœurs est obtenue quand les PES sont positionnés sur les places marquées par les lettres B.

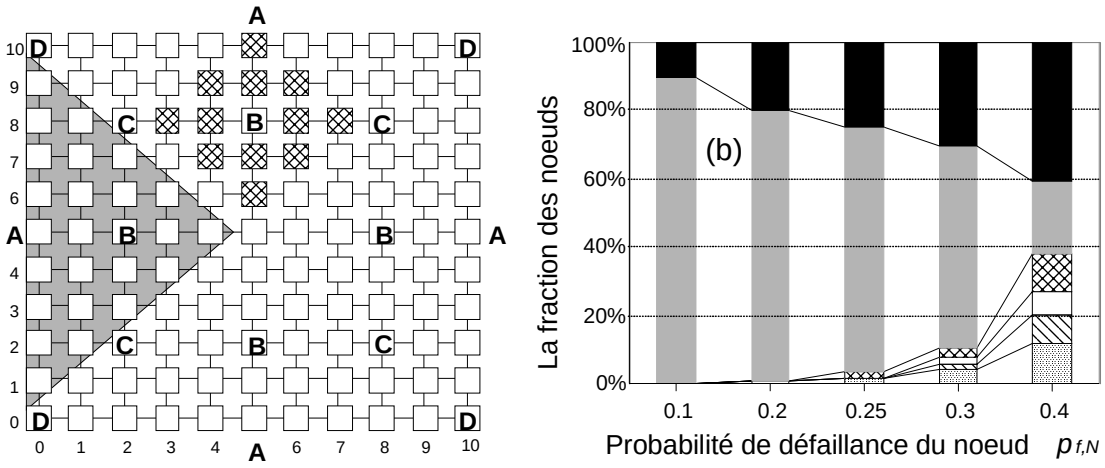


Figure 5. Architecture multiport et les résultats d'étude de l'accessibilité des nœuds. : Cœurs défaillants, : cœurs partagés par 4 PES, : cœurs partagés par 3 PES, : cœurs partagés par 2 PES, : cœurs partagés par un seul PES; : cœurs inaccessibles

- Etudié l'efficacité de la découverte des routes et défini deux métriques de validation des puces (section 4.4.3 de la thèse). Les métriques reposent (comme pour les architectures uniport) sur la capacité des ports de contacter les cœurs selon la distribution de la charge de traitement. Les simulations montrent que dans le cas de lourde charge de travail, il est très difficile de garantir l'équilibre de la charge de travail parmi les ports si la fraction de nœuds défectueux dans le chip dépasse 20%. L'impact de liens défectueux sur l'accessibilité des nœuds est moins grande (par rapport à l'impact des nœuds défectueux) que dans le cas des architectures uniport.

- Montré que l'accroissement de la connectivité des PES (importante pour préserver la bande passante du PES en présence de lien défaillants) n'a qu'un effet positif mineur sur la capacité du port de contacter les cœurs (section 4.4.3 de la thèse).

F3. Allocation et exécution des tâches

L'allocation des tâches sur les cœurs disponibles afin d'exécuter un traitement ne peut être effectuée que de façon dynamique (c'est-à-dire à l'exécution) dans les puces défectueuses. Il a deux raisons essentielles à cela :

1. Le compilateur ne peut pas savoir quels cœurs sont défectueux dans une puce. En effet, leurs positions changent d'une puce à l'autre.
2. Il ne peut pas savoir non plus quels cœurs sont disponibles pour exécuter un traitement à un instant donné puisque cela dépend de la compétition entre les différents PES (pour verrouiller les ressources de traitement) et de l'historique de traitement de la puce.

Aussi avons-nous développé un protocole de verrouillage des cœurs et d'allocation des tâches qui permet à plusieurs PES de partager les nœuds (section 5.1 de la thèse). Ce protocole est basé sur l'échange de deux messages, d'une part l'émission de requêtes de verrouillage par les PES et sur la réponse (accord ou rejet) des cœurs disponibles pour réserver le traitement. C'est une variante du *contract net protocol* (CNP). Il faut souligner que l'arbitrage dans la réservation des ressources de traitement est fait par chaque nœud contacté (qui donne ou ne donne pas sont accord. Ce mécanisme est donc totalement distribué et ne requiert aucune coordination entre les PES. Il est aussi indépendant du nombre de PES.

Nous présentons également des simulations qui calculent l'accroissement de la latence de communication induite par la présence des nœuds défectueux (section 5.2). Néanmoins, il a été estimé que la dégradation de la performance provoquée par la latence est négligeable en comparaison avec la dégradation provoquée par la réduction du nombre des bons nœuds sur une puce. La figure ci-dessous montre la peine de communication pour le réseau carré. Elle est mesurée comme le nombre moyen des sauts additionnels entre un nœud et le PES résultant de la présence des nœuds défectueux. Les barres verticales démontrent l'écart type autour de la moyenne. Clairement, l'accroissement de la latence devient important quand $p_{f,N}$ dépasse 0.2.

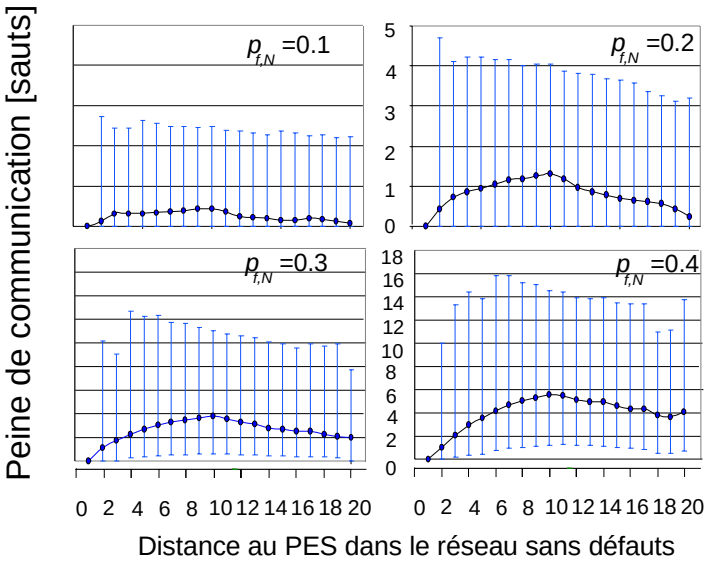


Figure 6. L'accroissement de la latence de communication dans le réseau carré

Nous avons également présenté et comparé trois méthodes d'exécution des tâches basées sur la redondance spatiale ou temporelle (section 5.3 de la thèse) permettant de tolérer l'occurrence de fautes transitoires en opération. Les meilleurs résultats de simulations (conformément à nos modèles) ont été fournis par la méthode basée sur la redondance d'exécution temporelle. Cependant, la méthode basée sur la duplication spatiale devrait être privilégiée (bien qu'elle donne des résultats un peu moins bons) parce qu'elle permet aussi de tolérer les fautes permanentes présentes en opération, éventuellement des fautes non détectées pendant la phase de diagnostique (en raison de la couverture incomplète du test initial) ou bien des fautes dues au vieillissement.

A titre illustratif, nous décrivons brièvement ci-dessous l'étude de l'efficacité de traitement lorsque l'on considère la redondance d'exécution temporelle en présence de fautes transitoires. L'étude est menée de la façon suivante en utilisant le simulateur MASS :

1) On simule un réseau carré 10x10= 100 processeurs avec 4 PES.
2) Le simulateur génère au hasard une fraction de nœuds défectueux dans le réseau, conforme au paramètre $p_{f,N}$.
3) Chaque port exécute le mécanisme de découverte des routes.
4) Chaque port commence à allouer des tâches en exécutant le CNP décrit précédemment, ce qui permet de partager dynamiquement les cœurs disponibles.

5) Pendant l'exécution de la tâche par un cœur, le simulateur génère aléatoirement des erreurs transitoires conformes au temps moyen entre défaillance (MTBF).
6) Le mécanisme d'exécution des tâches basé sur la redondance temporelle est implémenté : chaque cœur exécute une tâche deux fois. La troisième exécution est nécessaire si les résultats des deux exécutions précédentes sont différents comme conséquence d'une erreur transitoire.
7) Le simulateur compte le nombre de tâches complétées par tous les cœurs au cours de la simulation.

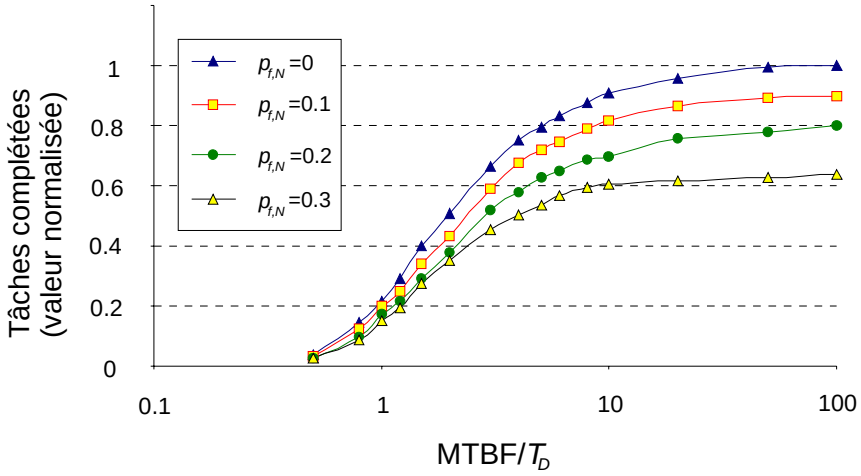


Figure 7. Résultats des simulations de l'allocation et l'exécution des tâches

La figure 7 permet d'analyser l'impact des fautes transitoires et permanentes sur la performance d'une puce. Elle montre que la fraction de nœuds défectueux est le facteur dominant entraînant une réduction de la performance. Néanmoins la dégradation induite par les erreurs transitoires devient dominante quand le temps moyen entre pannes devient comparable à la durée moyenne de la tâche T_D. De plus, nous avons révélé qu'il est très difficile d'estimer l'impact des fautes sur la puissance de traitement d'une puce parce que la dégradation de la performance dépend largement des paramètres d'opération du chip tel que la charge de travail.

F4. Conclusion

Comment assurer la sûreté de fonctionnement des futures puces complexes fabriquées dans les futures nanotechnologies ? L'accroissement des défauts (lié à la réduction des dimensions) sera probablement si important que la protection des circuits et la tolérance aux fautes dans la seule couche physique sera insuffisante pour garantir la sûreté de fonctionnement. Elle devra être acquise en mettant en œuvre des mécanismes de protection et de tolérance aux fautes dans les différentes couches, c'est-à-dire au niveau du composant, du circuit, au niveau architectural et dans les mécanismes d'exécution. Plus précisément, le rôle de la protection exécuté par chaque couche est de réduire l'impact des fautes non tolérées et se propageant depuis les couches inférieures. C'est donc la protection combinée de tous les niveaux qui permettra d'assurer la sûreté du fonctionnement des futures puces.

Cette thèse propose une méthodologie de tolérance aux fautes au niveau architectural dans les puces multi-cœurs massivement défectueuses. L'idée principale de ce travail est qu'une puce devra être organisée en une architecture réplicative (ce qui résout en grande partie le problème de la complexité) et devenir aussi autonome que possible pour augmenter sa résilience contre les défauts permanents et les erreurs transitoires apparaissant en opération. C'est pourquoi nous introduisons plusieurs méthodes d'auto-configuration de la puce qui permettent de détecter et isoler les cœurs défectueux, de désactiver les cœurs isolés, de configurer les communications et de diriger l'allocation et l'exécution des tâches. L'efficacité des méthodes est étudiée en fonction de la fraction d'interconnexions ou de cœurs défectueux et du taux d'erreurs transitoires. S'il faut simplement résumer ce travail sur un seul résultat, la conclusion principale est que jusqu'à typiquement 20% de cœurs défectueux, les méthodes d'auto-configuration proposées permettent de tolérer efficacement les éléments fautifs dans des réseaux carrés, mais qu'il devient bien difficile de les tolérer au-delà de cette valeur limite. Une solution est alors d'accroître la connectivité des nœuds.

Table of contents

1. Introduction

The invention of general-purpose processor truly revolutionized the world. Who would have thought in 1970 that the simple Intel 4004 chip will be one of the most important human inventions in the 20th century? The idea behind designing this microprocessor was quite simple: it was meant to be used in a calculator! Actually, the only reason why Intel developed a general-purpose chip (GPC) was that they had no experience in designing dedicated chips. Other processor architectures were designed in the following years and with the Intel 8080 processor, manufactured in 1974, a new era of microprocessors became a reality. Processors became the leader chips in the world of constantly evolving electronics. Many new ideas and technologies in electronics were first implemented to improve the processing power of microprocessors. In fact, some ideas were even especially and intentionally developed to be used in the microprocessor design.

The permanent increase during over three decades of the processing power of GPC results from the conjunction of miniaturization (i.e., the reduction of dimensions of transistors and interconnects), of architectural evolutions of GPCs and of changes in the execution model of applications:

- Miniaturization enables increasing the number of transistors and the operation frequency.
- Changes in the architecture (such as the introduction of several levels of cache, of the instruction parallelism and of branch prediction, etc.) have mostly enabled mitigating the increase of access latency to the main SDRAM memory.
- New execution models (such as multi or hyper-threading) have also been contributing to increase the processing power by avoiding pending states of the processor in multitask context.

Regarding miniaturization, we must settle the inevitable question to know whether the observed downsizing will continue indefinitely or if ultimately, some physical barriers (in particular the granularity of matter) will make the fabrication of smaller transistors impossible. This question was raised several decades ago but, until now thanks to constant technology evolutions, chip manufacturers have been able to sustain this trend. Today, designers and manufacturers generally agree that the transistor size will continue to decrease to reach nanometric dimensions and that they will be able to fabricate nanochips in the next decade (i.e., chips with nanometric transistors). Fig. 1 displays the evolution of the transistor count, the chip area and the technology feature size. It also shows the trends expected until 2020, on the basis on an exponential extrapolation which is the continuation of the evolution during 3 decades. In particular, the upper most curve shows that the number of transistors on

chip is expected to increase to reach hundreds of billions. Does that mean that designers face no problems and that the constant increase of the processor's performance is ensured?

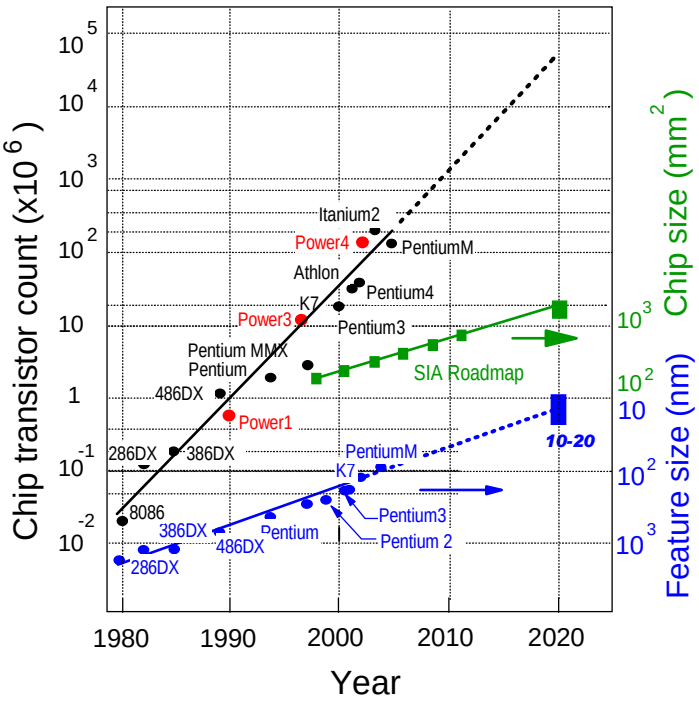


Fig. 1: Processor architecture roadmap

Despite the optimistic extrapolations shown in Fig. 1, there are now more challenges that need to be taken into consideration than in any moment of processor design history. In this thesis, we focus on the three following problems:

1. **The architectural scalability**: Uniprocessor architecture no longer guarantees the desired performance gain because there is a crisis on how efficiently "using" the increased number of transistors. In other words, there are no new architectural solutions in uniprocessors which would ensure the desired performance gain. The current trend consists in increasing the number of cores in the framework of symmetrical multiprocessor architecture. However, this architecture lacks scalability and is not a long-term solution.
2. **The power dissipation**: Designers can no longer increase the chip operating frequency because the power dissipation would become prohibitively large at higher frequencies.
3. **The operation dependability in massively defective technologies**: The transistor channel size is becoming so small that it will be comparable to atomic dimensions, increasing the dispersion of transistor characteristics and consequently the fraction of defective (and potentially failing) elements. In fact, the defect ratio will probably become so high that the production of fault-free chips will be very difficult and the production yield will dramatically decrease. Moreover, the risk of transient errors during the processor operation will have to be considered especially in the nanoscale technologies.

Not surprisingly, designers already work on new ideas and solutions which will enable overcoming in the future the above mentioned problems. The improvements are possible at many levels, from the component-level to the application-level.

In this thesis we present architectural-level solutions to improve the operation dependability and steadily increase the performance of multi-core chips. We present our idea of a future multiprocessor architecture and (re)configuration mechanisms which partially solve the problems mentioned in the previous paragraph. Self-organization is the basic and underlying idea of this thesis. Our view is that:

1. Due to the increase in complexity, future chips should evolve towards replicated architectures built from a small number of well-controlled building blocks, and above all that
2. The basic mechanisms of fault diagnosis and configurations (necessary to tolerate the faulty blocks) should be executed by the chip itself, with as little as possible external control. In other words, future nanochips fabricated in the massively defective technologies should exhibit some kind of autoimmune mechanisms similarly to biological tissues to maintain their processing capabilities. In this thesis, we essentially consider chip self-test via mutual diagnosis and route self-configuration (RSC) as autoimmune mechanisms, and we study the limit of the RSC efficiency as a function of the fraction of defective building blocks. We consider two kinds of defective blocks: cores, or internode links. At runtime, special mechanisms based on execution redundancy should be also implemented to eliminate errors induced by transient faults.

This thesis is organized as follows:

- In the second chapter, we present the current challenges in the design and fabrication of VLSI chips. We also briefly introduce some new emerging solutions which should be envisaged in the future nanoscale technologies.

- In the third chapter, we present the architecture of general-purpose nanochips, which we consider to be a solution to the challenges presented in chapter two. Moreover, we describe several chip self-configuration mechanisms, which allow coping with faults in the future nanoscale chips.

- The fourth chapter concentrates on estimating the impact of permanent faults on chip communication performance. Both core defects and interconnect defects are studied and we present both simulation results and analytical calculations.

- In the fifth chapter, we analyze the impact of both permanent and transient faults on task allocation and execution in a chip. Several protection methods against transient faults are described and compared.

- The general conclusion is presented in the sixth chapter.

2. Issues and challenges in current VLSI chips

2.1 Introduction

The world’s first microprocessor was introduced in 1971, a simple Intel 4004 architecture which was composed of only 2,300 transistors. According to Moore’s Law [1], the number of transistors in a chip should double every 18 months. In other words, every year and 6 months twice more complex chips should be fabricated. Following this rule, the processing core evolved from the relatively primitive 4004 processor to powerful multicore processors with almost billion transistors. The list of major improvements to the architecture of mainstream single core processors is shown below:

- 8 bit bus width in 8008 processor (1972)
- Single 5V supply voltage in 8085 processor (1974)
- Instruction pipelining in 8052 processor (1975)
- 16 bit bus width in TMS9900 processor (1976)
- Interrupt controller and DMA controller on chip in 80186 processor (1982)
- 32 bit bus width in iAPX432 processor (1981)
- Multitasking in 80286 processor (1982)
- Branch prediction in SPARC processor (1985)
- Level 1 cache on chip in 80486 processor (1989)
- Integrated floating point unit chip in 80486 processor (1989)
- Out-of-order execution in POWER1 processor (1990)
- 64 bit bus width in PowerPC 620 processor (1994)
- Superscalar architecture in Pentium processor (1995)
- Level 2 cache on chip in Pentium Pro processor (1995)
- SIMD instruction set in Pentium Pro processor (1997)
- SIMD instruction set for floating point operations in K6-2 processor (1998)
- Advanced Transfer Cache in Pentium II Xeon processor (1998)
- Prefetch cache predictor and a translation look-aside buffer in Athlon 4 processor (2001)
- HyperThreading in Pentium 4 processor (2002)
- HyperTransport in Athlon 64 processor (2003)

Note that meanwhile, the operating frequency increased from 104 kHz to impressive several gigahertz, the supply voltage decreased from 12V to about 1.3V and the transistor channel size reached nanometer dimensions.

As we scroll down the above list, we may notice that the architectural improvements themselves have become more and more complex in terms of functionality and especially in terms of transistors' number. All these changes were made to achieve one main goal: increasing the chip processing power. It is undisputable that along three decades, this struggle has been successful. The processing abilities of contemporary processors reached a performance level that the manufacturers of the first processors never dreamt of.

However, in any domain, it is always necessary to question the cost of the success. More precisely, in the field of processor design, we should compare the processing power gain with the transistor count increase. We can introduce here a sort of efficiency metrics $perf / n_t$, where *perf* is the performance (measured for instance as the number of instructions executed per second) and n_t the transistor count. Of course, ideally $perf / n_t$ should (at least) stay constant, which means that doubling of the number of transistors would allow twice greater performance, tripling of the number of transistors would allow three times greater performance etc. Nevertheless, in reality, the increase of the processing power is far from this ideal case. The relation of the performance *perf* versus the number of transistors n_t for different type of processors shows that simpler processors are usually more efficient than the most complex ones! Moreover, the ratio $perf / n_t$ has systematically decreased as the chip size increased. In other words, the price to pay in terms of number of transistors (or chip area) for gaining more and more power has grown inexorably. Moreover, recent studies show that the dissipated power per transistor also increases which, as the chip size increases, is quickly becoming a significant design problem.

In the following sections of this chapter, we precisely describe the problems which have to be solved so that processors continue to evolve toward more processing power.

2.2 Variability problem in the future nanotechnologies

2.2.1 Manufacturing faults

Current processors are fabricated using the 45-nm process technology[(1)]. As the technology will scale down, the manufacturers will face new problems resulting from atomic fluctuations. Let us now describe these problems more precisely. Intuitively, one may guess that the influence of atomic fluctuations will increase greatly when reducing the transistor dimensions. For example, as the doping density stays at the same level, we might be able in the future to fabricate a transistor with a channel area so small that it will contain only one single doping atom. Not only will it be hard to predict the transistor's properties, but also it will be extremely difficult to fabricate transistors with reproducible properties. One may even settle a more provocative question: what happens if the technology scales down so much that there will be some transistors with no doping atoms in the channel area?

Currently, the transistor failure rate (TFR, defined as the percentage of malfunctioning transistors) is held at acceptable level, but inevitably nanometric technologies will be much more defective. It is almost impossible to warrant that all transistors will be fault-free in the future large ULSI including hundreds of billions transistors. One may of course perform testing after production and select only fault-free chips, but if the TFR is high, the production yield will be unacceptably low. There would have no sense to manufacture chips if for instance 98% of them could not be validated for deployment.

However, the studies conducted by the major chip manufacturers show the variability of transistor parameters is becoming in the future nanoscale technologies a problem more critical than the occurrence of defective transistors [2][3][4]. More precisely, most transistors will have different physical characteristics because of the variations in the process parameters such as:

- diffusion depths,
- density of impurities,
- resolution of the photolithographic process.

As a result, all (or most) transistors work, but a significant fraction of them will have reduced performance. The changes in the transistor parameters affected by the process variations may lead to:

- changes in the channel geometry,
- reduced I_D current,
- dispersion of threshold voltage $V_{T,}$
- increased gate capacitance.

(1) 45 nm is the MOSFET transistor channel length.

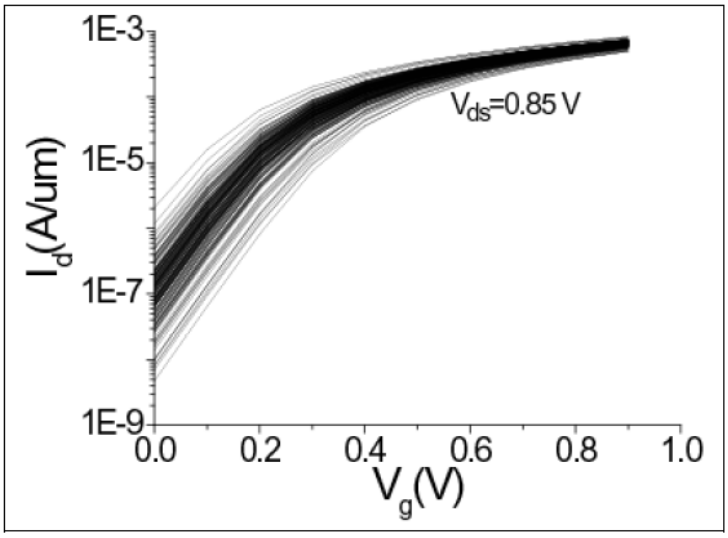


Fig. 2: Dispersion of transistor characteristics (source: [4])

Fig. 2 shows the estimation of the dispersion of the current/voltage characteristics for 200 transistors fabricated in a nanoscale technology. The transistors are of 30x30 nm size and are macroscopically identical. However, it can be easily seen that the differences in their current-voltage threshold are very significant. The variability will probably increase when considering transistors with reduced channel length.

Why are the fluctuations of intrinsic transistor parameters dangerous for the proper circuit operation? Let us reuse the example presented in [4]. In a 10-billion-transistor chip, if we assume some Gaussian distribution of transistor parameters, at least 20 transistors are expected to have a 6σ deviation in their parameters (σ being a standard deviation). If the threshold voltage standard deviation is in the range of 20–30 mV and if the supply voltage is 0.85 V, there will be at least 20 transistors with threshold voltage equal to zero or to half the supply voltage.

Moreover, fluctuations may not directly cause the complete failure of the transistor, but the dispersion of propagation times on critical paths. This in turn may lead to device malfunction, as most modern synchronous chips operate under very strict timing constraints. Consequently, the production yield will decrease, as not all chips will be able to operate at the nominal frequency. The manufacturer will be forced to decrease the nominal frequency (to increase the yield) or to sort the chips versus their top operation frequency.

The fundamental question is: what will be the TFR of future nanotechnology? Currently, the TFR of a CMOS transistor is quite low. According to Infineon, the typical TFR in the 0.35-μm CMOS process is of the order of $1\text{-}2\text{x}10^{-8}$ [6]. According to Nikolic et al [7], the TFR was in the range 10^{-7}-10^{-6} in 2002. We show below that when the TFR will reach approximately 10^{-5} (which is very likely to happen in the nanoscale technologies), we have to abandon the idea of producing fault-free chips, even when applying redundancy-based protection techniques in the physical layer.

The most popular protection technique, called N-modular redundancy (see for example [8]), is based on dividing the chip into "elementary" blocks and replicating each block N times. The voting technique is then used to provide fault-free operation even if some defective transistors

are present in some blocks. More precisely, even if a block provides a faulty output, the other corresponding blocks will “outvote” it (assuming of course that the majority of blocks and the voter are fault-free). Fig. 3 below displays a simple example of 3-modular redundancy.

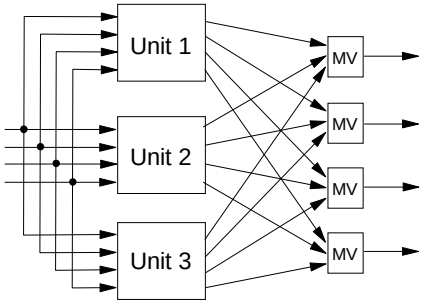


Fig. 3: Three modular redundancy example

The figure shows the example of such a redundant block, which includes majority-vote gates (MV). Each unit replicates a 4-bit combinatorial block. The relative overhead sacrificed for redundant elements is N-1, when the area needed for the voting circuit may be neglected.

Let us now consider an example of a multi-core chip. Fig. 4 shows the dependence between the transistor fault probability (TFP) and the failure probability of a core split in basic blocks of various sizes. It must be emphasized here that this figure is based on the pessimistic assumption that one defective transistor causes the failure of the entire block. It is assumed that each block is protected using triple modular redundancy (TMR). The core size ranges from one million transistors (see the 3 uppermost curves in the top left corner) to thousand transistors (see the 3 bottommost curves). For each core size, we consider several sizes of the basic building block. For instance, the 3 curves in the top left corner show the variation of the failure probability of the million-transistor core when we consider its decomposition in blocks with 1000, 500 or 250 transistors.

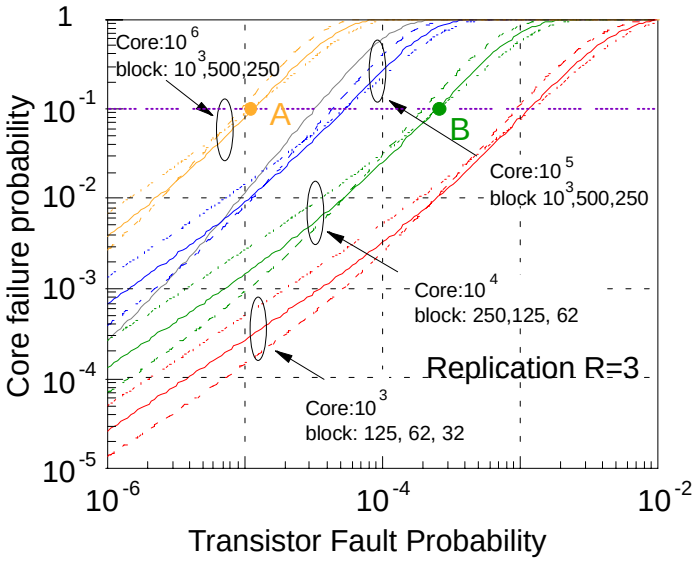


Fig. 4: Core failure probability versus transistor fault probability for chips protected with triple modular redundancy

Point A for example shows that the failure probability of a 1-million transistor core is approximately 10^{-1} when the transistor failure probability is 10^{-5}. Consequently, it means that even if we use heavy protection in the physical layer by using TMR, there will still exist 10% of defective cores in a multicore chip. Thus, the figure clearly shows that even triple modular redundancy technique is not enough to assure failure-free chip operation.

2.2.2 Transient faults

Transient (runtime) faults are perhaps even more menacing to the operation dependability of future chips than the permanent faults [9]. These errors are possibly caused by crosstalk, electromagnetic noise or cosmic radiation. In the future nanoscale chips operating at very high frequencies, the radiation noise from the gates will inexorably induce soft errors, especially when the gate operation voltage will be low. Moreover, additional errors may be caused by crosstalk. Already today, it is a great problem for large scale supercomputers [10]. As the dimensions and the supply voltage decrease, the same problems will appear in general-purpose processor chips. It may be safely stated that transient errors will constitute in the future a serious threat to the dependability of chip operation. A more detailed analysis of this problem can be found in section 5.3.2, page 105.

Therefore, we feel that assuming that nanochips will be fault-free would be dangerously misleading. Moreover, architecture designed with the assumption that there are no defects in the chip would be catastrophically inefficient or even inoperable if defects actually occur. Hence, in our work not only do we assume a defective technology, but also we consider nanochips to be massively defective. By massively defective we mean that a chip could comprise from several percent to as much as 40% of defective cores. One may wonder if indeed such defective technology would ever be implemented. The problem is that the core failure probability does not depend only on the technology and the transistor failure rate but also on the core complexity as shown in Fig. 4.

2.3 Complexity problem

Until the end of the 20th century, the efficiency ratio $perf / n_t$ (see section 2.1, page 5) was still at acceptable level. Designers succeeded in making significant improvements in the chip architecture, adding for instance more and more cache and, consequently, the processing power constantly increased. As shown in the figure below, in the eighties and in the beginning of the nineties, the size of processing cores and the size of caches evolved almost at the same rate. However, in the beginning of the 21th century, we may observe a significant change of trend: the cache memory size began to increase much faster than the processing core size.

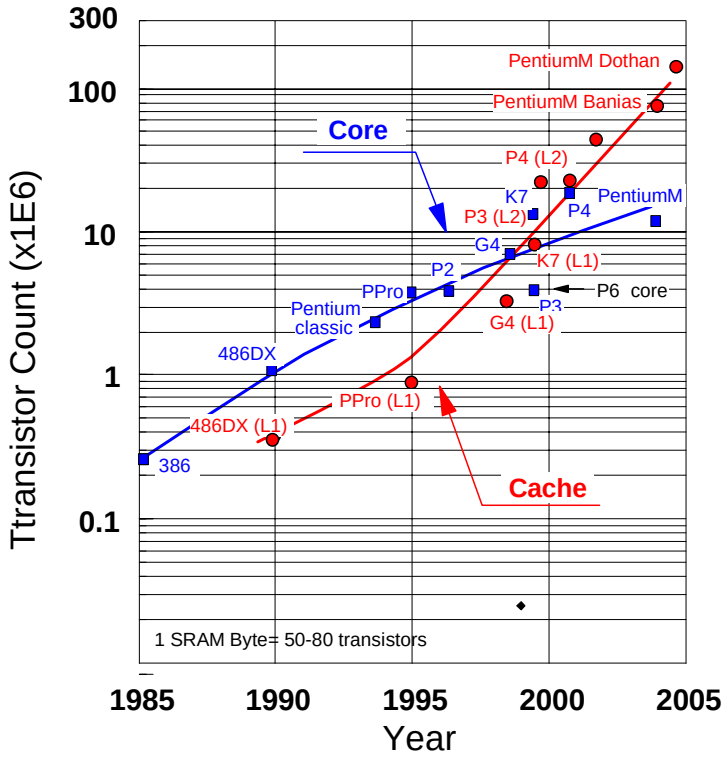


Fig. 5: Evolution of core and cache memory architectures

The main cause of this evolution is that there has been a kind of crisis in the architectural evolution of single-core architecture because the investigations to improve the core processing power do not offer significant performance gain. Why? Perhaps, the main reason is that designers are unable to “simply” increase the processing power of computers because of the large access latency to the main memory [11] which is in fact a longstanding architectural problem. Even the most innovative changes to the core micro-architecture produce little performance gain when the latency necessary to access data (or instructions) in the main memory does cease increasing in processor cycle units. Or in other words, designers could probably succeed in increasing the processor’s performance, but at unacceptable cost. For example, creating a new architectural solution which requires doubling the number of transistors on chip and provides only 5% of performance gain would clearly be inadvisable.

In this context, pure investigations on processor architectures were progressively reduced (or even stopped) at the end of the nineties, and the palliative solution has consisted in adding

more and more on-chip cache to mitigate the large off-chip memory access latency. It is exactly what is shown in Fig. 5. Of course, one may continue increasing the size of caches, maybe even introducing another cache level, but this solution leads to a dead end and becomes more and more inefficient due to the increase of the memory access latency.

A radical change of approach is needed, perhaps integrating the whole main memory on chip. The alternative solution which attracted many researchers recently is 3D stacked memory [12]. More precisely, the main memory is stacked on top of the processing chip and connected with it using through silicon vias (TSV) [13][14] which are in fact vertical interconnections between different chip layers. This solution provides very high bandwidth reaching dozens of GB/s and decreases the power consumption. See section 2.7.4, page 24 for a more detailed description of such a system.

2.4 Power consumption issue

The table below illustrates how the power consumption in processors has increased as processors became more and more complex in conjunction with the increase of the operating frequency.

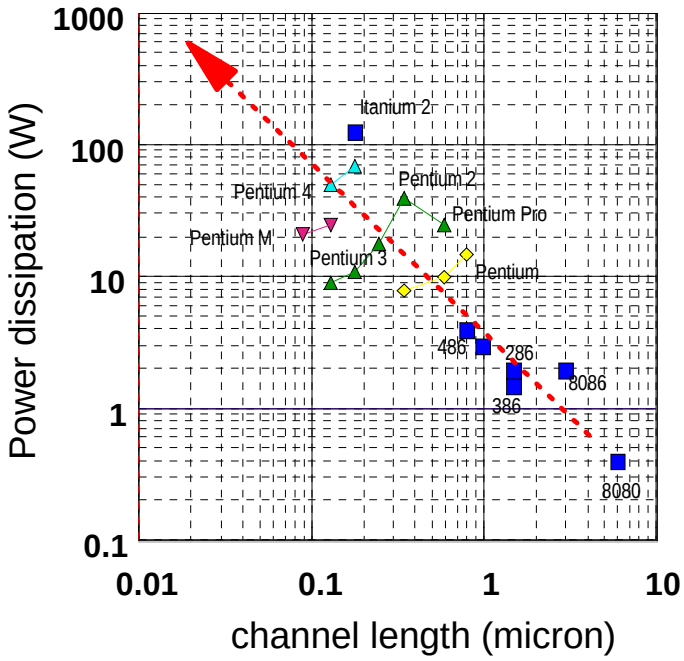


Fig. 6: Power dissipation in the mainstream processors

As we can see, the power consumption in processors in the recent years has become dangerously high. High power dissipation means high temperatures which, unless complex cooling techniques are applied, may simply destroy a chip during operation. As early as in 2003, the major processor manufacturers stopped the frequency race (partly because of power dissipation problems) and began to switch to multi-core architectures. The tables below present the power dissipation in the mainstream single core processors (Table 1) and multi-core processors (Table 2).

Processor type	Clock speed	Power consumption [W]
Pentium	75 MHz	8.0
Pentium	200 MHz	15.5
Pentium II	450 MHz	27.1
Pentium III	866 MHz	26.1
Athlon *Thunderbird*	1000 MHz	54.3
Pentium III (FC-PGA2)	1400 MHz	31.2
Athlon XP *Palomino* 1700+	1433 MHz	64.0
Athlon XP *Thoroughbred B* 2200+	1800 MHz	62.8
Athlon XP *Barton* 3200+	2250 MHz	76.8
Athlon 64 *Clawhammer* 3400+	2400 MHz	89
Pentium 4-C	2.4 GHz	67.6
Pentium 4 HT	3.06 GHz	81.8
Xeon	3.2 GHz	110
Pentium D960 B1 Stepping	3.6 GHz	130

Table 1. Power consumption in various single-core processors

Processor type	Clock speed	Power consumption [W]
Core Duo T2500	2.0 GHz	31
Core 2 Duo E6600	2.4 GHz	65
Dual Core Xeon 5060	3.2 GHz	95
Athlon 64 X2 4800+ *(Socket 939)*	2.4 GHz	110
Athlon 64 X2 4800+ *Brisbane (Socket AM2)*	2500 MHz	65
Athlon 64 X2 *Windsor* 6000+	3000 MHz	125
Core 2 Quad QX9650	3.0 GHz	130

Table 2. Power consumption in various multi-core processors

If we compare the two tables shown above we see that the power dissipated in quad core processors is at the same level as in single core processors but of course multicore processors have much higher processing power.

Why exactly multicore processors allow reducing power consumption? The power dissipation can be approximated by the following equation:

Eq. 1 $$P = P_{dyn} + P_{stat} = ACV^2f + VI_{leakage}$$

The first factor is of course the dynamic power which is lost due to charging/discharging chip capacities. The second factor is the static power lost because of leakage current. *C* represents the total capacitance of all gates and *A* is the percentage of switching gates.

Although the reduction of the transistor's channel dimensions provoked a considerable rise of the leakage current, the dynamic power is still the dominant factor in the above equation. So, in what follows, we will take only the dynamic power into account.

For the sake of our analysis, we assume that the operating frequency is proportional to the voltage. Consequently, we obtain:

Eq. 2 $$P \sim V^3$$

Bearing this in mind, let us now compare the power consumption and performance of single and dual-core processor. For the sake of analysis, we assume that the both cores in dual-core system correspond exactly to a single core processor. Assuming arbitrary units, the power consumption of a single core processor is 100% and its performance is equal to 100%. Following the analysis presented in [15], if we scale down in a dual core system the supply voltage to 85% of the value of the single core supply voltage. As a result we obtain:

- 15% reduction in frequency (as frequency is proportional to voltage)
- power consumption stays at the same level because there are two cores and each consumes 50% less power because of lower voltage (see Eq. 2)
- performance can be estimated at 180%

Thus, a significant performance gain is expected[(2)] without increasing the power consumption. This clearly proves that multicore architectures provide a good solution to increase processing power without largely increasing the power dissipation.

[(2)] The principal weakness of this approach occurs when executing applications which cannot be parallelized, as they must be executed by a single core, which of course has reduced processing power with respect to the single core chip.

2.5 Problems related to global clocking

Contemporary high-performance microprocessors are still being built as completely synchronous chips. One single clock is distributed to all parts of the chip. Chip designers must ensure that the clock signal arrives at the same time to all parts of the chip (or at least within acceptable time limits). If this constraint is not fulfilled, the chip will simply function incorrectly. Until now, chip manufacturers have succeeded in designing synchronous chips but at great cost. According to [16], even 40% of total power consumption in a chip is sacrificed to distribute the global clock signal!

Another dramatic problem appears as the technology scales down to nanometer dimensions and the chip is getting larger, due to the increase of latency of the interconnects. If the interconnect latency between any two parts of the chip becomes comparable with the clock cycle, the chip will be desynchronized and should be designed differently. Moreover, the transistor speed increases significantly faster than that of interconnects. Therefore, in the near future, the delay caused by onchip interconnects will be so high that it will limit the operating frequency of synchronous chips. Paradoxically, the chip operation frequency will not be limited by the transistor speed as today, but by the interconnect speed. Ultimately, the high-speed integrated circuits would have to be designed without global clocking.

What are then the possible alternatives to global clocking? Completely asynchronous systems [18][19] may be considered. Although we are perfectly capable of manufacturing completely asynchronous systems in theory, they are unlikely to be widely accepted in the near future. The problems of this solution are: lack of CAD tools, lack of experienced designers and very difficult testing. Recently, EPSON manufactured 8-bit asynchronous microprocessor [17]. However, most other manufacturers agree that such dramatic change in approach is not entirely sound.

GALS (Globally Asynchronous Locally Synchronous) systems [20][21][22] are a compromise between synchronous and asynchronous systems. The chip includes multiple synchronous domains, but the domains' clock signals are not synchronized (see Fig. 7). Within one domain, the designers are able to use all tools and solutions applicable to synchronous systems. The problems arise on the verge of two asynchronous domains. The communication between the synchronous parts is asynchronous and special solutions have to be applied to ensure the fault-free transfer of signals between the synchronous zones.

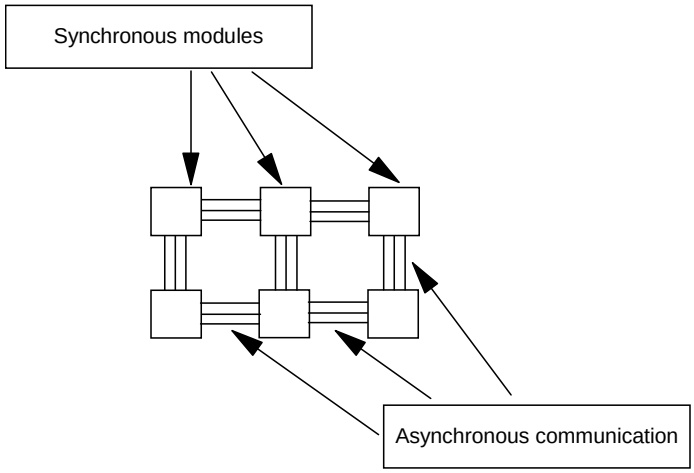


Fig. 7: An outline of a GALS system

The major problems of GALS systems are:

- Metastability:
  This problem has been known for many years [23] and it is especially dangerous in GALS systems. Let us now briefly outline the metastability problem. Metastability in flip-flops occurs when the value of input changes right before of right after the clock edge which triggers the flip-flop.

- Communication latency:
  Data transfer between two asynchronous domains is considerably longer than the transfer between two synchronous domains. The latency overhead is needed to ensure correct synchronization and fault-free transfer of data. Multiple solutions have been proposed to maximize transfer speed in GALS [25][26]. However, the fastest transfer speed in GALS systems is still slower than the transfer speed in synchronous mode.

A solution similar to GALS has been proposed by Intel [27]. Their 80-core chip includes 80 independent $3mm^2$ tiles. The “teraflop” chip has been designed to allow for the clock to arrive at individual cores out of phase. This mesochronous solution enables considerable power savings. Intel estimates that the power required for clock distribution to all of the tiles on the chip at 4 GHz is 2.2 W, which constitutes only 1.2% of total power consumption under load.

2.6 Memory-processor performance gap

The large access latency to the data in the DRAM memory is perhaps the most important and permanent problem which has influenced the design of computers. Whereas the processor's frequency increased in accordance with Moore's Law, the frequency of DRAMs increased much slower. Therefore, the difference of performances of the CPU and the main memory (known as memory gap) also steadily grows (see Fig. 8 and Fig. 9).

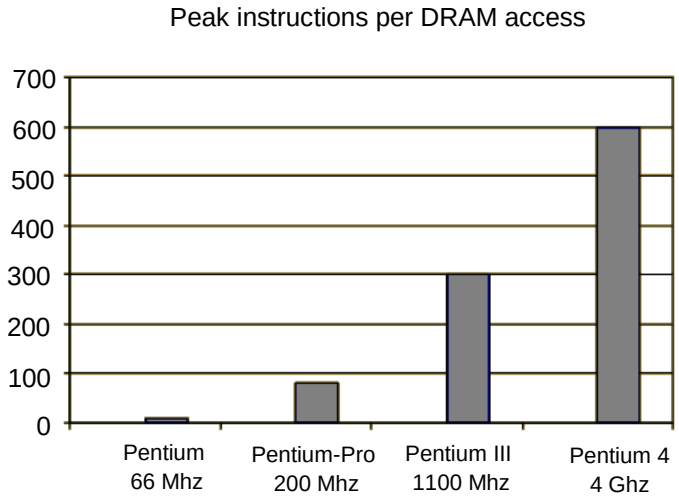


Fig. 8: Number of executed instructions per DRAM access (source: [15])

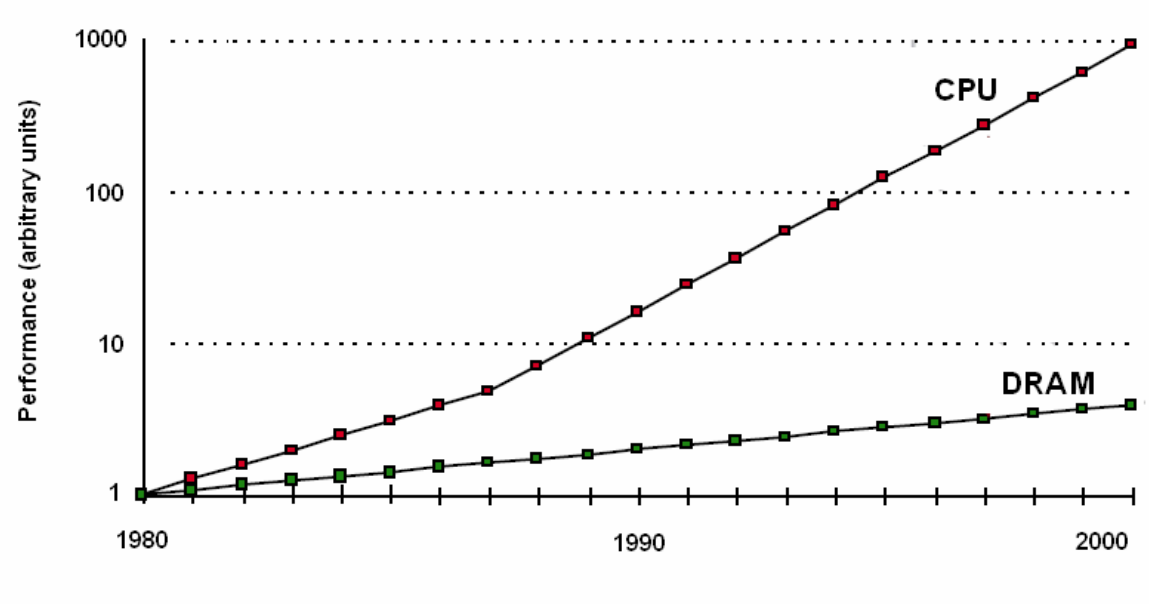


Fig. 9: CPU-memory gap (source: inst.eecs.berkeley.edu/~cs162/fa06/Lectures/lec13-cachetlb.pdf)

Why exactly is the memory gap dangerous? The memory and CPU are parts of the computer system. When the processor executes a program, it must read a sequence of instructions and operands in the memory. Consequently, the processor is pending (of course in a simple sequential operation) as long as it accesses, reads and retrieves this information in the memory. This is especially dramatic in modern high-performance systems where the access latency to the memory amounts to hundred processor cycles. In this context, most of the architectural evolutions implemented during the last decade, have consisted on speeding up the access to data (or the instructions) without reducing the latency to the memory. Concretely, the memory gap problem has been mitigated by adding one, two, three cache levels, by increasing the cache size, by adding a prefetch mechanism, a branch prediction, and using multithreading in multitask execution mode.

However, it may soon become apparent that a more radical change in approach is needed to “close” the memory gap. An example of such radical approach is integrating the main memory on chip [28]. A more conservative solution to reduce the access latency to the memory is to build 3D chips with memory stacked on top of the processor (see section 2.7.4).

2.7 Scalability in multiprocessor architectures

Parallelism has always been a dream of computer designers [29]. Execution of multiple tasks simultaneously is believed to be the solution which could considerably increase computing performance. Processors used pipelined architectures in the eighties. Then, in the next decade instruction-level parallelism (ILP) was developed, as a way to execute multiple instructions in the same processor cycle. Super scalar processors [30] boosted considerably the processing power. However, it seems that for the reason given in section 2.3, there is no or very little possibility to exploit ILP on a uniprocessor. Recently, thread level parallelism was introduced allowing a significant performance gain and this trend is supposed to continue as multi-core processors emerged on the market. Consequently, it seems that we enter an era of thread-level parallelism as multicore processors will allow simultaneous execution of multiple threads.

The detailed description of some well-known multiprocessor systems is presented in Appendix C. In what follows, we will mainly concentrate on the scalability problem in parallel architectures. A multi-core system is said scalable if it provides a constant improvement of the processing power as new cores are added to the system. In other words, adding new cores has no negative side effects on the communication bandwidth, the power consumption, etc. of the entire system and therefore, the performance rises almost linearly with the number of cores.

2.7.1 Small-scale multiprocessors

By small-scale multiprocessors, we mean architectures comprising typically up to 32 cores. Currently, the most popular architecture is based on the shared memory system with cores connected by a common bus. This solution, called Symmetrical Multiprocessors (SMP) is shown in Fig. 10 [31].

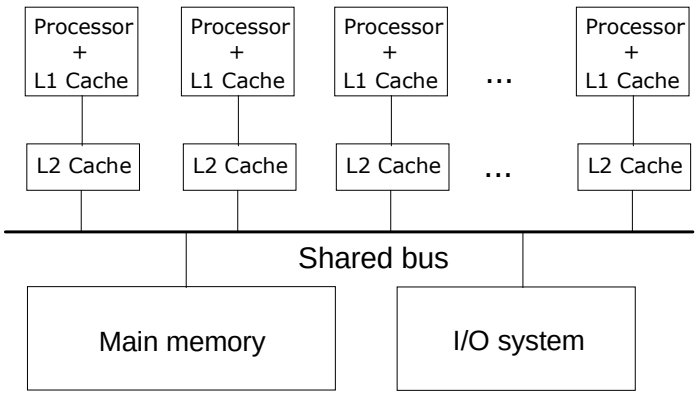


Fig. 10: Symmetric multiprocessor architecture

It is an example of UMA system (see Appendix C) and all cores are identical with the same access time to the main memory. SMP architecture is currently the most popular one. All cores can contact the shared memory via the shared address bus. The I/O subsystems are also shared by all cores. The management of task scheduling and execution is controlled by an

operating system which is in charge of the communication between processes. Note that this architecture provides high degree of interaction between processes contrarily to loosely-coupled multiprocessor systems like clusters.

The advantages of SMPs are the following:

- the existence of multiple cores transparent to the programmer,
- identical memory access times,
- easy inter-process communication,
- snoopy-based cache coherence,
- no need for routing protocol.

Note that this implicates that the failure of one core does not cause the failure of the entire system. For example, the Intel Core Solo processors [32] are simply dual core SMP with one faulty core.

SMP architecture has many important advantages. However, it is not scalable, just because the bus architecture is not scalable. More precisely, connecting more than several dozen cores to the same bus brings no performance gain because the heavy traffic on the bus prevents the efficient use of processors' resources and results in limited processing power. We may conclude that the solution based on symmetrical multiprocessors seems transitory and that the future belongs to scalable architectures, which have been extensively studied for decades in relation with the development of supercomputers.

2.7.2 Networks-on-chip

Networks-on-chip (NOC) is a new emerging architectural trend [33][34][35][36]. This paradigm has been proposed several years ago and its popularity systematically grows. It is based on an idea that a system-on-chip should be composed of independent instances (called resources) communicating with one another using scalable on-chip network. Memories, DSPs, processing cores, FPGAs, specialized ASIC circuits may all constitute resource blocks. The main goal of research in the NOC area is to create a standardized communication between resources in spite of their completely different nature. The achievement of this goal will allow a completely new approach to systems-on-chip as it will be possible to design general-purpose chips to perform specialized functions.

Fig. 11 shows a sample NOC. The connected tiles may perform a wide range of functions (see the caption of Fig. 11). The tiles are connected to switches (routers) which form a communication network. Here, it is a mesh network. The communication between tiles is standardized because each tile has a special network interface.

Thus, networks-on-chip are a very wide area of research because the functions of the resource blocks may be very different. Since the detailed description of NOC systems is beyond the scope of this thesis we will concentrate on the most interesting property of NOC which is the communication architecture.

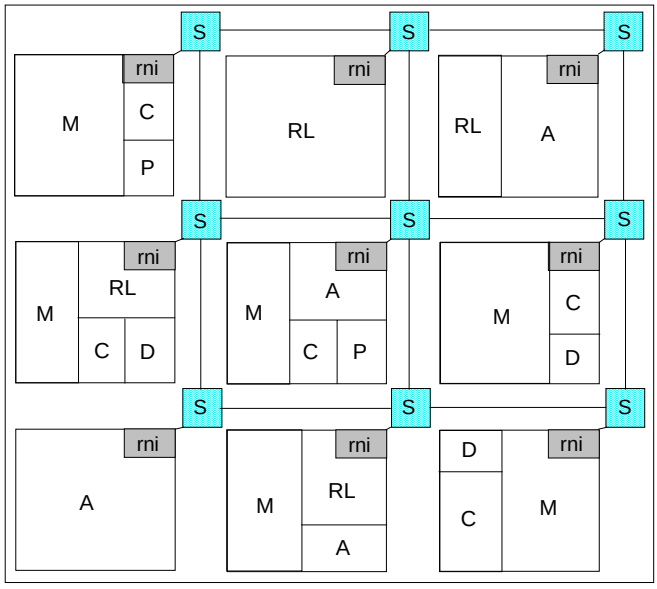


Fig. 11: Sample NOC [33], S=switch, rni= resource-network interface, P=processor core, C=cache, M=memory, D=DSP core, RL=reconfigurable logic, A=dedicated hardware

Nodes of the on-chip network are connected with one another by multiple interconnections which enable sending and receiving the data. The communication is enabled by switches (routers). Two adjacent switches are connected by n-wide bus where n is the number of data bits which may be transferred at the same time plus the number of bus control bits. The smallest transfer units are called *flow control units* or *flits*.

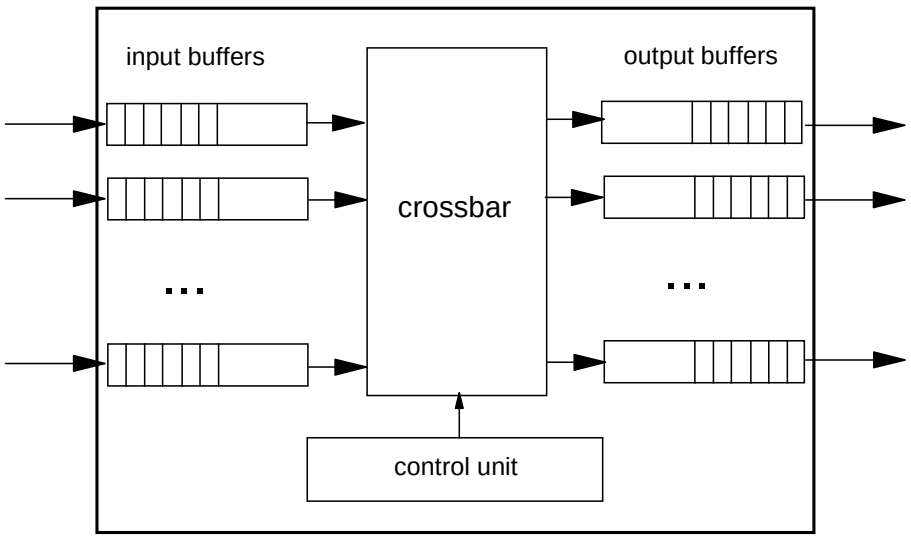


Fig. 12: Simplified switch architecture

Fig. 12 presents a simplified router architecture which is usually considered in networks-on-chip. The basic role of the router is to transfer packets from the input buffers to the output buffers according to some routing protocol. The fastest routers are designed by using crossbars, which are controlled by some combinatorial control circuit. Depending on the switching protocol, packets which are usually composed of multiple flits, may or may not be stored in internal router buffers before being sent out.

We distinguish:

- store-and-forward switching,
- cut-through switching,
- wormhole switching.

In the first method, a complete packet is stored in router's buffers before it is forwarded. In case of cut-through switching, the first flit of a packet is forwarded before the entire packet is received. This of course increases speed but reduces reliability. The wormhole switching is in fact a modification of cut-through switching: the most crucial difference is that in the former method the header flit reserves the route (the "wormhole") for the next flits. The advantage is that routers do not have buffers for entire packets but only for a few flits. Currently, the store-and-forward and wormhole switching seem to be the best solutions for NOCs, but the final choice will surely depend on the specific network architecture and design methodology. In this thesis, we deal mainly with message-level of abstraction; the store-and-forward switching is assumed, but most of our analysis is expandable to other switching methods.

2.7.3 Topology

What are then future possibilities for multiprocessor architectures? All proposed solutions are built using the same main principle: a scalable processor architecture is composed of multiple cores which, contrarily to the SMP solution, are connected by a scalable communication network.

Various scalable network topologies may be considered [37][38][39]. The choice depends on an enormous number of constraints and parameters from various domains. Very often, these constraints are contradictory which forces researchers to look for tradeoffs. Some sample parameters which largely depend on the network topology are: bandwidth, the length of interconnects, implementability on chip, connectivity, latency, etc. These parameters further affect other domains. For example, the longer the interconnects, the higher wire delay and the lower transfer speed. Ultimately, there is no clear answer which topology is the best. One topology will be more suitable for highly parallel networks whereas the other may be more suitable for network with small number of nodes. Before deciding on a network topology, one must have a fairly complete overview of the entire system. The most popular NOC topologies are presented in Fig. 13.

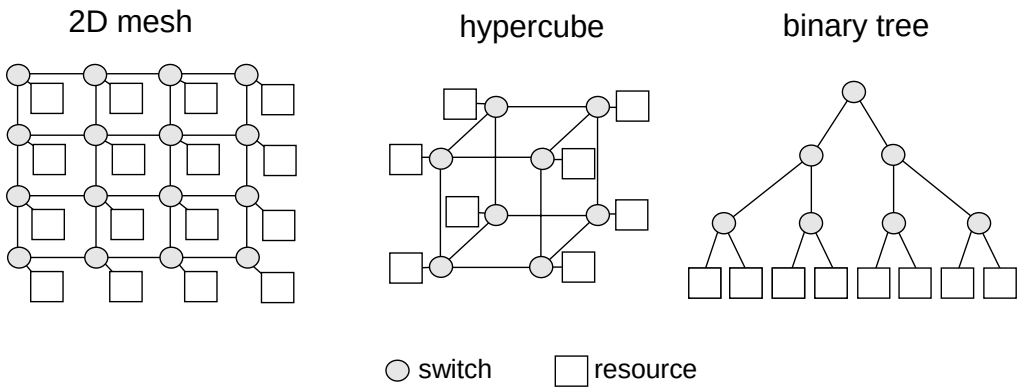


Fig. 13: Various NOC topologies

Many other topologies exist (rings, k-ary n-cubes, tori) but currently the most popular choice among the researchers is the 2D-mesh topology. Its major advantages are:

- The implementation on chip is very easy, as the topology is planar and has a very regular structure.
- Interconnects have the same length which makes crosstalk and delay more predictable.

The major drawback of this topology is its relatively smaller bandwidth as compared to other topologies. For example, a hypercube provides much higher connectivity (higher bandwidth) but is not planar and cannot be easily designed in 2D technology. It is of course possible, but it would require longer interconnects and irregular wiring. Other k-ary n-cubes [40] face the same problem. Tree topology risks running into congestion problem at root nodes. The torus, which is nothing else that a folded mesh, is an attractive alternative but it also requires at least some long interconnects to connect two opposite network borders.

In our approach, we do not make any specific assumptions on network topology. The majority of analysis is performed for 2D mesh, but some of the ideas are easily transferable to other topologies. In section 4.2, we actually compare various topologies in terms of resilience to the presence of manufacturing faults.

2.7.4 Chip multiprocessor

In the previous sections, we presented the idea of network-on-chip which is currently being heavily studied by many researchers. One may wonder: is that idea directly transferable to multicore systems? In other words, does a NOC system with only processing cores constitute a multicore processor? This question cannot, despite its simplicity, be easily answered. The NOC research focuses mainly on systems where all parts of a system are integrated in a single chip whereas multiprocessor systems must contain main memory which is usually off-chip.

The term *chip multiprocessor* (CMP) [41] is usually applied to a multi-core processor integrated on a single die. However, this term is not reduced to cores connected via common bus like symmetric multiprocessors. Most often, each core has a private L1 cache and a shared L2 cache which is virtually shared although it could be physically distributed.

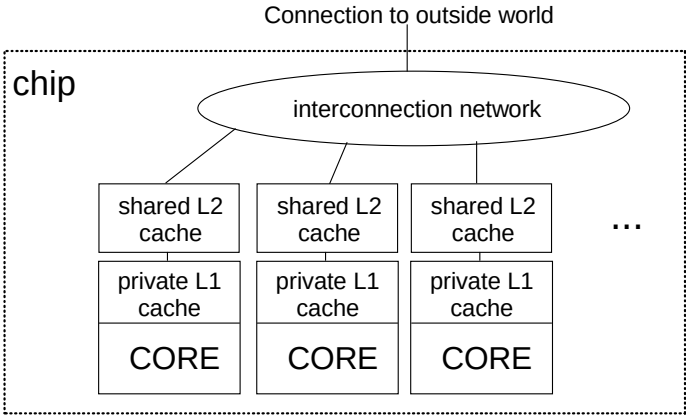


Fig. 14: Chip multiprocessor (CMP) architecture

How is the CPU-memory gap (see section 2.6) handled in CMP systems? Two most popular solutions have been proposed in the recent years. The first one is to integrate the main memory on chip. Although there are many researchers who support this idea [42], the amount of memory that may be integrated on chip may be too small to satisfy the needs of future computer systems. Additionally, mixing DRAM and core technologies will require a dramatic change in almost all aspects of chip manufacturing. Therefore, a second less revolutionary solution is currently being studied.

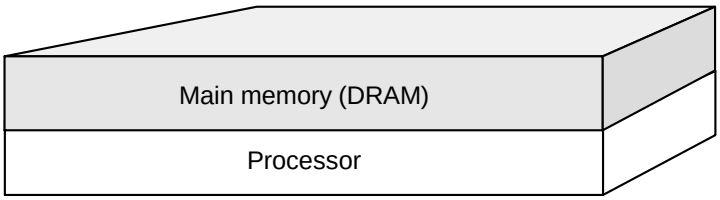


Fig. 15: 3D memory stacking

Recent technological advances in manufacturing vertical vias (or through-silicon vias) allow stacking the memory on top of the chip (see Fig. 15). Instead of connecting the chip with the main memory with an off-chip bus, the designers propose to connect the chip with memory using vertical vias. Of course, the major advantages of 3D stacked memory are extremely high bandwidth and lower latency than for off-chip DRAM. This revolutionary solution will, according to Intel, reduce considerably memory access time. Although it is still being analyzed, more and more manufacturers like for example Samsung adopt the 3D stacked memory idea [43].

3 Self-configuration mechanisms

3.1 General-purpose nanochip architecture

Our idea of general-purpose nanochip (GPN) is based on the reasons presented in the previous chapter. Let us summarize our basic assumptions:

- Only scalable architectures may provide a long-lasting solution for the need of constantly increasing performance.
- Nanoscale technologies will inevitably be massively defective. This conclusion also applies to the emerging technologies derived from molecular electronics studies.
- A high transistor fault rate together with large number of transistors on chip will make building fault-free chips virtually impossible.

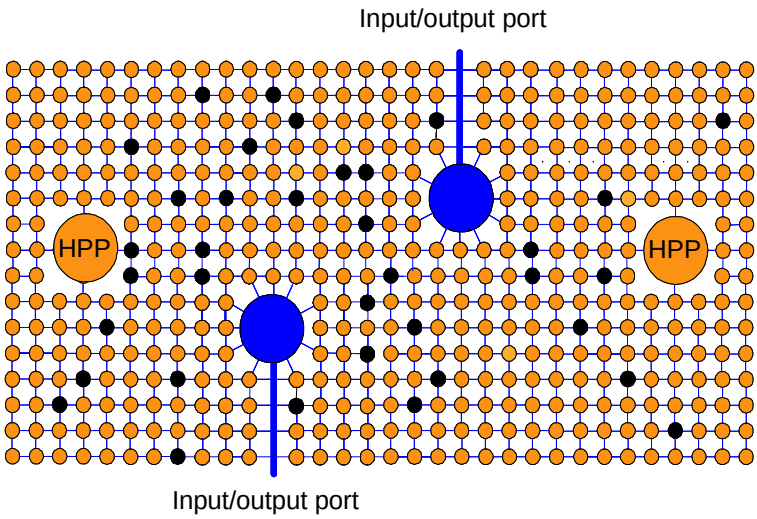
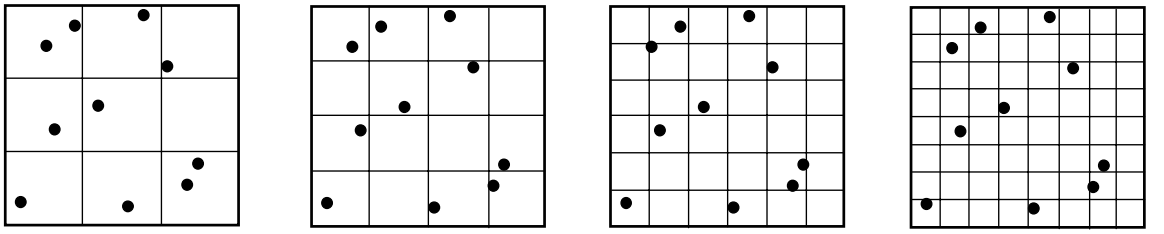


Fig. 16: Architecture of a general-purpose nanochip

Fig. 16 presents a general example of GPN architecture. The circles represent processing nodes which in the above example are organized as a 2D mesh network. Of course other scalable network topologies may be also used and will be discussed in section 4.2 page 52. As we consider a defective technology, blackened circles represent permanently faulty cores (i.e., cores including defective elements) which cannot take part in the processing. Note that the locations of these defective cores are *a priori* unknown. The number of faulty cores depends of course on the technology, but also on the core complexity, which is exemplified in Fig. 17. Here, we consider a chip fabricated with 10 defective transistors in random locations,

represented as black dots. The large square is the chip and smaller squares are on-chip processing cores.



9 cores,
8/9 defective (88,9%)

16 cores,
10/16 defective (62,5%)

36 cores,
9/36 defective (25%)

64 cores,
10/64 defective (15,6%)

Fig. 17: Multiplying cores on the chip as a solution to chip defects

Fig. 17 shows that when the chip is "partitioned" into more and more "tiles" and, for a constant number of defects on chip, the percentage of defective cores ("tiles") systematically decreases. This simple example shows the interest of having many cores on the chip from the point of view of protecting the chip against faults. However, decreasing the tile size comes to considering simpler cores and implies higher level of parallelization of the applications to maximize the processing power. What is the benefit of having many cores when programs are highly sequential and cannot be parallelized to make use of all the cores?

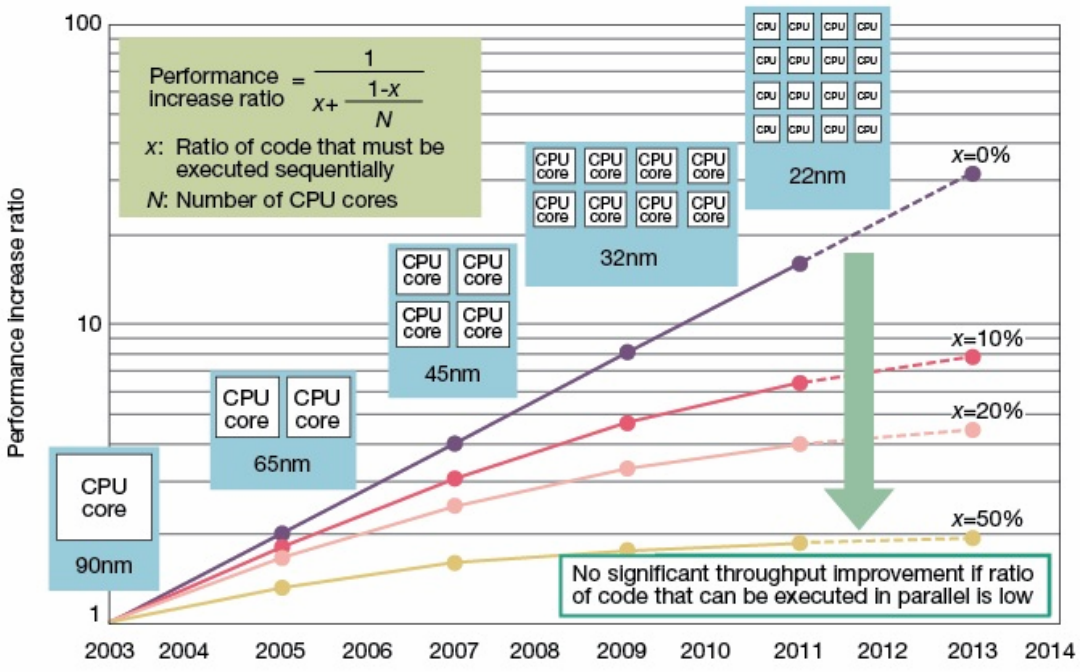


Fig. 18: Influence of application parallelization on chip performance [44]

Fig. 18 shows that increasing the number of cores does not provide a high performance increase, if the ratio of the code which must be executed sequentially is high. Consequently, we need new techniques which will allow higher code parallelization. Recently, Speculative

Multithreading (SpMT)[45][46] was proposed as a solution to this problem. Today's compilers always assume that two portions of code are dependent when they cannot determine that they are mutually independent. One can say that they perform "pessimistic" compilation. Contrarily, SpMT is an "optimistic" approach: when the compiler cannot determine if two portions of code are mutually dependent, it always assumes that they are independent and, therefore, may be parallelized! Consequently, the code can be parallelized in many tasks which may be executed simultaneously. During execution, the dependence between two code fragments is verified and if dependence is found, the dependent fragment of code must be re-executed and previously obtained results must be discarded. Simulations have shown that the higher parallelization SpMT provides about 2 times faster execution than the "pessimistic" compiler [47] and, therefore, it is very suitable for future multicore architectures.

We consider the communication architecture to be a sort of network-on-chip (see section 2.7.2, page 21) in which every node is composed of a processing core and a router (see Fig. 19).

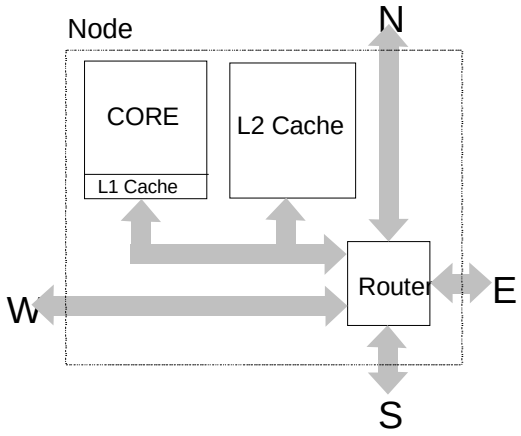


Fig. 19: Node architecture for mesh network

Thus, a router is connected to several neighbors (depending on the interconnect topology, see for instance Fig. 30, page 53) and to the core in its own node. Each node is a fully synchronous island but the clocks of different nodes are not synchronized. Consequently, a GPN is an example of GALS system (see section 2.5), as there is no global clock signal. It must be also emphasized that not all cores must be necessarily identical. In other words, there may exist in the network several cores with different processing powers. In Fig. 16, the larger circles marked HPP represent more complex cores with higher processing power. In this thesis most of the analysis is conducted under the assumption that the network is homogenous (i.e., that all cores have the same size and complexity). However, in a general approach one should consider heterogeneous networks. For example, in a massively parallel system with simple cores, several complex cores might be implemented to execute weakly parallelizable applications. The disadvantage of this solution is the disruption of network regularity. Moreover, the probability that a core is defective increases proportionally with the core complexity for a given transistor failure rate (TFR).

Two large circles with off chip connections represent in Fig. 16 two Input/Output Ports (IOP). An IOP is a special circuit which is responsible for interfacing the communications with the outside world, like the main memory or the I/O subsystem. Note that optimizing the number of IOPs in the network is a complex issue. On one hand, the number of IOPs must be kept small, because if it is high, it is harder to guarantee that all of them would work correctly which is necessary for the correct operation of the entire chip. On the other hand, a low number of IOPs, especially in a

network with large number of cores, may lead to a communication bottleneck with the outside world, as all communication traffic from all cores must be handled by one IOP. For a more detailed discussion and results see section 4.4.1, page 79. It is worth to note that to avoid any communication bottleneck, the IOPs connectivity (i.e. the number of connected adjacent neighbours) may be higher than the connectivity of other cores in the grid. For example, in Fig. 16, both IOPs have connectivity 11 whereas other cores have only 4 adjacent neighbours. Hence, another disadvantage of this solution is the disruption of network regularity.

3.1.1 Memory architecture

In order to make our work as general as possible, we tried to make no specific assumptions on the memory system. However, some assumptions are necessary to correctly design the entire system. In this thesis, we consider that the multi-core GPN is connected to a memory via one or multiple Input/Output Ports. The main memory may be located off-chip or stacked on chip, as presented in section 2.7.4, page 24. We strongly believe in the latter case, as the off-chip memory latency is a prohibitive factor to the performance of the entire system. In other words, the chip resources would not be effectively used and the processor would be slowed down by the memory access latency. It must be emphasized, however, that some of the ideas presented in this thesis are independent of the memory architecture. Consequently, they remain valid for other architectures like systems with the memory integrated on chip.

Cache coherence in chip multiprocessors is currently a very thoroughly investigated topic (see for example [48]). Clearly, snoopy-based cache coherence protocols are only applicable in common bus architectures because they are based on the idea that all cores trace the memory bus traffic. An interesting alternative for non-bus architectures is the directory-based cache coherence. It requires maintaining one or multiple directories which store the state of every cache copy of a memory block. When the block is modified, only the caches that store the same block are informed which eliminates broadcasting. Several other cache coherence protocols exist which try to use the advantages of both snoopy-based and directory-based protocols. Since cache coherence is a very wide area of research, we only mention an outline of cache coherence protocol in section 5.3.1, page 104. It is the slightly modified directory-based protocol, since our protocol must also work correctly in the presence of defective cores.

3.1.2 Self-configuration methodology

Perhaps a most important idea in this thesis is accepting the fabrication of massively defective chips. Then, the following question arises: how can we make defective chips work? Several approaches have been proposed in the literature, mostly based on using spare elements and physical reconfiguration [50]. In this thesis, we consider rather a “logical” self-configuration of the chip. Since we consider dozens or even hundreds of cores on a single chip, we may imagine a chip which works properly in spite of containing a fraction of defective cores. It seems possible provided that we succeed in isolating the defective cores and in enabling fault-free execution of tasks on non-defective cores.

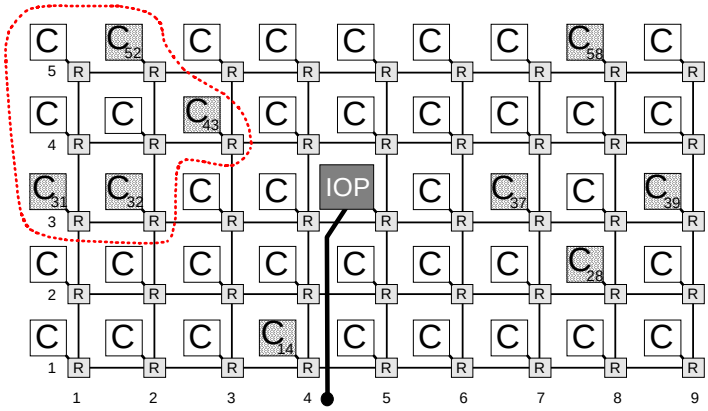


Fig. 20: Sample defective GPN with mesh topology

Fig. 20 shows a GPN with about 20% of defective cores in *a priori* unknown positions, represented as grey squares with letter C. In the following chapter, we shall thoroughly explain the methods and techniques which allow for fault-free operation of the defective chips. In short, the chip should become as autonomous and independent as possible. External interventions should be completely avoided or reduced to the minimum. To achieve that, a chip should be able to self-configure during a preliminary phase which will subsequently guarantee fault-free operation. Of course, special self-configuring mechanisms should be also implemented at runtime to control the chip operation.

Our main ideas for self-configuration include:

- Self-diagnosis of cores with mutual tests.
- Self-shutdown of inactive processors.
- Self-configuration of communication routes.
- Self-adaptive task allocation and redundant execution at runtime.

Thus, this work may be viewed as a presentation of a self-configuration methodology which allows tolerating both permanent and transient faults in massively defective multicore chips.

One of the most important questions which we try to answer in this thesis is: how many defective cores can we tolerate in a chip? Therefore, in our analysis, we will usually consider that the core failure probability extends on a wide range and that there may be even as much as 40% of defective cores in a chip. One may wonder whether it is realistic to consider that the node failure probability $p_{f,N}$ could reach 0.4, i.e., that 40% of the nodes could be defective. In fact, the crucial problem is that the node failure probability $p_{f,N}$ does not only depend on the reliability of the nanoscale technology. The basic question of deciding whether 10, 20 or 40% of nodes could be defective is also a design problem as the fraction of defective nodes depends on the node complexity (see Fig. 4, page 9). On one hand, it is attractive to increase the node complexity because it increases the node processing power and limits the need of heavy application parallelization (see section 3.1). However, on the other hand, increasing the node complexity raises the failure probability $p_{f,N}$ and, consequently, increases the fraction of defective nodes. Therefore, one must consider the node failure probability $p_{f,N}$ as a parameter which must be chosen as a trade-off between the desirable maximum complexity of the core (the more complex the better) and the protection overhead necessary to maintain $p_{f,N}$ below the values tolerable by the fault tolerance methodology at the architectural level which we describe in this thesis.

3.2 Self-diagnosis through mutual tests

The idea of self-test is based on the reasonable observation that external testing of very complex nanochip could become very difficult if not impossible. More precisely, self-test is based on the idea that all testing actions are performed by the chip itself in order to identify faulty cores. Additionally, the testing mechanisms must be associated with some configuration actions so that defective cores do not interfere with the operation of the good ones. This section is organized as follows: first, we present the advantages and drawbacks of various chip-testing methods; second we analyze the possible defect locations, and finally we propose an efficient method for defect discovery and isolation of defective cores.

3.2.1 Comparison of testing methods

Testing integrated circuits can be described as a process which allows detecting faults in the fabricated chip. A fault is usually discovered by a mismatch in the circuit's response to test vectors with the expected response. We distinguish:

- Manufacturing testing, which is performed at the end of the manufacturing process.
- On-line testing (periodic testing) which is performed during the normal operation of the tested chip.

Moreover, on-line testing is necessary to discover permanent faults which may be caused by aging or external factors (electromagnetic field, cosmic radiation, excessive temperature etc.).

The first part of the testing process is the generation of testing patterns which, when applied to the tested circuit, should have the highest possible fault coverage. Of course, the acceptable fault coverage level may vary, depending on the circuit complexity and application. Increasing fault coverage may be done by adding more and more test vectors but a point is quickly reached where even the large increase of the number of test vectors will only provide a very small improvement in fault coverage. Obviously, beyond this point new approaches are needed to achieve the desired level of fault coverage.

In this work, we discuss the testing of powerful processors, which constitute a special case of tested circuits. Due to the fact that they operate at extremely high frequencies and under heavy workload, modern processors have a highly optimized architecture. Every change in the design would disturb the architecture optimization and may lead to significant performance degradation or increased power consumption. Moreover, any additional hardware disrupts the optimized design of the chip area. Thus, processor might have to be re-designed while taking into account the additional testing circuit.

Functional testing [49] relies on the verification of all circuit's functions rather than the correctness of the circuit. Contrarily, structural testing uses a structural fault model so that a gate-level model is needed. Therefore, structural testing usually provides higher fault coverage than functional testing. It must be emphasized however, that a unique feature of general-purpose processors is the presence of an instruction set which *de facto* defines all processor's functions. Therefore, functional testing may be especially suitable for testing

processing cores. More precisely, the functional testing of the processor may use the instruction set to create test patterns which can be later applied to test the processor.

Testing patterns for a processor may be:

- Pseudo-random, which means that pseudo-random operands and pseudo-random instructions sequences are generated and applied to the processor.
- Deterministic, which means that pre-calculated test sequences are used.

Either solution has advantages and disadvantages. While pseudo-random testing allows fast generation of test patterns which needs no gate-level model, it suffers from low fault coverage. Moreover, a large number of test vectors is necessary. On the other hand, deterministic testing achieves high fault coverage with shorter test sequences, but needs gate-level details and test vector generation is much more difficult.

Testing of extremely complex chips is already today a very complex and time-consuming task. In the future, as the number of transistors on chip increases, new approaches and methods will have to be applied to ensure the efficient, fast and reliable testing. Let us now describe briefly the current methods of chip testing, their advantages and disadvantages and their possible applications in the future nanoscale technologies.

External scan-based testing

External scan-based testing [51] is usually performed using some Automatic Test Equipment (ATE) which is simply an external tester. Of course, to provide the external access to the circuit, additional hardware on chip must be implemented. The basic idea is that in the test mode (scan mode), all flip-flops in the circuit are linked so that the external tester has an access to the flip-flop chain. The detailed description of this method is beyond the scope of this thesis, we should concentrate instead on the advantages and limitations of this technique. The main advantages include high fault coverage and almost no manual intervention.

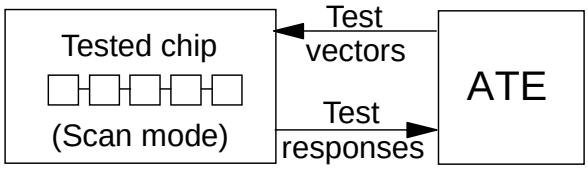


Fig. 21: External scan-based testing

However, this method is not entirely suitable for testing complex nanochips due to following drawbacks:

- Requirement for additional hardware
  Additional hardware consists mostly of multiplexers needed for different circuit configurations (for the test mode and for the normal operating mode). Obviously, this increases significantly the area on chip. Moreover, the additional circuit elements change the delays on the circuit paths. Since the additional elements are inevitably present on the critical paths, this may cause unacceptable performance degradation, especially if we consider nanochips operating at high frequencies.

- Testing duration
  Scan-based techniques are, because of their nature, very time-consuming. The larger the flip-flop chains, the more time is needed to insert input vectors in the chip and then collect output vectors from the chip. In complex circuits this time may already become a limitation and when nanochips with hundreds billions of transistors are considered, the testing duration will become prohibitively large.

- Need of sophisticated ATE tool
  The more complex the tested circuit, the more complex the external tester should be. For example, the memory on tester should be large enough to store all test patterns and test responses. Moreover, the gap between circuit operating frequency and tester operating frequency should be as small as possible to guarantee the test quality which significantly increases the cost of the tester.

- Undetected delay faults
  Since the ATE operating frequency is always lower than the nominal chip operating speed, some timing faults may not be detected yet they may occur during chip operation.

- Overtesting
  Since the circuit configuration in a test mode is different from the circuit configuration in its normal operation mode, some faults may be detected during test which would never affect standard chip operation. In other words, a chip which would function properly could be sometimes discarded.

Built-in self-testing

Since external testing cannot overcome the limitations listed in the previous paragraph, built-in self-testing (BIST) [52][53] has been proposed. The idea of self-testing chip has been proposed many years ago. It is based on integrating the entire testing circuit into the chip. Pattern generation, testing and response collection are all performed inside the chip. The additional hardware should also be able to test itself to eliminate the possibility of faults in the testing circuit.

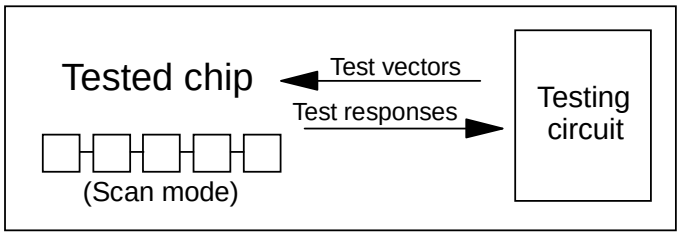


Fig. 22: Built-In Self-Testing

BIST solves partially the issues of external testing which we enumerated in the previous chapter because:

- It eliminates the need for testing equipment.
- It provides a much better access to circuit's nodes.
- It usually provides higher fault coverage.

- It enables at-speed testing (chip is tested using real operating frequency).
- It eliminates tester inaccuracies.
- It enables on-line testing (chip may be tested periodically during normal operation).
- It reduces testing time.

Still, these advantages come with a price which is, of course, a considerable hardware overhead because the device under test is put in the test mode (scan mode). Note that even more additional hardware is needed than in case of external testing as the testing circuit is integrated on chip. BIST may use deterministic or pseudo-random test pattern generation. In the former case, a considerable amount of additional on-chip memory must be provided to store the test vectors and responses. In the latter case, pattern generator and analyzer must also be integrated on chip.

BIST is very suitable for testing regular structures like memories. In fact, memory testing techniques based on built-in self-test and repair (BISTR) [54] has been used with success because it achieves high fault coverage at reasonable cost. However, for powerful high-end processors, the most important factor is not the additional hardware itself, but the impact of additional circuits on processor's performance. Of course, we may imagine a situation when the testing circuit has zero impact on chip performance because the critical paths are not changed, but in reality there will be always performance degradation when the testing circuit is added to the chip.

Software-based self-testing

Software-based self-testing (SBST) [55][56][57][58] is a relatively new method based on functional testing and is especially suitable for processor testing. Its major advantage is the fact that a processor itself takes over the role of the testing circuit! All testing phases, including test pattern generation, the test itself and the collection of responses are performed by the processor itself. In other words, the processor is tested by executing a special testing program.

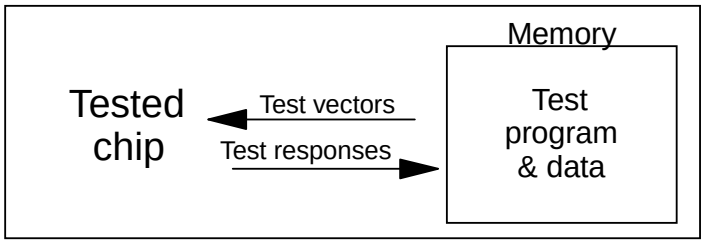


Fig. 23: Software-based self-testing

Note that no additional hardware is needed because the chip operates in its normal mode. The test program may use the standard processor instruction set. The program constitutes in fact a sequence of test patterns which are fetched and executed. The processor's responses are saved in the memory either as raw words or as CRC-based signatures. Perhaps the most important question in this testing method is whether the test program should be integrated on the chip or whether it should be loaded into memory by an external tester. Both solutions are applicable and the choice largely depends on the program size.

Let us now outline the advantages of SBST:

- SBST do not suffer from the performance degradation due to hardware overhead, as the original circuit is not modified (non-intrusive method).
- No chip area is sacrificed for testing circuit.
- Chip is tested at its nominal frequency (at-speed testing) which allows discovering the faults undetectable at lower frequencies.
- If test program has to be loaded into the memory, there is no need for expensive, complex tool; a simple tester is good enough which significantly reduces the cost of testing.
- No overtesting problem: only the faults that indeed prohibit correct processor operation are detected.
- Significant flexibility, as the testing program may be easily changed if the processor architecture evolves.

In what follows, we will consider SBST of chips with defective cores. One drawback of this method is its relatively lower fault coverage in comparison with BIST. In fact, it must be emphasized here that no method can guarantee 100% fault coverage. Therefore, we suggest implementing a mechanism which enables discovering faulty cores, undetected during testing phase. The method is related to redundant execution mechanism and is described in section 5.3.3, page 105.

3.2.2 Defect locations

A node typically comprises one core, a router and a memory (see Fig. 19 page 29). Each of these components can be defective.

Memory protection

All memories are built from highly regular structures, and the protection techniques always take advantage of this property. Memories may be affected by cosmic radiation which generates soft errors or by permanent fabrication faults. The protection against manufacturing faults is achieved by adding spare units [59]. Regarding soft errors, the abundant literature shows that the implementation of memory protection techniques makes the impact of soft errors almost negligible in the existing technologies [60]. The memory protection techniques allow achieving memory production yield close to 100%. However, the increase of manufacturing faults in future technologies and the increase of the sensitivity to cosmic radiations raises the question of the cost of the protection in terms of material overhead. Physical protection of memory may be performed by:

- Implementing error detection and correction.
- Hardening the transistors of the SRAM cell.
- Increasing the size of the critical-node capacitor of the bistable circuit.

Core protection

Protecting a processing core is much more difficult than protecting a memory because a core does not have a regular structure. Even with enormous redundancy, there will be considerable percentage of defective cores on chip (see Fig. 4 page 9). A method which tries to cope with the defective nodes consists in adding spare node cells to the chip (see for example [61][62][63]). The defective nodes are then identified and replaced. We believe, however, that it is not suitable for massively defective architectures. First, spare cells may also be defective. Second, reconfiguring the network to replace a defective cell is a very difficult task, especially if the number of needed replacements is very high. Therefore, we chose another approach which tolerates a fraction of defective nodes in the network without any physical reconfiguration.

Router protection

The router is a unit smaller than the core. Its size depends strongly on the network bandwidth (the more bits in the physical word, the larger the buffers) and on the complexity of routing algorithms. However, even in case of very high bandwidth, the router should be typically several times smaller than the core (in [27] the crossbar router has the size of the single floating point unit). Thus, defects in routers should be relatively rare.
However, it must be emphasized that the occurrence of defects in a router means that neither the memory nor the processor in the corresponding node can be accessed. In other words, a router defect implies that the whole node becomes in fact inaccessible. There are two approaches to solve this problem: One can either try to protect the routers at any cost by using redundancy or by implementing a router testing technique. Since the router size is not known, it is difficult to calculate the number of redundant transistors that would have to be sacrificed to protect the router. In the second approach that we describe below, the entire node is tested with no dedicated test to detect faults in the router.

3.2.3 Mutual tests

First, we want to emphasize that the discovery of defective cores is performed only once at a system startup. The results of the initial test are maintained in cores’ non-volatile memory and saved when the power supply is turned off.

Testing multi-core chips constitutes a very difficult task. First of all, the chips are very complex so that a lot of hardware overhead would be needed in case of external-based testing or hardware-based self-testing (because in these methods testing circuits are integrated on chip, see chapter 3.2.1). Second, since we consider defective chips, the defects may also be present in these testing circuits, making the testing process even more problematical. Therefore, a software-based self-testing which does not require on-chip testing circuits seems a viable testing technique for complex multicore chips. We suppose that each core has a special non-volatile memory in which a testing program is stored. The execution of this program allows for testing of all core functional units. Alternatively, test vectors may be loaded by a simple external tester or generated on chip by a special circuit. However, the problem with on-chip vector generation solution is that we must guarantee that the test vector generating circuit is free of defects.

The simplest and intuitive method to identify defective nodes is node self-test. Every node should have a non-volatile memory with test and result vectors. The number of test vectors should be defined as resulting from a tradeoff between maximizing the fault coverage and reducing the needed memory space. Test vectors are successively loaded into the processor and the results are compared with the result vectors. However, an obvious problem appears when a defective core tests itself: it can behave unpredictably, and, therefore, produce false diagnosis. For example, a faulty core can decide that it is, in fact, not faulty. To avoid such ambiguous situations, a special testing circuit would be required to perform the testing operations. This testing circuit would have to be very carefully protected to ensure that no false decisions are produced at the end of testing. Since this method is the simplest, it should be not completely rejected, but we are convinced that the technique described below is more suitable for massively defective multicore chips.

To avoid the need of additional testing circuits, we suggest the following method proposed in 1994 by Laforge, Huang, and Agarwal [64]. In this work, a mathematical model for testing blocks on a semiconductor wafer was presented. It was originally designed to be used with BIST and focused primarily on mathematical aspects of such a technique. However, we will show that after some modifications, this method is applicable to SBST of multi-core processors. Let us now present the testing mechanism.

This technique uses the mutual node test of adjacent cores to identify faulty nodes. More precisely, the mutual test starts just like the self-test: every core executes the testing program. The crucial difference between self-test and mutual test is that the node compares its result vectors with those obtained from its neighboring nodes. The table below shows the possible outcomes of such a test.

Node 1	Node 2	Node 1 decides that Node 2 is:	Node 2 decides that Node 1 is:
GOOD	GOOD	GOOD	GOOD
GOOD	FAULTY	FAULTY	UNKNOWN
FAULTY	GOOD	UNKNOWN	FAULTY
FAULTY	FAULTY	UNKNOWN	UNKNOWN

Table 3. Four possible outcomes of a mutual node-node test

As we can see, a good node will always produce a good judgment about its neighbor state but a defective node may produce a false result.

3.2.4 Isolation of defective nodes and links

Each node performs the following action: it permanently inhibits both incoming and outgoing communication with the node that it has diagnosed as faulty. In other words, all links to the neighbours diagnosed as faulty are logically disconnected. This method is illustrated in Fig. 24. Note that we assume that a faulty node may or may not disconnect the link. Let us now analyze the outcome of the test using this figure. Crosses symbolize logical disconnection of the link and question marks mean that the state of the link is unpredictable.

We may conclude that:

- Good nodes will always remain connected (situation a).
- Good nodes will be always disconnected from faulty nodes (situations b and c).
- The state of the links between two faulty nodes is unpredictable, two faulty nodes may (but do not have to) be interconnected (situation d).

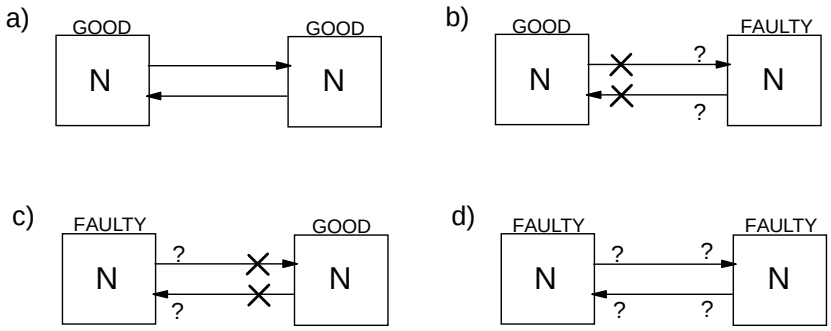


Fig. 24: Four possible outcomes of a mutual test

Thus, **the isolation mechanism relies exclusively on the actions of good cores without making any assumptions on the decisions of faulty cores**. **The only requirement of the method is that, in case of mismatch, a good core stops communicating with the neighbor which generated different test results**.

Now, what is the consequence of this isolation mechanism on a multicore array from a global point of view? The state of a sample network after mutual tests have been performed is depicted in Fig. 25.

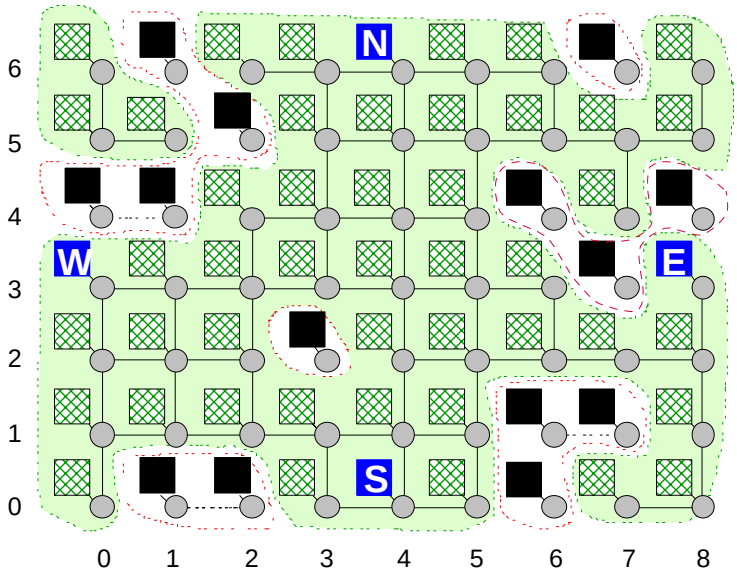


Fig. 25: Example of self partitioning of cores in a 2-D grid architecture resulting from the mutual-diagnosis mechanism. The grid includes a 4-IOP and 14 defective cores.
good core, faulty core, router, IOP.

The figure displays the partitioning of chip resulting from mutual test performed by all nodes. The chip is a 7x9 mesh grid with 14 defective nodes (black squares) and four IOPs represented by squares with letters N, E, S, W. We included multiple IOPs to show that our method remains valid even for multiport architectures. The core and the router are represented by a square and a circle, respectively. The solid lines between routers represent interconnects. Disabled links have not been drawn. The clusters of faulty cores are surrounded by dotted curves. Let N_{ij} be a node located at coordinates (i,j). Fig. 25 shows that a network has been split into several simple-connected zones (SCZ). All nodes in a SCZ are of the same type: either good or defective. Unfortunately, we do not know *a priori* which zone is good and which is defective! Therefore, we propose to remove this ambiguity by executing an external test of IOPs.

Needless to say, if one of the IOPs is faulty, the chip cannot work properly, as it does not have all necessary I/O connections. If an IOP is good, all nodes belonging to the same zone can also be identified as good. Usually in the chip there will be one large zone which comprises all IOPs for the fraction of defective cores that we consider, typically below 40%. This zone is typically the largest zone in the network and is therefore called the main SCZ. SCZs around ports include all nodes which can be later used to execute tasks. Inside each SCZ, there may exist defective nodes (for example $N_{32,}$ N_{76} in Fig. 25) or groups of defective nodes (N_{10}, N_{20}). Note that neighboring faulty nodes can form a group (like already mentioned N_{10} and N_{20}) or be disconnected ($N_{61,}$ N_{60}). However, the states of the links between defective nodes do not have any influence on the network operation because they will not be used in the subsequent operation. It should be also emphasized that all good nodes do not belong necessarily to the SCZ around IOPs. Some good nodes may be isolated from the IOPs by a group of defective nodes. For example the good subgroup N_{06}, N_{05}, N_{15} in Fig. 25, is lost for processing - they are not accessible from any IOP because they are blocked by the faulty nodes N_{04}, $N_{14,}$ N_{25} and N_{16}.

Let us now stress the following important advantages of our approach:

- The disconnection of the nodes is entirely logical, not physical and we do not consider replacing faulty nodes with spare ones. Thus, there is no physical reconfiguration of the chip as considered in [50][65][66].

- Mutual tests have yet another advantage over simple node self-test: the comparison of results require that the routers and the link between two neighbors work correctly. More precisely, the comparison of results may be configured in such a way that it diagnoses routers and interconnections between them. The technique described in section 3.2.3 may be used to discover faulty links as well as faulty cores and routers.

- The partitioning mechanism which splits the network into zones of either good or defective cores is performed by good cores only. The behavior of defective nodes has no effect on partitioning process and therefore, the entire testing method is independent from the operation of faulty nodes.

To sum up, we have proposed a new approach for nanochip testing. Instead of performing a difficult and long external testing of the whole chip, we propose testing externally only the IOPs. Then, the cores execute mutual diagnosis with their direct neighbors and isolation mechanism which enables constructing clusters of fully-connected good nodes (which may be later used by IOPs to allocate tasks) and in isolating the defective nodes,

3.3 Self-discovery of routes

3.3.1 Introduction

Communication protocols for defective network must guarantee failure-free communication between nodes. The imminent question arises: how to route messages in the presence of defective nodes? Obviously, a simple XY routing [67] is not suitable because defective nodes constitute holes in the network, disrupting the array's regularity and therefore, eliminating the possibility for the XY routing approach. Let us consider peer-to-peer communication in the 2D mesh network with holes. Since the positions of holes in the array are *a priori* unknown, the use of multicast seems inevitable to enable communication between two nodes. However, frequent multicasting or broadcasting may cause heavy communication traffic, bottlenecks and network saturation. A reasonable compromise between two contradictory requirements has to be found. On one hand, redundancy (multicast) is needed to move around defective nodes. On the other hand the use of redundancy significantly increases the traffic overhead, making the communication very inefficient. Indeed, if every node needing to transfer data uses multicast, not only will multiple copies of the same data be present in the network, but also the communication latency will be very high due to the heavy traffic.

Let us now summarize the constraints for efficient inter-node communication:

- The routing protocol should be as simple as possible to keep router simple.
- It should enable moving around defective nodes located in unknown positions.
- It is necessary to minimize the communication latency.
- It is necessary to minimize the network traffic.

3.3.2 A brief reminder on communication

In this section we describe the communication between the IOP and the nodes in the array, which is a special case of peer-to-peer communication. Recall that an IOP needs to contact the nodes to allocate the incoming tasks.

We propose here using a contract net protocol (CNP) to discover the routes to the destination nodes (DN) in the presence of defective nodes, i.e., of holes in unknown positions in the array. The CNP is generally based on three phases:

- **Request**: The IOP multicasts a request message (RM) which includes a special route field to enable Route Field Forwarding.

- **Acknowledgement**: A node receiving the RM unicasts an acknowledgement message (ACK) to the IOP. The ACK message follows the route which was discovered by the RM and stored during the propagation of the RM.

- **Data transfer**: After receiving the response from a destination node, the IOP can use the route stored in the ACK message to send data to the node using unicast.

Let us describe now in details the three phases. Why are the request and acknowledgement phases needed? If we assume that we have to send *k* packets of data, multicasting all *k* packets would have a disastrous effect on the network traffic. Instead, only one request message is multicast or broadcast. Note that this single message may be easily stored in one packet so that the multicast does not generate a huge traffic in the network. The role of this initial multicast is purely to discover a valid route between the source and the destination. Then, the acknowledgement message which contains the route is sent and received. How does the IOP find the routes to a node? We propose to use Route Field Forwarding, a technique inspired by [68]. A detailed description of this method is provided in Appendix A. Briefly, each RM message stores in a dedicated field the route which it follows during the multicast phase. When it arrives at the destination node, the message stores the entire route to the IOP so that the node may use it to send the ACK message back to the IOP. Note that the same route is also incorporated in the ACK message so that when it is received by the IOP, it can be stored in the IOP's buffers for further use. Finally, *k* packets of data are unicast by the IOP, following the route to the node. The whole mechanism allows to dramatically limit the utilization of the network resources. However, two main problems persist in this approach:

- The round trip needed for exchanging the RM and ACK messages (needed for route discovery) inevitably increases the communication latency.

- This method uses multicasting and this may still lead to some network saturation problem in case of heavy traffic as multiple multicasts propagating at the same time will quickly lead to congestion in the network.

Various multicast protocols are very well known and have been thoroughly described in the literature [69][70][71]. We considered two multicast techniques:

- flooding,
- gossiping.

The above protocols are described and compared in Appendix A. We also studied the modifications of these protocols (limited flooding etc.).

3.3.3 Comparison of the efficiency of multicast protocols

In this section, we compare the robustness and the efficiency of various multicast techniques. The parameters in our study were the percentage of defective nodes $p_{f,N}$, the re-emission probability (REP) for gossiping (see Appendix A) and the maximum number of message retransmissions (TTL). We performed simulations for the following values of REP: 1 (which corresponds to flooding), 0.8, 0.6 and 0.5.

We used the simulator MASS to carry out this study. The detailed description of the simulator may be found in Appendix B.

In short, MASS is an integrated development environment (IDE) to study the coexistence of large populations of agents. An agent is any autonomous system. The critical point is that we consider a behavioral description of agents, where each agent executes its mission following a decision tree. This approach allows to model agents and their environment as finite-state automata. This paradigm has been widely used as a unifying perception throughout the literature of agents and multi-agent systems. Each agent executes a mission. The extensions include the code to define the agent mission and the policy followed in the framework of the mission. MASS is split in two parts:

- The core includes the general classes to write a Windows application. It encapsulates the simulation engine in an interactive graphical interface.

- The behavioral code of the agent is represented by dynamic linked libraries (DLL) written by third-party programmers. Each DLL describes one (or several) policy (policies) followed by the agent to execute its mission.

In the studies considered here, each router in the core array is represented by a state automaton (SA), which forwards incoming messages. All routers are activated in the asynchronous mode through a global scheduler.

Let N_N be the total number of nodes in the network. The principle of each simulation is as follows:

- The simulator randomly generates a fraction $N_N \cdot p_{f,N}$ of faulty nodes in the network.

- The IOP emits a RM message which is broadcast in the defective network according to the rules described in Appendix A.

- The program calculates how many nodes N_C received the message. An array $table[N_N]$ is created and the element $table[N_C]$ is incremented every time exactly N_C nodes received the RM.

This procedure is repeated typically 2000 times. At the end of this simulation, the element $table[i]$, determines the number of times i nodes were reached. For instance, $table[26]$=110 means that the message reached exactly 26 nodes in 110 simulations. The probability to reach 26 nodes is therefore $p(26)$=110/2000.

Some exemplary results are presented in the figure below. We limited the propagation to 4 hops (TTL was set to 4). Note that a “hop” is a distance between two adjacent nodes in the network. The simulated multicasts methods were flooding and gossiping with REP equal to 0.5, 0.6 and 0.8.

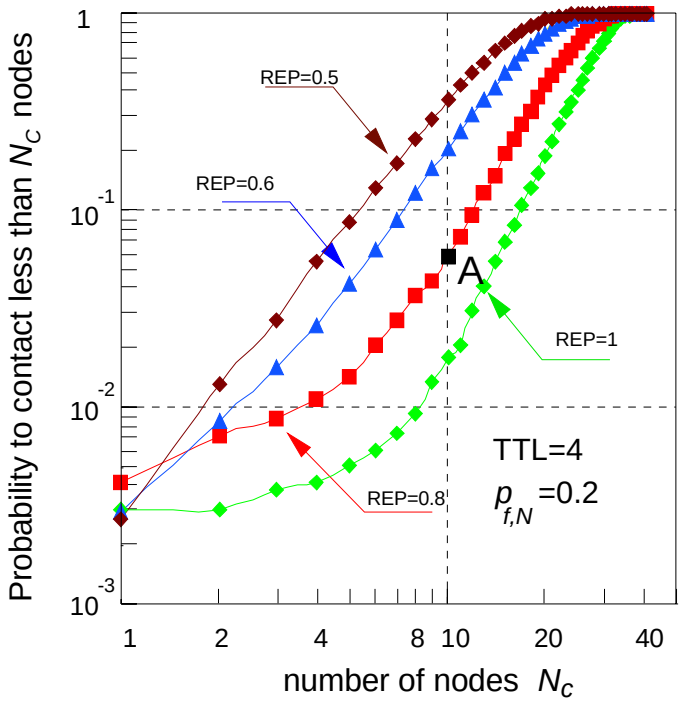


Fig. 26: Simulations results for various multicast protocols

Fig. 26 shows the probability that less than N_C nodes received the message in a defective network including 20% of defective nodes. Consequently, it allows to compare the multicast techniques in terms of practical viability. Consider for example point A. It shows that when gossiping with REP=0.8 is used as multicast method, the probability $p(N_C<10)$ that the message was received by less than 10 nodes is 0.05. Consider now the same probability for other protocols: flooding (i.e., REP=1) is the most robust, as $p(N_C<10) = 0.02$. The results of other gossiping protocols are worse. For example, when gossiping with REP=50% is used, the same probability $p(N_C<10)\approx 0.3$ which is 15 times higher than the same probability for the flooding protocol.

Considering these results, we conclude that gossiping is certainly an interesting technique to limit the number of useless retransmissions. However, it is less efficient to contact the cores in the network, which in our work is a very important parameter. Moreover, the major advantages of the flooding protocol are that it may be easily implemented in hardware and that it guarantees that the message will be delivered to all accessible nodes. Therefore, we believe that despite heavy redundancy, flooding should be used as a broadcast protocol to discover routes in multicore chips.

3.3.4 Route discovery mechanism

It is obvious that frequently broadcasting messages will greatly increase the traffic overhead in the network. Therefore, we should reduce broadcast emissions as much as possible. Consequently, we propose to change our approach to route discovery. Instead of emitting a broadcast every time the IOP has to contact a node, we propose to broadcast one single request message to all nodes at the beginning of chip operation. Then:

- Each non-defective node sends back to the IOP an ACK message, using the mechanism of Route Field Forwarding (see Appendix A).

- The IOP receives all ACK messages which comprise the routes to all nodes (see section 3.3.2) and creates a Valid Route Array (VRA) which can be implemented in the IOP's memory. The VRA stores the routes to the reachable nodes, and possibly the nodes' address, the nodes' state (busy, idle or pending) as well as any additional necessary information. Whenever the IOP wants to contact the nodes during chip operation, it may use the routes stored in its VRA. This method allows to avoid frequent broadcasting because the route discovery which uses broadcast is only executed once at the power-on and possibly periodically at runtime to update the routes. Thus, the traffic in the network is greatly reduced.

Let us now analyze the number of bits necessary for storing the route in a message. In the approach based on Route Field Forwarding (RFF, see Appendix A) that we suggest, each node adds to the propagating request message the information about the message route. A route field in the message header will naturally increase the message size. The number of bits necessary to store the route in the message is of course proportional to the distance between the IOP and a node. In our direction coding scheme (see Appendix A), exactly two bits per hop are needed, assuming mesh topology. This solution is much better than the similar idea presented in [72], which was based on storing the absolute node addresses in the route field. Let us compare the number of bits needed for storing the route in the message as a function of the network size.

We consider a *N*x*N* 2D mesh (square) network without defective nodes and an IOP located in the centre. The longest route (from the IOP to the farthest node) needs exactly (*N*-1) hops. When absolute node addresses are used in the route field, the number of bits rapidly increases with the number N^2 of nodes, because we need more than $\log_2(N^2)$ bits to effectively address *N* nodes:

Eq. 3
$$n_{bits,1} \geq \log_2 N^2$$

Consequently, the total number of bits needed for coding the longest route in the network using absolute addressing is:

Eq. 4
$$n_{bits,1} \geq 2\log_2(N) \cdot (N-1)$$

When 2-bit direction coding is used, the number of bits needed for coding the longest route in the network is:

Eq. 5 $$n_{bits,2} = 2(N-1)$$

The advantage of direction coding is clearly visible in the graph below, where we plotted Eq. 4 and Eq. 5 as a function of the network size *N*.

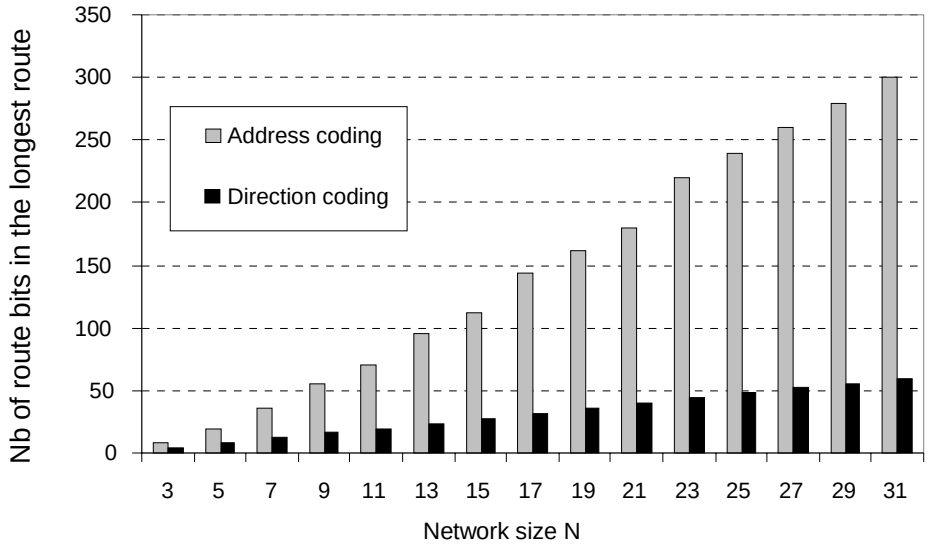


Fig. 27: Comparison of address and direction coding

For example, a 21x21 network comprises 441 nodes. Consequently 9 bits are needed for absolute addressing. The longest route is 20 hops implying that 20·9=180 bits are needed in the route field when address coding is used. Contrarily, only 20·2=40 bits are needed in case of direction coding. Clearly, the former solution is not suitable for large networks, as the message header would increase dramatically, which would likely constitute an unacceptable communication handicap.

Note that the routes are longer in the presence of defective nodes because the messages have to move around holes in the network. Consider for example Fig. 54 page 75 which shows that in the 21x21 network, some nodes are located even 30 hops away from the IOP. Consequently, in very large networks with large node failure probability, the route field may become very large even in case of direction coding.

3.4 Self-deactivation of defective cores

In this section, we propose to improve the methodology presented in section 3.2.4, page 38. Let us now analyze the disadvantages of the method based on the isolation of defective nodes. Let us consider a faulty core as shown in the figure below. Letters C represent cores (the right hand core is faulty) and letters R represent routers. If we apply the technique described in the previous chapter, the left good core will stop communicating with its neighbor. Note that the good core cannot know whether the defect in the neighboring node is a core defect, a router defect or an interconnection defect, although core defects are much more probable. There is then no other choice than disconnecting the entire node. In such a case, disconnecting the node means that we cut off the router which is possibly good!

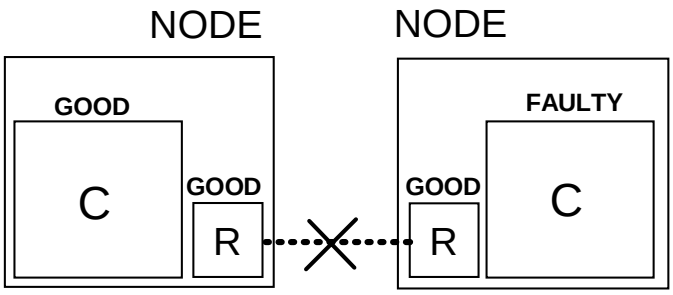


Fig. 28: Illustration of the problem in the testing method based on isolation of defective nodes

Now, the loss of good routers might be a major performance limiting factor. Indeed, if we isolate the entire node, we create a hole in the network, as shown for example in Fig. 25, page 39. Not only does it increase the communication latency because the messages have to be routed around the holes, but also holes may block the access to the good nodes, making them unreachable and, therefore, useless. In other words, the network connectivity is greatly reduced and the network performance decreases because there are fewer cores to execute tasks.

Is this performance degradation (due to the holes created by the isolated nodes) inevitable? In other words, can we suggest another method which would avoid this problem? A more effective technique dealing with core defects would allow the disconnection of defective core only. In a first step let us assume that all routers and communication links in the network are fault-free. The new technique is based on the self-shutdown of defective cores. This method *de facto* shutdowns all processors outside the SCZs which enclose the IOPs (see Fig. 25, page 39). A faulty core is shut down by its corresponding router. The simplest method to describe the self-shutdown technique is undoubtedly to show its execution with an example. So, we show this in Fig. 29 considering a one dimensional network with one IOP. It must be emphasized here that the following analysis is perfectly similar for a 2-D network. Line 1 displays a linear network with 8 cores connected to the IOP with three defective cores: 1, 6 and 8.

- First, the mutual test is performed following the rules presented in section 3.2.3, page 37. When the IOP tests its neighbors (cores 2 and 3), it diagnoses them as good and the test result is stored in the IOP router by setting both East and West bit to G (see line 1 in Fig. 29). Similarly, when core 2 tests its neighbors, it diagnoses core 1 as faulty and the IOP as good, and this result is stored in its router by setting the West bit to F (for faulty) and East bit to G (for good). The diagnosis being executed by all nodes, we get the diagnosis result represented by all router bits in line 1, namely X FG GG GG GG GF XX FF X. Letter X means that the result produced by a defective node is unpredictable (G or F).

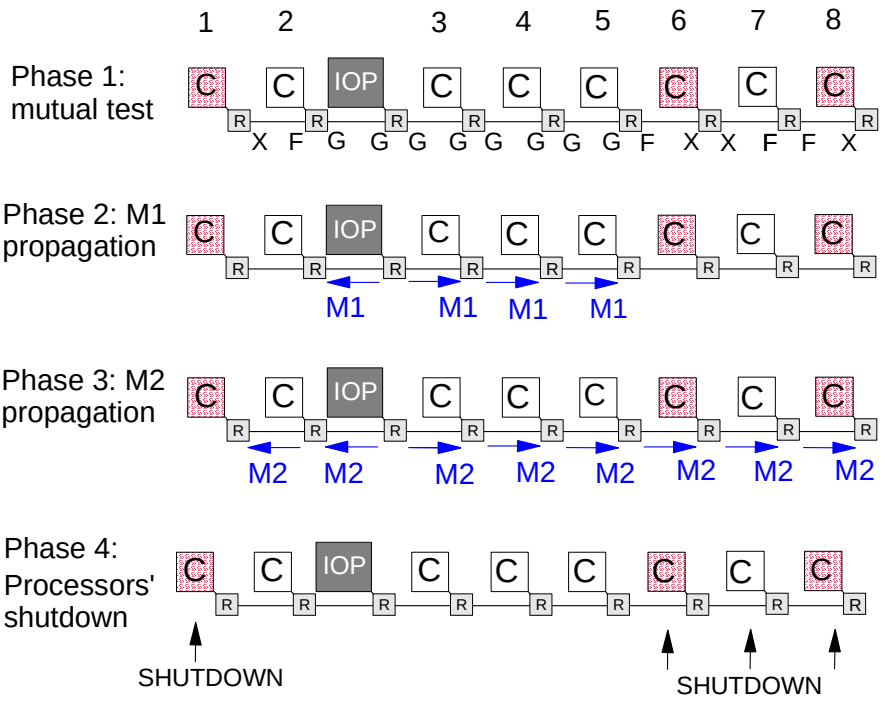


Fig. 29: Illustration of the self-deactivation method in one dimensional network

- Second, the M1 message is broadcast by the IOP (see section 3.3.4, page 41). When a good node receives this message, it only retransmits it to the neighbors which it diagnosed as good by the mutual test. Therefore, the message will be received by all nodes inside the SCZ around IOP and the broadcast will be stopped at the border of this SCZ. In our sample figure, line 2 shows that the message will be received by nodes 2, 3, 4, and 5.
- Third, a second message (M2) is emitted by IOP, which is again broadcast across the network. The crucial point is that the broadcast is now unrestricted and propagates across the entire network (also outside the SCZ) as shown in line 3, as we assume that routers are good.
- Finally, each router which received the second message M2 but did not receive the first message M1 shutdowns its corresponding core (sets its state to “faulty”). As a result, all processors outside the main SCZ enclosing the IOP are shut down.

Additionally, although a core is deactivated, its corresponding router remains active and may be used to route traffic in the network. The disconnection of routers would create holes in the network, which can result in unbalanced traffic, bottlenecks and the need for more complex routing protocol. In this method routers are not disconnected and the network connectivity will remain high. The example of this important effect is shown in Fig. 29: the routers corresponding to the faulty cores (1, 6 and 8) are not isolated. Thus, we avoid reducing the network's connectivity. Note that this method succeeds in deactivating all defective processors (cores 1, 6 and 8 in Fig. 29) but at the same time some good processors are also deactivated (core 7 in Fig. 29). To answer the question how many good nodes are unnecessarily deactivated (to analyze the method efficiency), we can use the results of the simulations which will be presented in chapter 4, because the number of cores validated by our method is actually equal to the number of nodes in the main SCZ (nodes which can be contacted by IOP). Please also note that the good nodes which are deactivated using this method are also lost for processing in the node isolation method.

Little is changed in this method when the array comprises several IOPs. The M1 messages are emitted by all IOPs and each copy propagates in the zone around each IOP. Then, M2 message is emitted by all IOPs and each router which received the second message but did not receive any of the first messages, shutdowns its corresponding core.

Now, a new improved mechanism is based on one critical assumption: we assume that 1. the links and the routers in the multicore network are not defective, or that 2. they have been already tested externally and that all defective routers and links have been detected.

- This first assumption is not unreasonable: routers are typically smaller than processors and the probability of defects in the router circuit is therefore lower.

- The second assumption is also acceptable, as an external router and link test is easier to execute than the test of the entire chip because, as mentioned previously, routers constitute only a fraction of the chip and, what is even more important, they form a naturally connected array which is very suitable for testing purposes [73].

3.5 Summary

We presented in this chapter three methods for chip self-configuration. First, we analyzed the software-based self-test techniques:

- The simplest method, based on node self-test needs no inter-node communication. However, due to the fact that there exists a non-zero probability that a defective node declares itself as non-defective, an additional testing circuit would have to be implemented. Note that this circuit should be also tested or designed with a high protection level, as it must work correctly to perform testing operations.

- The second method avoids the above-mentioned problem. It is based on the mutual tests of adjacent cores, which requires for neighboring nodes to communicate. This method succeeds in isolating defective nodes, but, as a result, the network connectivity is reduced.

- The third method is more efficient since it succeeds in disconnecting the cores only, instead of disconnecting the entire node. However, this method may only work under the assumption that all links and routers are non-defective or have been earlier tested externally.

For now, the decision which of these methods is the best depends on many factors like the core failure probability, the router failure probability, the network size and complexity etc. However, we believe that the second method presents the best solution for multicore nanochips, as it is the most general: it is based on no assumptions considering the state of the routers and depends on the actions of good cores only.

4 Study of route discovery in defective grid

4.1 Introduction

In this chapter, we analyze the efficiency of the route discovery, which is performed according to the mechanism presented in section 3.3.4, page 41. The metrics that we will frequently use throughout this chapter are based on calculating the probability to access a fraction of nodes. Recall that the number of accessible nodes is generally lower than the number of good nodes, because the defective nodes sometimes block the communications between some good nodes and the IOP(s), as shown for instance in Fig. 25, page 39.

We consider several chip topologies to determine which one guarantees the best performance against various fractions of defective cores and (or) defective interconnects in the network. Moreover, we also consider multi-port architectures, where we additionally analyze the protection of ports, their optimal positioning in the network and the distribution of the workload among the ports.

The number of accessible nodes (i.e., of nodes free for processing) may be also viewed as an estimate of the chip performance. We also developed two analytical chip validation methods which take into account the number of accessible nodes and their position in the network. These methods may be used to determine whether a chip will have acceptable performance or, contrarily, cannot provide the desired performance and should not be validated for deployment. We also estimate the percentage of accepted chips (production yield) in function of percentage of defects for various topologies.

4.2 Route discovery in uniport architectures

In this section we study the capability of the IOP to contact the valid cores in the defective network. More precisely, we have performed simulations which determine the probability that the IOP contacts at least the fraction f_C of cores. This metrics may be viewed as a measure of the efficiency of the route discovery protocol. More precisely, knowing how many cores are accessible and can take part in the processing versus the node failure probability $p_{f,N}$, we can estimate the performance degradation of a chip as a function of the fraction of defective cores.

The performance degradation is inevitably caused by the following factors:

- The defective nodes cannot take part in the processing.
- The defective nodes may block the access to some good nodes which will not be able to take part in the processing.
- The messages will have to move around the defective nodes, increasing the communication latency and, therefore, decreasing the global processing power.

The first two factors are analyzed in this chapter and the third one is described in detail in section 5.2 page 100.

Again, we used the simulator MASS to perform simulations of communications in a defective network. The simulations are similar to those described in chapter 3.3.3 page 42, but for the sake of clarity we will reformulate them here. The simulator randomly generates a fraction of $N_N \cdot p_{f,N}$ of holes (defective nodes) in the network. N_N is the total number of nodes and $p_{f,N}$ is the probability that a node is defective. $p_{f,N}$ is a parameter in our study and ranging from several percent to 40%. We assume that the defective nodes are isolated using the mutual tests, as described in chapter 3.2.4, page 38. Then, each IOP emits a message which is broadcast across the network following the rules of flooding. The program calculates the number of nodes which receives the broadcast which corresponds to the size of the cluster of good nodes around the IOP. In other words, we calculate the number of accessible nodes available for processing. Simulations have been performed for a wide range of $p_{f,N}$ (from 0.1 to 0.4) and for various network topologies.

We chose four network architectures: hexagonal, mesh, torus and 2-layer mesh with the node connectivity ranging from C=3 (hexagonal network) to C=5 (2-layer mesh). All topologies have been implemented in simulator MASS.

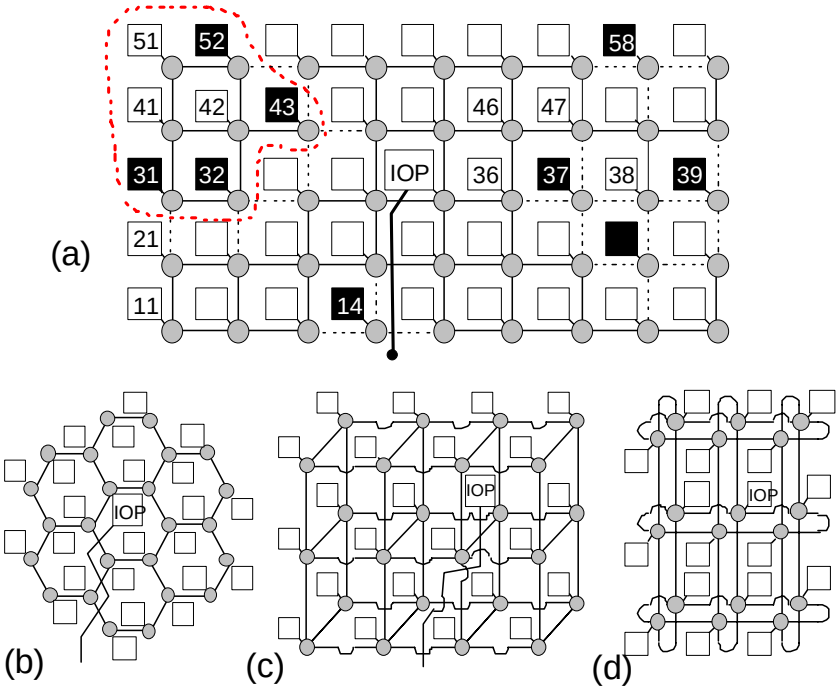


Fig. 30: Four analyzed chip topologies : 2D mesh (a), hexagonal (b), 2-layer mesh (c) and torus (d)

Obviously, the architectures have different connectivities: in the hexagonal network the connectivity equals C=3 because each node has exactly three neighbours (except for the nodes located at the border). Similarly, the connectivity of most nodes in the mesh network is C=4 and the connectivity of most nodes in the 2-layer mesh network is C=5. The torus is a modification of the mesh network, where in fact there is no border so that all nodes have the connectivity C=4. To enable comparison of the various topologies, the simulations for all networks have been performed for networks with a similar number of nodes. More precisely, the hexagonal network had 450 nodes, the 2-layer mesh network was constructed using two connected 15x15 meshes with a total of $15 \cdot 15 \cdot 2 = 450$ nodes. The mesh and the torus were 21x21 arrays, with a total of 441 nodes.

The simulation results are presented in two following sections.

4.2.1 Reachability in the presence of defective cores

By reachability, we mean the ability of the IOP to contact the cores in the network[(3)]. Reachability simulation results are presented in the following manner: The x axis represents a fraction of chip cores (f_C) whereas the y axis represents the probability that at least the fraction f_C is accessible and available for processing. The probability was calculated as a statistical average from several thousands of iterations. This way of presenting the simulation results allows us to determine the percentage of manufactured chips which guarantee at least a user-defined threshold of reachability. For example, the fact that the probability of contacting

[(3)] We simply follow the graph theory where the reachability is the notion of being able to get from one vertex in a directed graph to some other vertex.

more than 60% of cores equals 0.8 also means that statistically 80% of fabricated chips will be such that the IOP will be able to contact 60% of the cores.

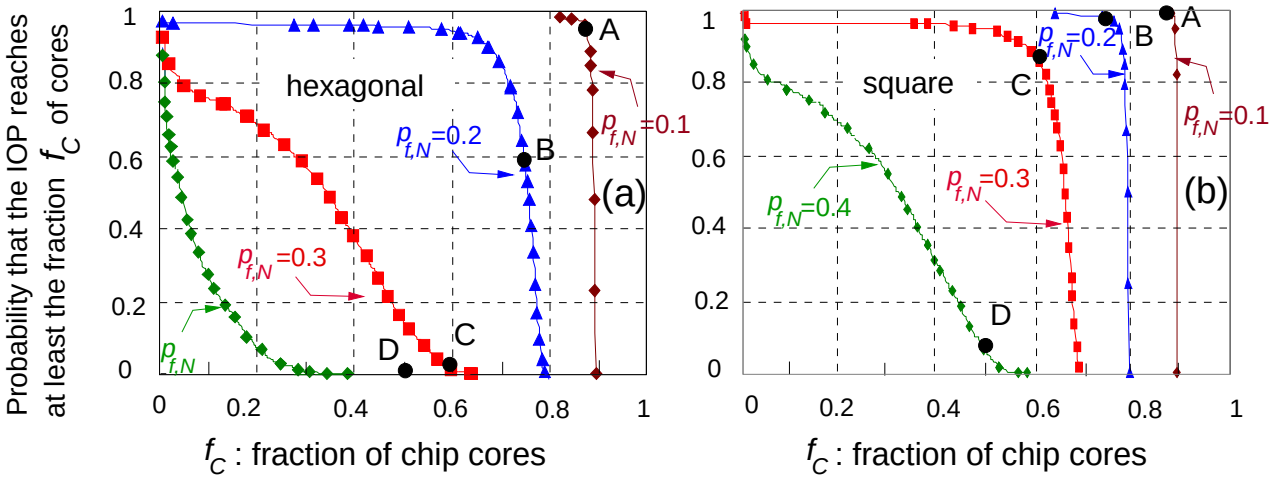


Fig. 31: Probability that the IOP reaches at least a fraction f_C of cores in a hexagonal (a) or a 2D-mesh (square) topology (b) versus the fraction of defective nodes

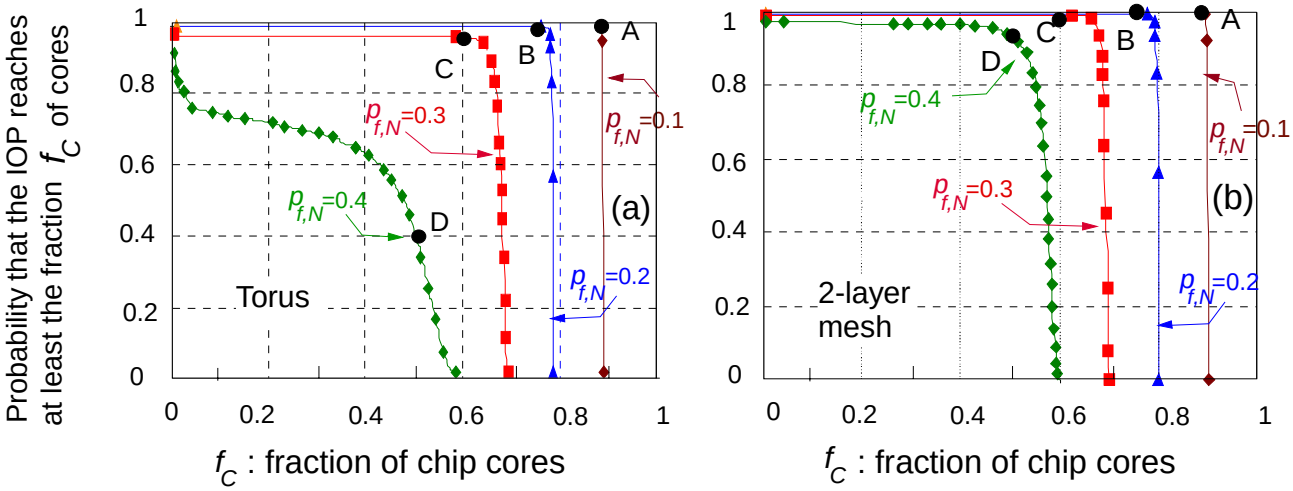


Fig. 32: Probability that the IOP reaches at least a fraction f_C of cores in a torus (a) or a 2-layer mesh (b) versus the fraction of defective nodes

.

To better explain the figures let us consider several particular points. For instance, let us consider the four points A with abscissa X=0.88 in the four figures. The ordinate of this point determines the probability that the IOP is able to contact at least 88% of the cores when 10% are defective (because $p_{f,N}$ =0.1). Consequently, about 2% of the cores are statistically not contacted, i.e., they are lost for the processing, probably because they are blocked by defective cores. Clearly, the efficiency of the Route Discovery Mechanism (RDM) may be considered as satisfactory in all topologies. Even if we analyze the smallest node connectivity (hexagonal topology), the probability is about 0.95 that the IOP will contact 88% of nodes (see Fig. 31).

Now, let us consider the RDM in a more defective network when the node failure probability is $p_{f,N}$ =0.2. Points B in the four figures determine the probability that at least 75% of the

nodes can be contacted by the IOP when the network includes 20% of defective nodes. The sole case which becomes problematical is that of the hexagonal network Fig. 31), as the probability reduces to 0.6.

Points C in the four figures show the efficiency to reach at least 60% of cores when the network includes 30% of defective nodes. It is extremely low in the hexagonal network (Fig. 31a). The two networks with connectivity 4 (i.e., the 2D mesh in Fig. 31b and the torus in Fig. 32a) may be considered as satisfactory up to this percentage of defective nodes, as the probability that the IOP contacts at least 60% of nodes is larger or close to 0.9. However, as much as 40% of the nodes are lost for the processing, corresponding to 30% of defective nodes and 10% of inaccessible nodes!

Now let us analyze the extreme case with approximately 40% of defective nodes, i.e., $p_{f,N}$ =0.4. Points D in the four figures show the probability to discover at least 50% of the nodes. Consequently, 50% of the nodes would be lost for the processing (40% because they are defective and 10% because they are inaccessible). The only network which guarantees the desired functioning is the 2-layer mesh shown in Fig. 32b. We can see that when the node connectivity is 5, the probability to reach at least 50% of the nodes in the chip (out of 60%) rises to about 0.93.

For the sake of better comparison of the various topologies, we also present the results in a different way: we display in Fig. 33 the curves for all topologies and the same value of $p_{f,N}$.

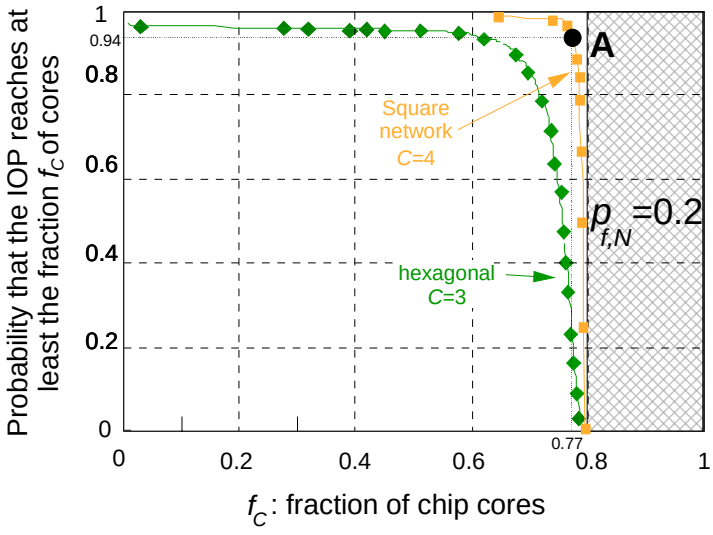


Fig. 33: Probability that the IOP reaches at least the fraction f_C of cores in a hexagonal and a square topology versus the fraction of defective nodes. The network is made up of 450 cores including 20% of defective and isolated cores

Fig. 33 displays the simulation results for a network with 20% of defective nodes. Clearly, even the network with the lowest connectivity C=3 provides satisfactory results and the results for the square (2D mesh) network are even better. For instance, point A means that the probability of contacting at least 77% of the cores (out of total 80% of non-defective cores) is 0.94 for the square (2D mesh) topology. For the sake of clarity, the curves for torus and 2-layer mesh topology have not been drawn in the figure, as it is obvious that the results for these topologies would be even better.

However, the core accessibility degrades dramatically when there are 40% of defective cores in the network as shown in Fig. 34. Let us suppose that we require the IOP to contact at least 53% of cores (out of the total 60% of non-defective cores). The figure below shows that it is quasi-impossible in the hexagonal or mesh network! The results for the torus (see point B in the figure below) are slightly better but still not satisfactory because as few as 20% of chips would fulfil the above requirement. Only the topology with the highest connectivity (i.e., the 2-layer mesh) provides satisfactory results, as the probability of contacting at least 53% of cores is 0.9.

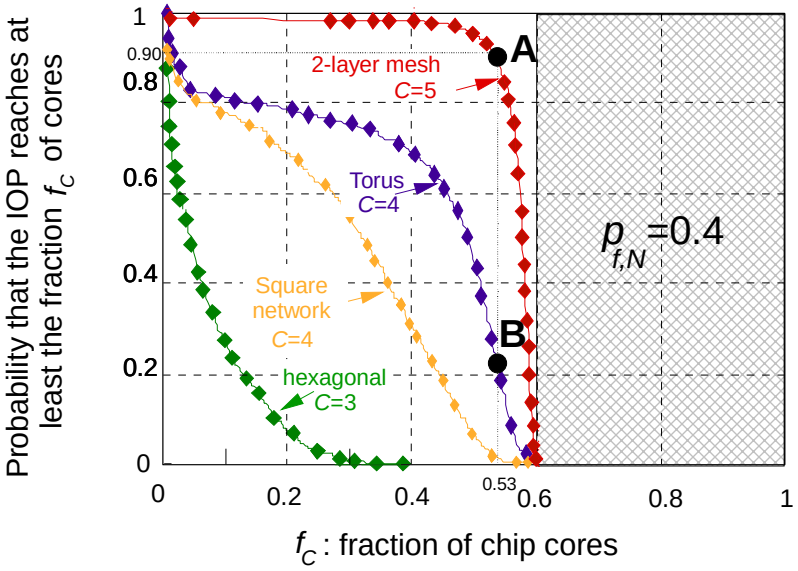


Fig. 34: Probability that the IOP reaches at least the fraction f_C of cores in a hexagonal, a square, a torus or an 2-layer mesh topology versus the fraction of defective nodes. The network is made up of 450 cores including 40% of defective and isolated cores

4.2.2 Reachability in the presence of defective links

Defective interconnects (links) will also inevitably degrade the reachability due to the following factors:

- The defective links may block the access to some good nodes which will not be able to take part in the processing.

- The messages cannot be transferred via defective links, which forces moving around defective nodes, increases the communication latency and, therefore, decreases the processing power.

To consider defective links, we have to include one more parameter (in addition to $p_{f,N}$) in our simulations: which is the link failure probability $p_{f,L}$. A crucial question is of course: what is the relation between $p_{f,N}$ and $p_{f,L}$? In other words, what is the dependence between node faults and link faults in the technology? Yet, the answer is not so simple. We might of course predict roughly the relation $p_{f,N}/p_{f,L}$ for a given technology, but these parameters do not depend only

on the used technology. The probability that a node is defective depends heavily on the node complexity (the number of transistors in the node) and on the hardware protection method which has been implemented. Similarly, the probability that a link is defective depends on its size and, again, on the hardware protection method. Therefore, since the ratio between $p_{f,N}$ and $p_{f,L}$ may vary depending on design choices, consequently we performed simulations for of wide range of values of $p_{f,N}$ and $p_{f,L}$. The simulation method is similar to the method described in the previous paragraph except that the program generates randomly defective links in addition to defective nodes. We studied the 2D mesh network in details and compared the obtained results with other topologies.

In a NxN mesh network, there are exactly $2N(N\text{-}1)$ inter-node links. The program first generates $N^2 \cdot p_{f,N}$ defective nodes and then $2N(N\text{-}1)\cdot p_{f,L}$ defective links.

First, we analyzed the reachability in a network with no defective nodes ($p_{f,N}$ =0) and for several values of $p_{f,L}$. The results are presented in the same way as in the previous paragraph.

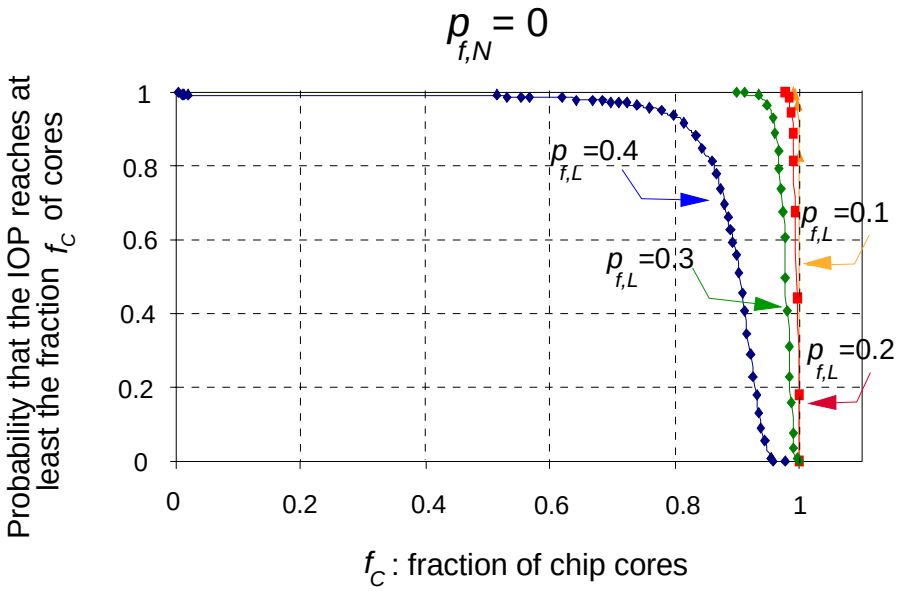


Fig. 35: Probability that the IOP reaches at least a fraction f_C of cores in 2D mesh network for various fractions of defective links and no defective cores

As we can see in Fig. 35, the reduction of the reachability caused by the defective links only becomes significant when $p_{f,L}$>0.3. It turns out that the degradation of the reachability due to defective links is smaller than the corresponding degradation due to defective nodes (compare with Fig. 31, page 54).

Even with 40% of defective links, the reachability is still more than acceptable (the leftmost curve). However, it decreases if defective nodes are added to the network. In Fig. 36, simulation results are presented for 10% of defective cores and various percentages of defective links.

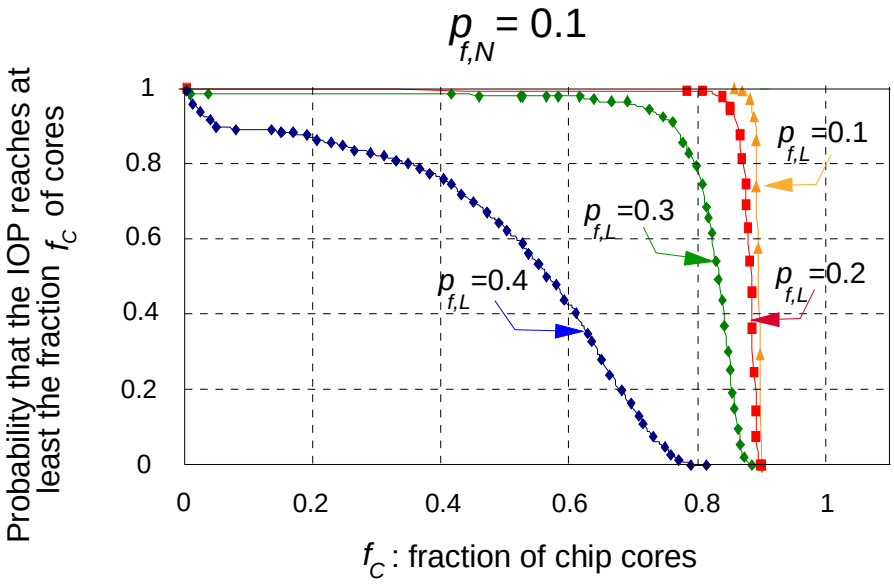


Fig. 36: Probability that the IOP reaches at least a fraction f_C of cores in 2D mesh network for various fractions of defective links and 10% of defective cores

The results in Fig. 36 prove that the performance degradation due to defective nodes is acceptable for the $p_{f,L}$ values up to 0.3. Then, as $p_{f,L}$ changes from 0.3 to 0.4, the performance degradation increases considerably. For example, the probability of contacting 60% of cores is almost 1 for $p_{f,L}$ =0.3 but only 0.4 for $p_{f,L}$ =0.4.

The following figure presents the results for various combinations of $p_{f,N}$ and $p_{f,L}$.

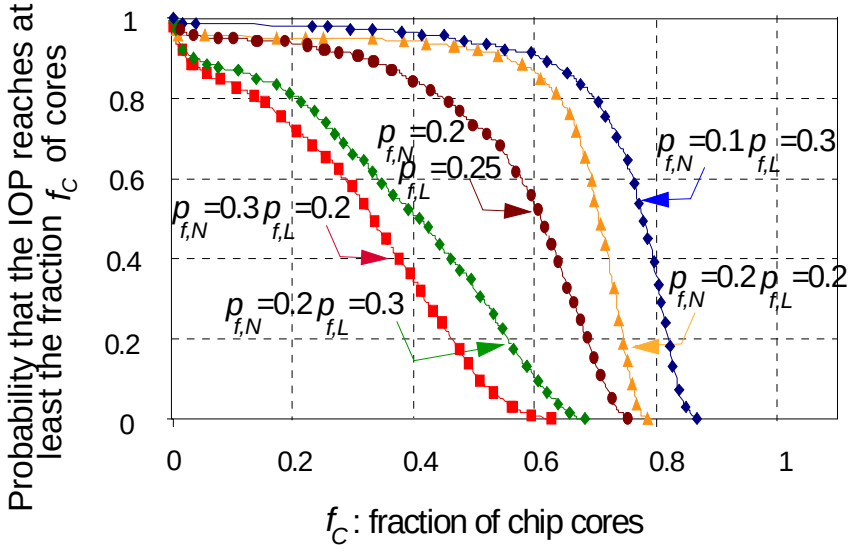


Fig. 37: Probability that the IOP reaches at least a fraction f_C of cores in 2D mesh network for various fractions of defective links and cores

Analyzing the results presented in Fig. 37, we can draw several interesting conclusions. Surprisingly, we see an enormous difference between the second and fourth curve (counting from the right). Both curves have been calculated with $p_{f,N}$ =0.2. The second curve represents the results for $p_{f,L}$ =0.2 and fourth curve for $p_{f,L}$ =0.3. Consequently, a difference of 10% in the

fraction of defective links is enough to cause a large dispersion of results. This means that probably there exists a threshold value of $p_{f,N}$ and $p_{f,L}$, and the accessibility of nodes drops significantly when exceeding this threshold.

The reduction of the reachability is more accentuated by the occurrence of defective cores than by the the presence of the defectives links. For example, the reachability is higher when $p_{f,N}$=0.2 and $p_{f,L}$ =0.3 than the inverted case when $p_{f,N}$ =0.3 and $p_{f,L}$ =0.2 (see Fig. 37). We can actually see that the $p_{f,N}$ parameter has almost twice greater impact on the results than $p_{f,L}$. The reason is obvious at low fault density. Simply, in a network with connectivity 4, the mutual diagnosis mechanism (see section 3.2.3 page 37) disconnects $4 \cdot p_{f,N} \cdot N$ links due to faulty cores and $2 \cdot p_{f,L} \cdot N$ links due to faulty links. So let us introduce the parameter $p_{f,O} = p_{f,N} + 0.5\ p_{f,L}$. The curves with the same $p_{f,O}$ should exhibit similar reachability. For example, for the second curve ($p_{f,N} = p_{f,L} = 0.2$) in Fig. 37 we may calculate:

Eq. 6 $$p_{f,O} = p_{f,N} + 0.5\ p_{f,L} = 0.3$$

Indeed, the results are very similar to the results for $p_{f,N}$ =0.3 and $p_{f,L}$ =0 (Fig. 31b). However, this rule is only valid for low values of $p_{f,L}$ (up to 0.2).

In general, defective links have a significant influence on chip performance only for high values of $p_{f,L}$ such that $p_{f,O} = p_{f,N} + 0.5\ p_{f,L} > 0.3$. Moreover, it seems that the reachability in such a network can be directly derived from simulation results performed by assuming only defective cores (no defective interconnects). More precisely, the network with the parameters $p_{f,N}$ and $p_{f,L}$ can be approximated by a network with parameters $p_{f,N}^*$ and $p_{f,L}^*$ where:

Eq. 7 $$p_{f,N}^* = p_{f,N} + 0.5\ p_{f,L} \quad \text{and} \quad p_{f,L}^* = 0$$

For example, let us assume that we want to estimate the route discovery efficiency for $p_{f,N}$=0.2 and $p_{f,L}$=0.2. Now according to Eq. 7, the results should be comparable with the results obtained for $p_{f,N}$ =0.3 and $p_{f,L}$ =0. Therefore, in the following chapters the majority of analysis and simulations has been performed for networks with defective nodes only, as we assume that the parameter $p_{f,N}$ used in our study may represent the overall impact of both defective nodes and links.

We also compared various topologies in terms of resilience to defective links. The figures below are similar to Fig. 33 and Fig. 34, but the crucial parameter is now the fraction of defective links. The simulations have been performed for networks with about 450 nodes.

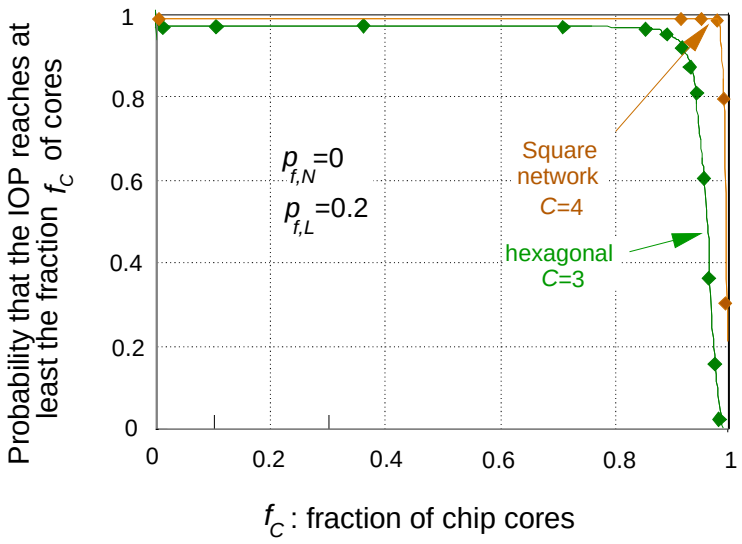


Fig. 38: Probability that the IOP reaches more than the fraction f_C of valid cores in a hexagonal and a square topology when the grid includes 20% of defective links and no faulty cores.

The figure above presents the results for networks with 20% defective links and no defective cores. It shows that even for the low connectivity network based on the hexagonal topology the effect of defective links is negligible: the probability that the IOP contacts 90% of the cores is around 0.9. The resilience of hexagonal network decreases significantly when link failure probability $p_{f,L}$ =0.2 rises to 0.4 (see Fig. 39).

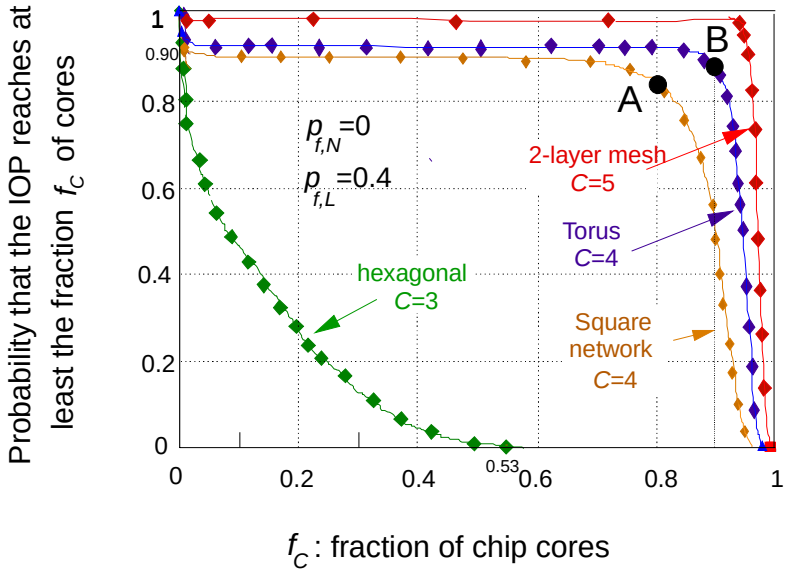


Fig. 39: Probability that the IOP reaches more than the fraction f_C of valid cores in a hexagonal, a square, a torus or a 2-layer mesh topology when the grid includes 40% of defective (and thus inactive) links.

Other topologies are quite resilient to link defects. When there are 40% of defective links in the network, the probability of contacting 80% of cores in a square (2D mesh) network is 0.85. The results for torus and 2-layer mesh topologies are even better. Consequently, if we compare the figure above with Fig. 34, we clearly see that in all grid topologies the

reachability degradation due defective nodes is much more important than the performance degradation due to defective links.

4.2.3 Production yield

The results obtained from our simulations may be used to estimate the production yield as a function of the core failure probability for various network topologies. The production yield (PY) is a crucial parameter in any massively defective technology. The first possible idea to validate a chip might be simply based on counting the number of cores responding to the RM emitted by the IOP, and on selecting the chips which enable contacting at least the pre-defined fraction f_C of cores. In this approach, the PY can be directly derived from Fig. 31 and Fig. 32. Simply, one need to multiply the ordinate of any point by the probability (1-$p_{f,N}$) , as the IOP must be also fault-free. Note that we assume here that there is a single IOP in the network and that its failure probability is the same as that of a single core. For instance, if we consider point C (X=0.6,Y≈0.95) in Fig. 32a, page 56 (which presents the simulation results for the torus topology for $p_{f,N}$=0.3), we conclude that the PY would be approximately 66% when selecting chips which enable contacting at least 60% of all nodes. However, one major disadvantage of this approach is that it only considers the average number of defective nodes and that it completely ignores their positions in the network, despite the fact that nodes close to IOP are more important to the network operation for preserving the communication bandwidth. It is clear that the traffic concentrates around the IOP and that the failure of nodes close to the IOP (especially the adjacent neighbors) will have a dramatically negative impact on the communication bandwidth and the communication latency.

Consequently, a more accurate selection methodology should include in the analysis the factor which takes into account the number of fault-free IOP neighbors. The probability $p_1(k, n_C, p_{f,N})$ that the IOP is fault-free and that at most k nodes (out of n_C connected to it) are defective reads:

Eq. 8
$$p_1(k, n_C, p_{f,N}) = (1 - p_{f,N}) \sum_{i=0}^{k} \binom{n_C}{i} (1 - p_{f,N})^{n_C - i} {p_{f,N}}^i$$

$p_1(k, n_C, p_{f,N})$ is a kind of locality factor, which takes into account the state of the environment around the IOP. If we select chips considering the close environment, the PY becomes the product of this locality factor multiplied by the probability $p(f_C, p_{f,N})$ to contact at least the fraction f_C of nodes in the network under consideration:

Eq. 9
$$PY(k, n_C, f_C) = p_1(k, n_C, p_{f,N}) p(f_C, p_{f,N})$$

Note that this metrics is based on the assumption that the IOP and the core have the same failure probabilities. It is the worst-case scenario, as the IOP is crucial for correct chip operation and a designer may decide to minimize the IOP failure probability by implementing special hardware protection methods. Moreover, in this estimation of the production yield we consider the networks with one port only.

Fig. 40 and Fig. 41 display the estimations of the PY using Eq. 9 for a 450-node network organized in the previously considered topologies (hexagonal, 2D-mesh, torus or 2-layer mesh, see Fig. 30, page 53), when at most one defective node adjacent to the IOP is tolerated for chip validation. In other words, at least 2 out of 3 nodes adjacent to the IOP are not defective in the hexagonal network, 3 out of 4 in the square lattice or in the torus, and 4 out of 5 in the 2-layer mesh.

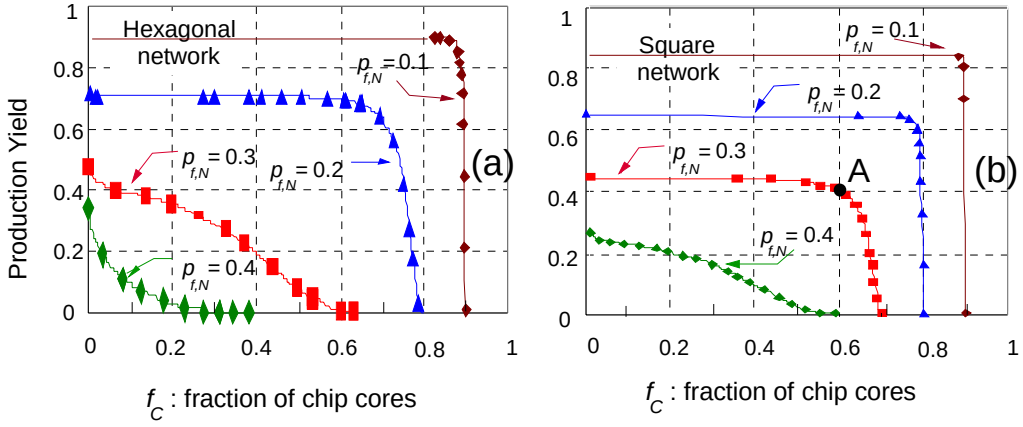


Fig. 40: Estimation of the production yield when selecting chips having at most one defective node adjacent to the IOP (a) Hexagonal; (b) Square (2D mesh).

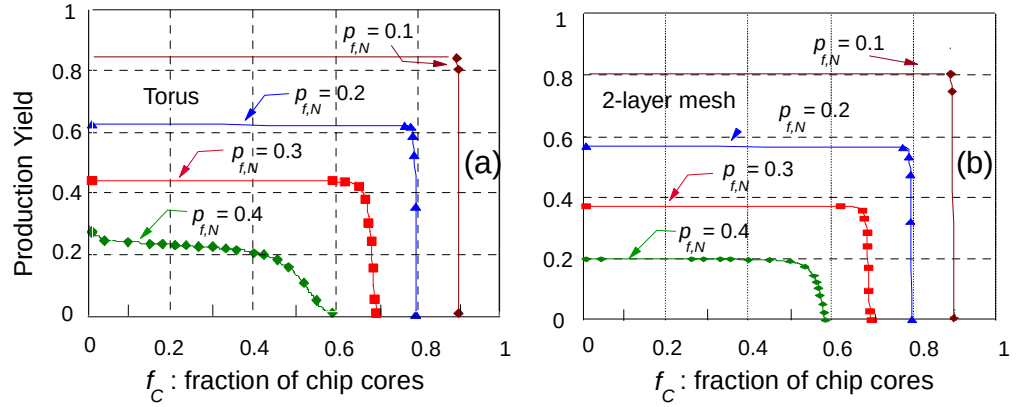


Fig. 41: Estimation of the production yield when selecting chips having at most one defective node adjacent to the IOP (a): Torus; (b) 2-layer mesh.

The figures show that when the locality factor is taken into consideration, the percentage of manufactured chips fulfilling the above-defined criteria is quite low for all topologies. If we compare the above figures with Fig. 31 and Fig. 32 page 54, we see that the locality factor caused a significant lowering of the curves. In Fig. 31 and Fig. 32, the curves for all values of $p_{f,N}$ have maximum values close to 1. Here, the maximum value depends on the node failure probability: the higher $p_{f,N}$, the lower the maximum value. In other words, the locality factor emphasizes the impact of $p_{f,N}$ on the results.

Let us now analyze one particular point for clarity. Consider the mesh network with 30% of defective nodes (Fig. 40b). Point A shows that statistically 40% of manufactured chips will guarantee that the IOP has at least 3 fault-free neighbors (out of total 4) and that the IOP can contact at least 60% of nodes (out of 70%).

Interestingly, the results for low connectivity networks are better when compared to high connectivity networks. For example, let us assume a network with 20% defective nodes ($p_{f,N}$=0.2). The production yield for hexagonal topology does not exceed 0.7 and for 2-layer mesh topology does not exceed 0.58. The poor results for higher connectivity networks are a direct consequence of the locality factor given by Eq. 8. Indeed, it is more difficult to guarantee that 4 out of 5 nodes adjacent to the IOP are good (2-layer mesh) than to guarantee that 2 out of 3 IOP neighbours are fault-free (hexagonal).

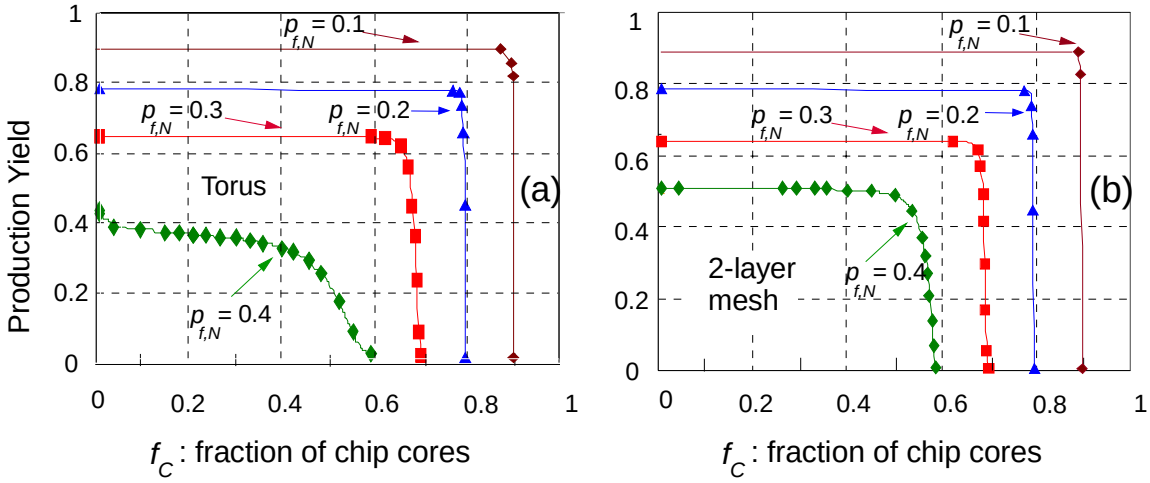


Fig. 42: Estimation of the production yield in the torus and 2-layer mesh topologies when selecting chips having at least two good nodes adjacent to the IOP.

Please note that in Fig. 40 and Fig. 41 the assumption that at most one node adjacent to the IOP may be defective is in fact detrimental to the higher connectivity networks. Therefore, we also present in Fig. 42 the same production yield (PY) for torus and 2-layer mesh topologies, but calculated under the assumption that at least two IOP neighbours are fault-free. The results are clearly better, which is no surprise, as we relaxed the constraint regarding the number of fault-free IOP neighbors. For instance, in the 2-layer mesh topology with $p_{f,N}$=0.4**,** the PY(f_C=0.4) is over 50% compared to 20% in Fig. 41. This suggests that to preserve production yield, one may consider applying additional protection methods to IOP and its adjacent cores to minimize their failure probability. As the number of such cores is not high, the chip would not suffer from large hardware overhead, yet the production yield will be surely higher.

One important question may be raised here: if indeed high connectivity networks (like the 2-layer mesh) provide better results than the 2D mesh network, why not simply using in the future the 2-layer mesh topology? It is possible, but mapping the 2-layer mesh topology on a planar chip surface is not an easy task. The 2D implementation of the 2-layer mesh will have many disadvantages like the occurrence of long interconnects of various lengths, irregular structure, unpredictable wire delay etc. It is obvious that the 2D mesh is the most natural choice of scalable on-chip network, as the 2D mesh topology is planar and therefore may be

easily mapped onto the chip surface without suffering from the above-mentioned disadvantages. **Hence, our analysis shows that the 2D mesh solution should be envisaged as a viable solution as long as the core failure probability does not exceed 20-25%. Above this core failure threshold, when the core failure probability is about 30%, the torus or the 2-layer mesh topologies will provide much better performance. When the core failure probability is greater than 30%, only the 2-layer mesh topology may be considered as a good solution, as any other topology will not guarantee the desired performance.**

4.2.4 Route randomization

When performing the route discovery simulations described in section 3.3.4, page 41, we discovered that this method has a significant drawback: the routes to the nodes located in the same region have a large overlap and are sometime almost identical. In other words, some directions in the network are privileged over others. This problem does not call our reachability studies into question, but it shows that the network traffic will not be distributed equally. Most of the messages will pass through the same nodes/links whereas other nodes/links will not be used at all. We illustrate this problem below with an example. Let us analyze the route discovery mechanism presented in section 3.3.4 to illustrate this problem with the array displayed in Fig. 43.

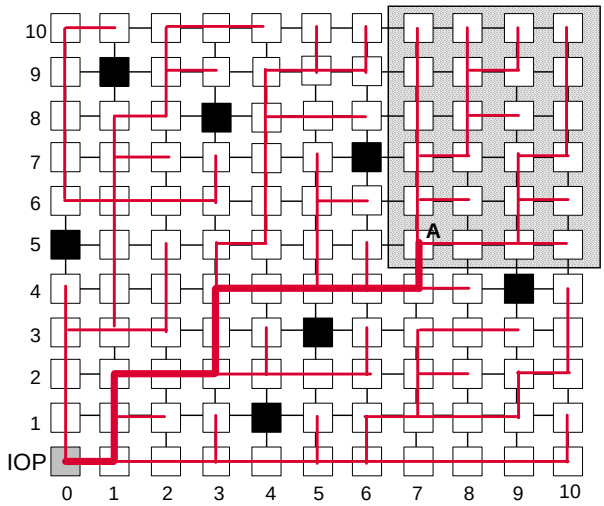


Fig. 43: Illustration of the route correlation problem

The IOP broadcasts the RM message. Every node stores in the message's route field the direction from which it was received and then forwards it to its neighbors. Note that only the first copy of the message is forwarded. Let us now take a closer look at Fig. 43 which depicts a fragment of 2D mesh network with one IOP. The sample propagation of the first copy of the broadcast message is marked by lines. When a message arrives at any node, its route field stores a complete route between the IOP and the node, which is identical to the propagation route of the first copy of the broadcast message. In other words, the lines in the figure above mark the routes which are later used for communication in the network.

We can see that the routes form a kind of a tree which connects the reachable nodes to the IOP. It can be easily pointed out that there are many nodes whose routes pass through the same nodes when contacting the IOP. For example, all nodes in the grey rectangle will use the same route (marked with the bold line) to communicate with IOP. Clearly, it will cause an unacceptable imbalance in the network traffic.

This effect is a direct consequence of the plesiochronous character of the network (see section 3.1, page 27). The plesiochronous operation means that all nodes' clocks have almost the same frequency and different phases. Moreover, the clock phases are random for every node.

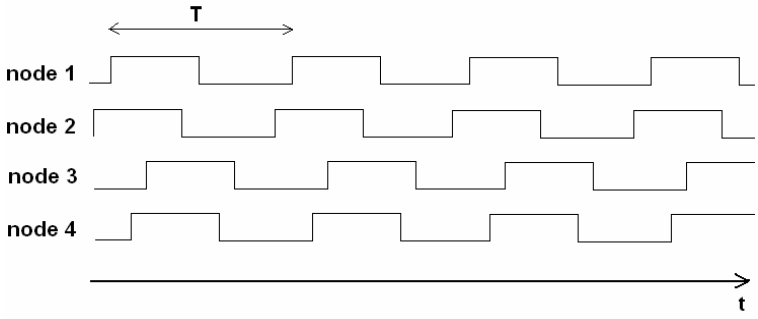


Fig. 44: Example of plesiochronous operation

Now, we observed that the dephasing between nodes' clocks allows faster propagation in some directions and causes slower propagation in the others. As a result, some directions are privileged in the network and most of the routes pass exactly through the same intermediate nodes during the route discovery mechanism. Of course, clock dephasings change with time, but routes stored in the IOP result from the initial correlation occurring during the initial route discovery mechanism! To study this effect, we analyzed the traffic distribution in the 30x30 2D mesh network, in particular the routes gathered by the IOP after one broadcast (flooding protocol). Then, we calculated the percentages of routes which pass through eight nodes located 2 hops away from the IOP. In case of balanced route discovery, a similar number of routes should pass through each node.

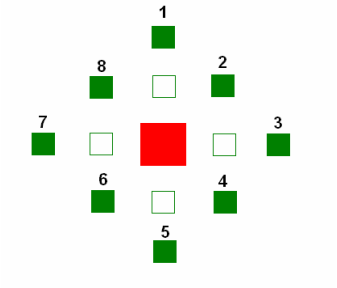


Fig. 45: Close environment of an IOP with eight nodes located 2 hops from IOP

The graphs below present the percentages of routes which pass through the eight nodes for three different trials. In every trial, we generate randomly the clock phases for all nodes.

To measure the distribution of routes, we used the standard deviation. Ideally, about 12.5% of routes should pass through each node and the standard deviation should be as low as possible, preferably zero. However, we discovered that the percentage of routes passing through the nodes varies from 0.1% to even 38.3% whereas the standard deviation is 10-12 %.

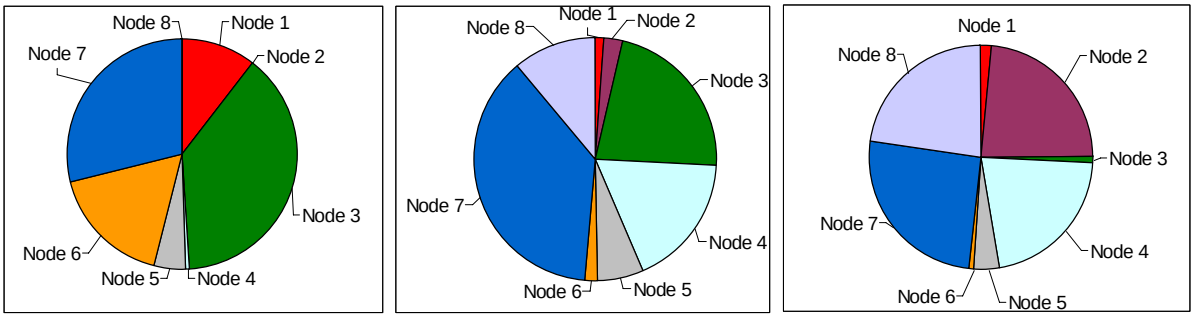


Fig. 46: Distribution of traffic passing through eight nodes located two hops from IOP without implementing randomization method

It is clear that the repartition of routes among the eight nodes changes from trial to trial but in one trial the distribution is never balanced. For example, in the third trial (corresponding to the right-hand chart in Fig. 46) only four nodes (nodes 2, 4, 7 and 8) out of eight are practically used to route the traffic. This means that the network bandwidth is virtually two times lower than the nominal network bandwidth. The occurrence of unbalanced traffic in the network means that all possible network resources are not used which can significantly reduce the system performance. Consequently, we describe below several alternative broadcast mechanisms.

Method 1

The first solution that comes to mind is to broadcast *m* times the RM message and to store in the IOP *m* routes to each node. This solution would work if the nodes clocks had different frequencies. However, we consider plesiochronous networks so all frequencies of nodes clocks are almost the same. Only clock phases are different. Consequently, if we broadcast *m* times the RM message in the relatively short time interval, it will propagate almost exactly following the same way and the *m* routes between any node and the IOP will be the same or will only be slightly different (due to the fact that the phases of the node clocks might slightly shift during the different broadcasts). Therefore, we applied a slightly modified method to obtain better results. We use several broadcasts, but every broadcast propagates randomly through the network. The randomization of the propagation is achieved by modifying the routing rules of the flooding protocol. We introduce a certain probability that message is withheld by a node during propagation (p_{HOLD}). More precisely, when a node receives a message, it does not always reemit it immediately. In each cycle a node performs some randomization and the message is kept in the node buffers with a probability p_{HOLD}. It results that the message can be reemitted immediately after having been received, but may also be reemitted several cycles later. As a result, every broadcast propagates differently and consequently the routes are distributed more randomly. Note that when the IOP emits *m* broadcasts, it may ultimately receive *m* routes for each node. This is unnecessary, as the IOP only needs one route for each node.

Method 2

In the method 1 described above, the *m* RM messages are broadcast and delayed randomly in each router. We assume that each node has an unique address, which may be stored in node's ROM memory. We modify the rules of flooding so that not all nodes respond to each

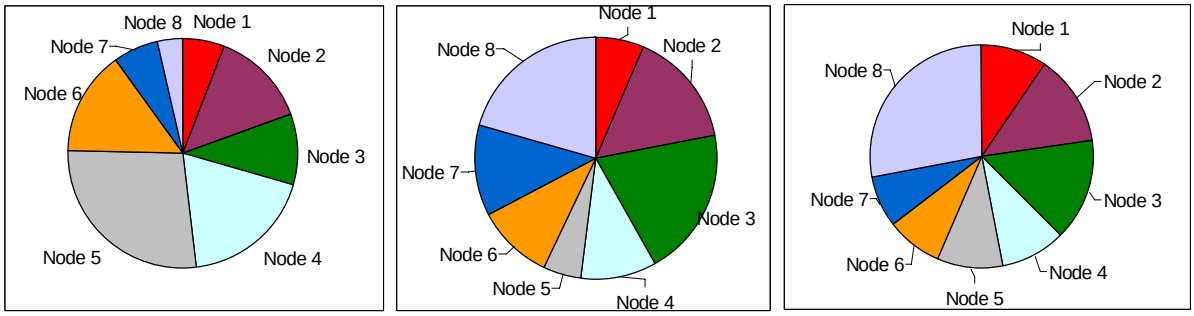
broadcast. The responding nodes are selected depending on their network address. Consider for example that 4 broadcasts are sent and each node responds when the last two bits of its address correspond to broadcast number (0, 1, 2 or 3). In this view:

Nodes 0, 4, 8, 12….etc will respond to broadcast 0
Nodes 1, 5, 9, 13….etc will respond to broadcast 1
Nodes 2, 6, 10, 14….etc will respond to broadcast 2
Nodes 3, 7, 11, 15….etc will respond to broadcast 3

Consequently, only 25% of nodes respond to each broadcast in this example. Of course, the number of broadcasts may be enlarged to increase the randomness of routes even more. As a result, the IOP will always have one route to every node. This solution needs *m* times less memory space in the IOP than the previous one.

The probability of withholding the message by a node (p_{HOLD}) and the number of broadcasts (*m*) were parameters in our study. Again, we performed an analysis of the traffic distribution in the 30x30 network. We analyzed the routes gathered by the IOP and calculated the percentage of routes which pass through the eight nodes separated by 2 hops from the IOP (see Fig. 45).

We study below the case of m=4 and p_{HOLD}= 5/6. We generate randomly the clock phases for all nodes at the beginning of each trial.



Std. Dev. = 6.0 Std. Dev. = 4.7 Std. Dev. = 4.6

Fig. 47: Distribution of traffic passing through eight nodes located two hops from IOP with route randomization method based on random delay. The number of multicasts is 4 and the probability of delaying a message is 5/6 (method 2)

When comparing Fig. 46 and Fig. 47, it can be easily seen that routes are distributed much more equally that in the case of one broadcast with no randomization. To increase randomness (decrease standard deviation) we changed the number of broadcasts and the probability to hold a message $p_{HOLD.}$ The simulations have shown that the best results are obtained for p_{HOLD}=1/2 and a large number of broadcasts.

The graphs below present the results for p_{HOLD}=1/2 and m=6.

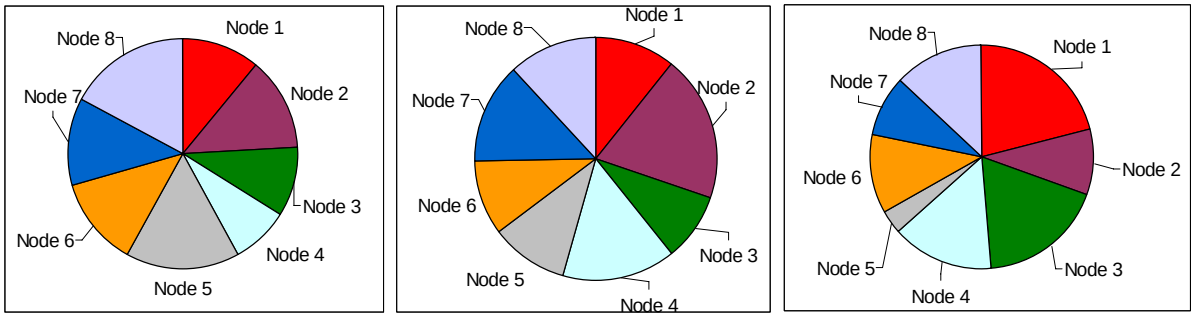


Fig. 48: Distribution of traffic passing through eight nodes located two hops from IOP with route randomization method based on random delay. The number of multicasts is 6 and the probability of delaying a message is 1/2 (method 2).

As we can see, the results are very good. Unfortunately, the proposed method has one significant disadvantage. The use of random numbers in hardware might be cumbersome and in reality the results are only quasi-random. It is then difficult to predict if real results will be as good as simulated results. Therefore, we invented another method which does not use random number generation and, therefore, is more suitable for hardware implementation.

Method 3

This method does not use the generation of random numbers and, therefore, it is more suitable for hardware implementation. As previously, the IOP sends m broadcasts and each broadcast message incorporates its number. When a node receive a copy of broadcast message, it sums its address with the broadcast number and the last two bits of the sum are used to determine the number of waiting cycles before reemitting the message. More precisely, if last two bits are “00”, the message is reemitted without waiting cycles, and similarly “01” corresponds to 1 waiting cycle, “10” to two waiting cycles and “11” to three waiting cycles. This way we achieve quasi-random diffusion because every broadcast propagates differently. Consider a simple example.

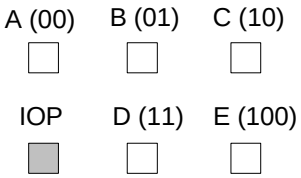


Fig. 49: Fragment of the network. The numbers represent exemplary node addresses

Let us assume the node addresses as in the figure above and let us consider four broadcasts, identified by the number m=0, m=1, m=2 and m=3. Now, how will each broadcast propagate? Consider Fig. 50 as an example. Let us see how messages will be routed from IOP to node C. According to the rule stated above, we calculated the waiting cycles for each node for each four broadcasts (see the numbers above each node). Clearly, the message will propagate along the route with the smallest sum of waiting cycles.

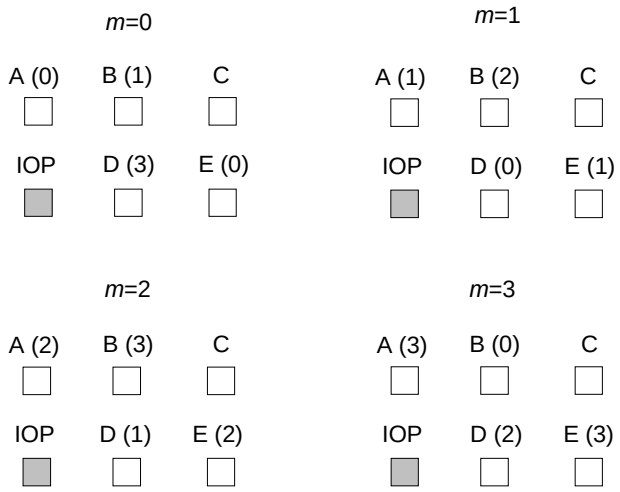


Fig. 50: Analysis of the quasi-random propagation. Numbers in brackets represent waiting cycles

We can see that depending on the broadcast number, the first copy of the message will propagate differently from IOP to node C:

- for *m*=0 route: IOP-A-B-C,
- for *m*=1 route: IOP-D-E-C,
- for *m*=2 route: IOP-D-E-C,
- for *m*=3 route: IOP-D-B-C.

All three possible routes have been used (one of them was used twice). This example clearly shows that this method offers a randomization of routes passing through the network. Moreover, this method is fairly simple to implement in hardware. To measure the randomness of the routes, we performed the same simulations as previously.

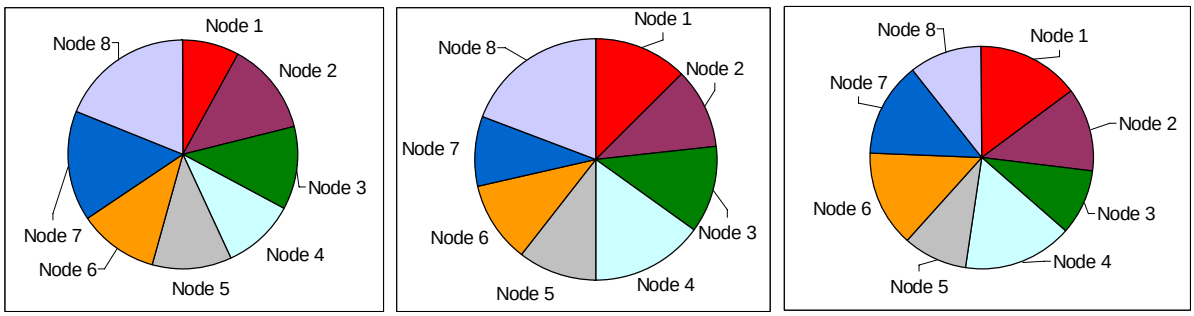


Fig. 51: Distribution of traffic passing through eight nodes located two hops from IOP with route randomization method based on our quasi-random propagation method (method 3).

The results are not only satisfactory, but surprisingly good considering the fact that they were obtained without using any randomizing function. Nevertheless, the obtained results are much better than the results of the random delay method with four broadcasts (see Fig. 47). This proves that the simple quasi-random propagation method described above is very effective.

Although the distribution of routes slightly fluctuates, this solution guarantees that the traffic passing through eight nodes located 2 hops from the IOP will be balanced. It must be emphasized that the initial broadcasts are only responsible for finding routes to enable chip operation, so they will not increase network traffic during operation. Surely, periodical broadcasts might also be necessary at runtime, to discover additional permanent faults which might have occurred. Note that at runtime the IOP may gather traffic information from nodes; consequently the routes can be configured in such a way that the network traffic remains balanced.

4.3 Chip sorting and validation

One of the most important tasks after fabrication of a multiprocessor chip is the evaluation of its performance. Needless to say, the number of defects has a direct impact on chip performance. In the first approximation, one may think that the degradation of processing power of a defective chip is fairly simple to calculate: if the core failure probability is $p_{f,N,}$, the chip performance, i.e. its processing power, will be degraded with high probability by $p_{f,N}$. However, we have already shown that the number of accessible nodes is lower than the number of non-defective nodes. Moreover, in the multicore chips that we study, the impact of defective nodes must also be evaluated at the communication level. It is clear that the defective nodes located near the IOP have a great impact on the IOP communication performance as considered in section 4.2.3 page 61.

In what follows, we introduce two additional sorting approaches which globally take into account the preservation of routes in the array and the allocation performance to distant nodes. The evaluation of both the communication and the computing performance enables us to make a decision at the end of the fabrication process whether a defective chip should be validated or discarded.

4.3.1 Node importance factor method

Principle

In this approach, the communication between the IOP and nodes is crucial for chip operation, as it allows allocating tasks (see section 5.1, page 95) and maintaining cache coherence (see section 5.3.1, page 104). In Appendix D we present the analytical calculation of a parameter which we called node importance factor which represents the “importance” of a node to the communication in the network. The method takes into consideration the impact of the defective nodes positions on the node-node communication bandwidth. It is based on a simple assumption that the average impact of a node on communication is expected to be proportional to the number of routes which pass through this node. To keep the presentation simple, the analytical calculations of the importance factor are presented in Appendix D.

Eq. 50 page 133 describes the node importance factor in a general way and, consequently, it is suitable to calculate the impact of defective nodes on any inter-node communication. For IOP-node communication, which is a special case of Eq. 50, we can use a simplified formula to calculate the importance factor of a node of coordinates (*i*,*j*) to the communication with the IOP of coordinates (*s*,*t*):

Eq. 10

$$imp_{N,IOP}(i,j) = \frac{\sum_{v=1}^{M}\sum_{w=1}^{N} \frac{C_{abs(v-i)+abs(w-j)}^{abs(w-j)} \cdot C_{abs(s-i)+abs(t-j)}^{abs(t-j)}}{C_{abs(s-v)+abs(t-w)}^{abs(t-w)}}}{\sum_{i,j} imp_{IOP}(i,j)}$$

Note that the conditions specified in Eq. 48 (see Appendix D) must also be valid. From the original definition of the importance factor given above, we may also conclude that Eq. 10 represents the fraction of IOP routes which will disappear if the node of coordinates (*i,j*) is defective.

So, we introduce the connectivity factor of the network which measures the communication ability between fault-free nodes and the IOP. We define here the network connectivity as the sum of all node importance factors of all network nodes. It represents the top network connectivity (fault-free network) whereas the sum of all non-defective node importance factors is the real network connectivity. The network connectivity factor C_F can be defined as the real network connectivity divided by maximum network connectivity:

Eq. 11

$$C_F = \frac{\sum_{i_{ND}, j_{ND}} imp_{N,IOP}(i_{ND}, j_{ND})}{\sum_{i,j} imp_{N,IOP}(i,j)}$$

where i_{ND} and j_{ND} represent the coordinates of non-defective nodes.

In a defective network, C_F ranges between 0 and 1. C_F =1 means that the network connectivity is equal to maximum network connectivity (network without holes). Of course, in general, the higher the number of defective nodes (holes), the lower the connectivity. However, the impact of holes on the network connectivity is not uniform. If most of holes are located near the network edges (far from the centre), the network has a high connectivity factor. Contrarily, if most of holes are located close to the IOP, the connectivity factor is low.

It must be emphasized that in this method, we assume that the impact of *n* defective nodes on the network connectivity factor C_F is equal to the sum of the impacts of all *n* defective nodes when analyzed independently. This assumption is based on the idea that the impacts of particular holes are not correlated. Further studies must be performed to evaluate network connectivity without making the above assumption.

Simulation results

Let us analyze the simulation results in Fig. 52 which calculate the distribution of the connectivity factor for a given percentage of defective nodes. For each curve 1 million simulations have been performed in a 21x21 network with 441 nodes with one IOP in the centre of the network.

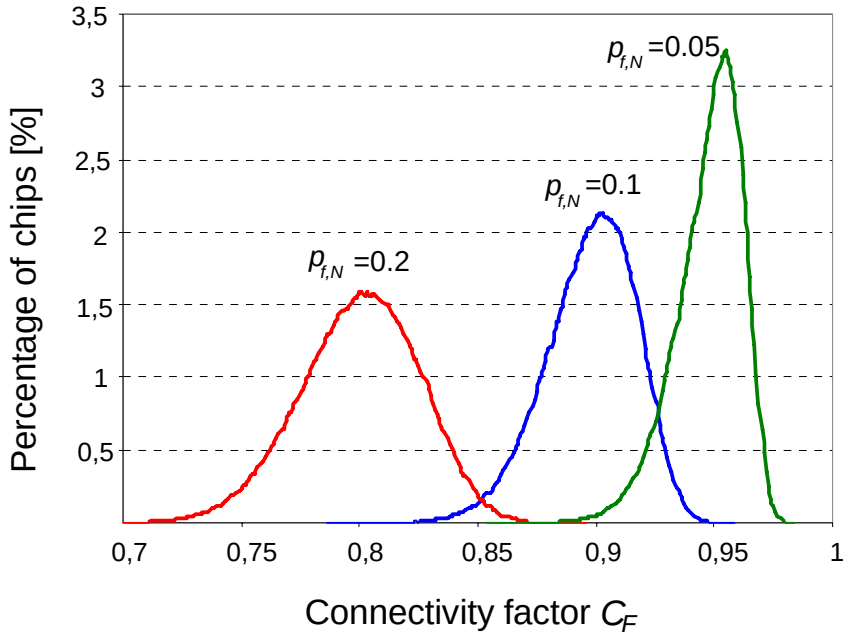


Fig. 52: Distribution of connectivity factor among chips with various values of $p_{f,N}$

Every curve represents the distribution of the connectivity factor C_F for different percentage of defective nodes. We will analyze below the results for $p_{f,N}$ =0.1 (the middle curve).

The results are distributed almost symmetrically around an average value. We can see that the average C_F is equal to the percentage of non-defective nodes:

Eq. 12

$$C_{F,average} = 1 - p_{f,N}$$

For $p_{f,N}$ =0.1 the average C_F is 0.9 (middle curve). However, the C_F values vary from 0.825 to 0.95, which means that the network with 10% of defective nodes (which constitute holes in the network) may have the connectivity factor as low as 0.825 or as high as 0.95. In other words, network with 10% holes may have the connectivity as low as an average network with 17.5% holes or as high as an average network with 5% holes. This example shows that the simplest chip sorting method just based on the consideration of the percentage of defective nodes does not always give good results: chips with good connectivity may be discarded or, in the opposite case, chips with very low connectivity could be validated. Therefore, we suggest using a more sophisticated method based on calculating the network connectivity factor. In this approach, only the chips with a connectivity factor C_F larger than a user-defined threshold $C_{F,min}$ are validated for deployment.

Every chip with $C_F < C_{F,min}$ is discarded and chips with $C_F > C_{F,min}$ are accepted. Let us show that this method gives very different results from the method based just on simple analysis of percentage of defective nodes.

Consider the following sample graph that presents the percentage of validated chips for $C_{F,min}$= 0.8.

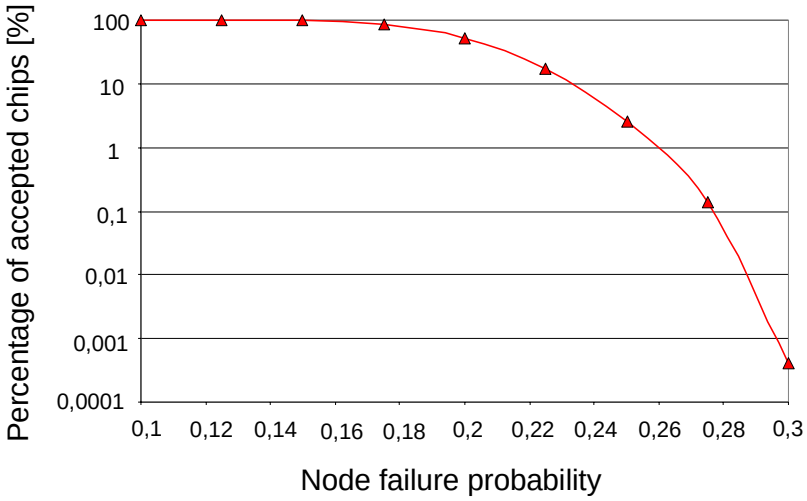


Fig. 53: Percentage of accepted chips as a function of the core failure probability

If all nodes had the same importance factor (if all nodes had the same impact on the network connectivity), the above graph would be very simple: the chips with less than 20% defective nodes would be always accepted and those with more than 20% defective nodes would be always discarded. It is not so simple with this method. There exists a non-zero probability that the network connectivity would be too low. For example we can see on the graph above that even if there are only 15% of defective nodes, still some chips do not match the connectivity criteria. On the other hand, we can safely accept some chips even with 25% of defective nodes because they have high connectivity factor. However, a question arises: is it possible to implement this method without excessive hardware overhead?

Implementation

In the crude method based on counting accessible nodes, the IOP simply broadcasts one or several RM messages to all nodes and then counts the responses of routes stored in its valid route array VRA (see the discussion on route randomization in section 4.2.4, page 64). The number of responses equals the number of non-defective accessible nodes. In this new method, the IOP (or external equipment used for chip validation) must not only count the responses but also it has to calculate the network connectivity factor. However, there is a fairly easy way to perform this analysis. The IOP should have a pre-defined importance factor for all nodes stored in memory. Thus:

1. The IOP broadcasts the RM message and stores the routes in the VRB, as in the simple method.
2. It calculates the position in the network of each responding node from the analysis of the routes in the VRB.
3. The IOP sums up all importance factors for non-defective nodes to calculate network connectivity factor and compare it to the pre-defined minimal C_F value. A special output bit may be implemented which will indicate if a chip should be approved or discarded. Alternatively, the IOP can store the calculated value of chip connectivity which can be later read by some external device.

The new proposed method requires then only an array which stores the pre-defined nodes importance factors. Every importance factor value can be stored with several bytes of data.

Therefore, even in a large network which comprises as many as 1000 nodes, the memory space needed for our method is not large.

4.3.2 Allocation capacity method

Another method for chip validation may be based on analyzing the number of nodes accessible from the IOP at distance *d*. Clearly, when the IOP is allocating tasks, the presence of a large number of close cores will speed up the processing whereas the processing power will be reduced when the majority of cores are located far from IOP.

Note that in a defect-free infinite mesh network, the number of nodes k_N at a distance *d* is always $k_N=4 \cdot d$. So, it grows linearly with the distance. In the finite network of size *N*x*N*, the number of nodes increases first linearly with distance *d*, but when *d* reaches *N*/2 the number of nodes starts to decrease. This property can be easily seen in Fig. 54: the curve $p_{f,N} = 0$ represents the results for defect-free network. In the presence of defective cores randomly distributed in the network, we used the simulator MASS to calculate the average number of cores at distance *d*, as analytical calculations are no longer possible. In each iteration, the program calculates the number of nodes found at distance *d* from the IOP located in the centre of the network. The mean number of reachable nodes is deduced by averaging the results of several thousand iterations.

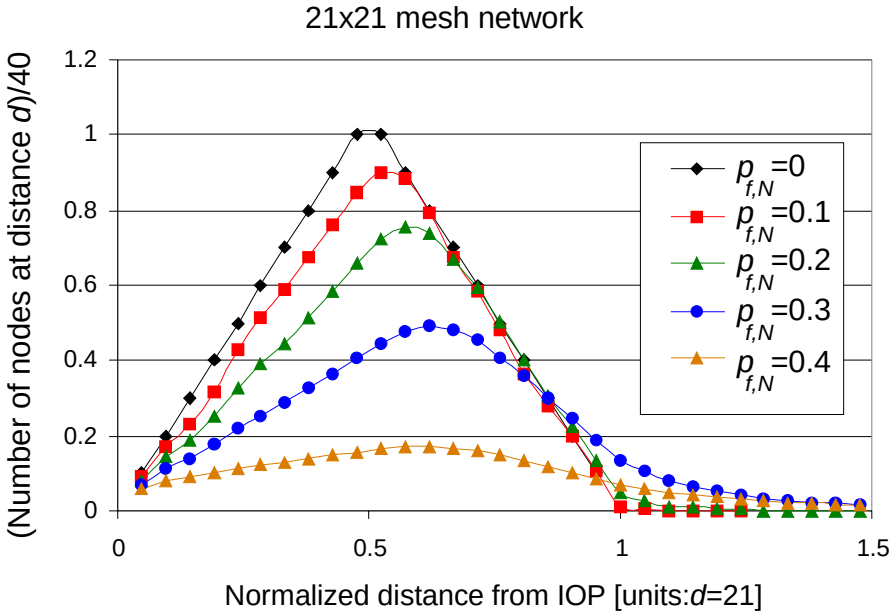


Fig. 54: Distribution of number of nodes in function of the distance from IOP in a 21x21 network

Fig. 54 presents the number of nodes as a function of the distance from the IOP. For both axes we used normalized values. We can observe that the number of nodes increases linearly with the distance until the normalized distance d_N equals 0.5. Of course, the higher the percentage of defective cores, the lower the linearity factor. It is also worth mentioning that the number

of nodes at the distance d_N does not decrease proportionally with $p_{f,N}$. This effect is of course caused by the fact that a defective node does not only diminish the number of good nodes at the distance d, but makes longer the routes to other good nodes in the network.

However, the results presented above cannot be directly used to decide whether the chip may be validated or not. A definitive parameter which would measure the IOP allocation capacity, (i.e. the ability to allocate as many tasks as possible in a given time) is needed to enable easy chip validation. In other words, the above curve which represents the number of routes at the distance d has to be transformed into a single value defining the IOP allocation capacity. This transformation is impossible without making some assumptions on the allocation process. The detailed analysis of allocation procedure is presented in chapter 5.1, page 95, but here we will present a simplified vision which will allow us to calculate the top allocation capacity. Consider an allocation process between the IOP and a node of index i. During time T_S, the IOP is able to allocate n_i tasks on this node, where n_i is given by the formula:

Eq. 13
$$n_i = \frac{T_S}{T_{ai}}$$

where T_a is the time needed for sending the task to the node and for receiving the node's response. Note that we neglected the execution time which is perfectly reasonable, as our goal is to calculate the allocation capacity and not to describe the real network operation. Regardless of the network topology, T_a may be viewed as the round trip time needed by a message to be transferred to a node and to come back to the IOP. Thus:

Eq. 14
$$T_{ai} = 2 \cdot d_i \cdot T_H$$

where d_i is the number of hops between a node and the IOP and T_H is the time needed to transfer a message between two nodes (i.e., the time needed for one "hop").

Thus, the maximum number of tasks which may be sequentially allocated on this node during the time interval T_S simply reads:

Eq. 15
$$n_i = \frac{T_S}{2 \cdot d_i \cdot T_H}$$

Consider now a chip with N_N nodes. To keep our calculations simple, we assume that the IOP can send simultaneously multiple allocation requests to different nodes. Then, we may calculate the overall number of tasks n_T which may be allocated in the network during time T_S:

Eq. 16
$$n_T = \sum_{i=1}^{N_N} n_i = \sum_{i=1}^{N_N} \frac{T_S}{2 \cdot d_i \cdot T_H} = \frac{T_S}{2 \cdot T_H} \cdot \sum_{i=1}^{N_N} \frac{1}{d_i}$$

This summation should be calculated for all non defective and accessible nodes in the grid. As the factor $\frac{T_S}{2 \cdot T_H}$ is constant for a given chip, the top number n_T of allocated tasks depends only on the following parameter which we called the top allocation capacity (TAC):

Eq. 17
$$C_A = \sum_{i=1}^{N_N} \frac{1}{d_i}$$

This formula is not a surprise, as the more nodes in the network, the higher C_A, but nodes located far from the IOP contribute less to the TAC than close nodes. For the sake of argument, let us consider an example. Let us calculate the TAC for a 5x5 grid including 4 faulty cores for the three following mappings of defective cores:

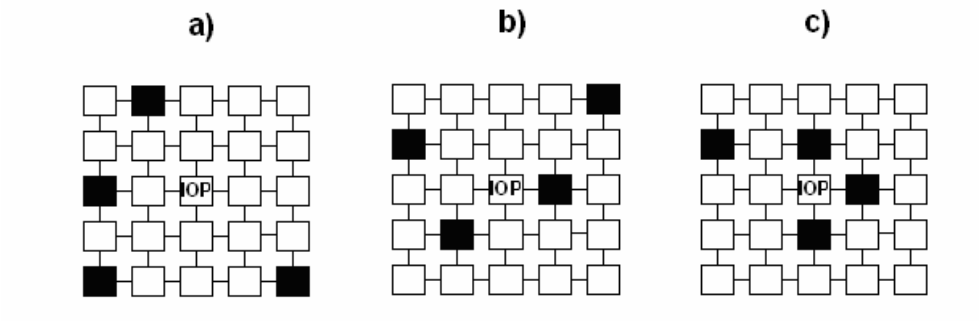


Fig. 55: Three sample 5x5 networks with different distribution of defective nodes

In network (a), there are 4 nodes at distance 1, 7 nodes at distance 2, 7 nodes at distance 3 and 2 nodes at distance 4. Consequently, the formula for allocation capacity C_A in this network reads:

Eq. 18
$$C_{Aa} = 4 \cdot 1 + 7 \cdot \frac{1}{2} + 7 \cdot \frac{1}{3} + 2 \cdot \frac{1}{4} \cong 10.33$$

Similarly, the allocation capacity for networks (b) and (c) reads:

Eq. 19
$$C_{Ab} = 3 \cdot 1 + 6 \cdot \frac{1}{2} + 7 \cdot \frac{1}{3} + 4 \cdot \frac{1}{4} \cong 9.33$$

Eq. 20
$$C_{Ac} = 1 \cdot 1 + 3 \cdot \frac{1}{2} + 3 \cdot \frac{1}{3} + 4 \cdot \frac{1}{4} + 2 \cdot \frac{1}{5} + 4 \cdot \frac{1}{6} + 2 \cdot \frac{1}{7} + 1 \cdot \frac{1}{8} \cong 5.98$$

The network (a) has clearly the highest TAC (that is intuitively no surprise as the defective nodes are located near network borders) and the third network the lowest TAC (as defective nodes are located very close to the IOP). It is particularly interesting to point out that the allocation capacity of network (c) is disastrously low. For example, consider a network (a) with additional six defective nodes, as shown in the figure below:

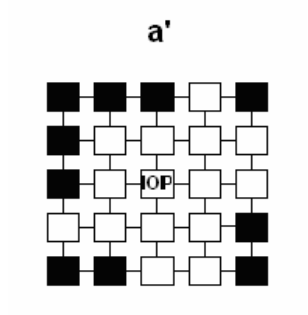


Fig. 56: Sample 5x5 network with 10 defective nodes

Now the new allocation capacity is calculated as follows:

Eq. 21
$$C'_{Aa} = 4 \cdot 1 + 6 \cdot \frac{1}{2} + 4 \cdot \frac{1}{3} \cong 8.33$$

Although there are about 40% of defective nodes in network (a') compared to only 20% in network (c), it has still a much higher allocation capacity! We can see that the state of the nodes located close to IOP is extremely important in this metrics. If IOP's neighbors are defective, the allocation capacity is dramatically reduced.

The analysis presented in this chapter shows that the validation of chips may be based on various criteria. Depending on the parameters that are taken into consideration, different validation decisions are possible. One criterion may favor increasing the number of accessible cores whereas others may be based on reducing allocation time. In section 5.2 we shall compare the impact of the number of accessible nodes and the allocation time on chip performance. However, it must be emphasized that the final choice of the validation method will heavily depend on an actual chip implementation and its operation mode.

4.4 Multiport architectures

In the previous sections we considered single-IOP architectures. However, in the long term, as the number of cores in the network increases, one must consider implementing several ports to ensure the scalability of the network. More precisely, the IOPs constitute critical elements of the system because all communications with outside world must be transferred via the ports. This problem is similar to the bus bottleneck in the Symmetric Multiprocessor systems [74]. Therefore, the only solution is to increase the number of ports which in turn introduces another important problem. Since all ports are necessary for maintaining high chip performance, the designer must ensure that IOPs are not defective. Clearly, with one IOP this should not be a problem but as the number of IOPs increases, their protection becomes more and more difficult. There is a kind of vicious circle in this approach: on one hand, it is necessary to increase the number of IOPs and on the other hand the probability that some of IOPs will be defective grows as we multiply them! For example, in a 4-IOP chip when the IOP failure probability is $p_{f,IOP} = 0.2$, the probability that all IOPs operate correctly is only $(0.8)^4 \approx 0.41$. Consequently, very few chips would operate correctly if no protection mechanisms were implemented. Let us analyze now the problem of the IOP protection. In the next section we study how the probability that all ports work depends on the redundancy applied in the IOP region.

4.4.1 Protection of the IOPs

Several complementary strategies exist to directly protect the IOPs. One may use:

- Hardened technology at the component level [75]. This approach is complex when it is necessary to mix several technologies.

- N-modular circuit redundancy [76]. The simplest solution is to place several IOPs in parallel, to test them with an external controller, and to select one of them through multiplexing and demultiplexing circuits (MD). This kind of a circuit will be called a redundant IOP (RIOP). For example, Fig. 57 shows the internal architecture of a RIOP based on 3 IOPs (thus, with triple modular redundancy), with 4 grid interconnects (labeled N, E, S, W) and one external interconnect Xtern.

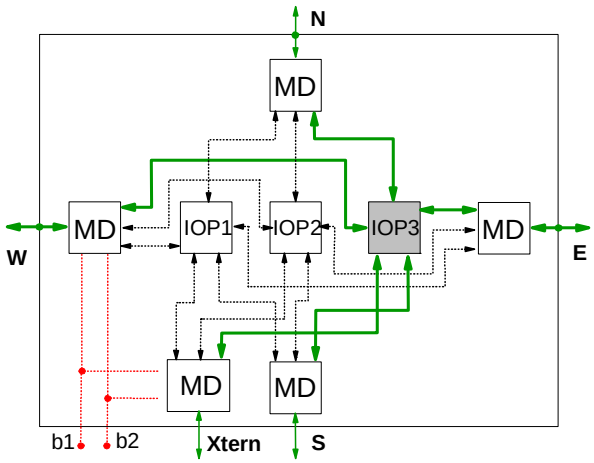


Fig. 57: Simple RIOP architecture designed to reduce the failure probability of the input/output port. Xtern is the communication channel with the external environment. b1 and b2 are two digital inputs to select one of the 3 IOPs, following external test. Bold lines show the interconnects which are activated when IOP3 is selected.

The most natural idea is to validate a chip for operation at start-up, if all RIOPs are functional (remember that each RIOP works if at least one of its IOP is not defective). However, it must be emphasized that this method does not protect the RIOP against transient faults during operation! Redundancy is necessary at runtime, possibly with level 2 for error detection and level 3 for error correction. Thus, we suggest in this work validating the chip if at least 3 out of R IOPs (where R is the total number of redundant IOPs) are fault-free at start-up in each RIOP. These three IOPs will provide a necessary redundancy at runtime to cope with transient faults. The probability to validate a chip according to this criterion is simply:

Eq. 22

$$p_{W,IOP} = \left[\sum_{i=3}^{R} \binom{R}{i} \left(1 - p_{f,IOP}\right)^{i} \; p_{f,IOP}^{R-i} \right]^{N_{IOP}}$$

where N_{IOP} is the number of RIOPs and $p_{f,IOP}$ the failure probability of an IOP. An implicit assumption beyond the triplication idea is that the failure probability of the multiplexing/demultiplexing circuitry (MD, see Fig. 57) should be much smaller (and even negligible) compared the failure probability of an IOP. Since an IOP is a much more complex circuit than MD, the above assumption seems quite reasonable.

It is obvious that the traffic concentrates around each RIOP and that failures of nodes close to the RIOP will dramatically limit the communication bandwidth, when contrarily, failures of nodes far from the RIOP will have little effect on communication. This problem was already discussed in section 4.3, page 71 in relation with the question of chip sorting and validation. In first approximation, we assume that the communication bandwidth will be proportional to the number of fault-free nodes adjacent to the RIOP. Therefore, we analyze the idea of increasing the RIOP connectivity n_C to limit the reduction of the IOP communication bandwidth due to the failure of direct adjacent nodes. Possible examples of the local modifications of the grid topology around each RIOP are shown in Fig. 58. Note that the connectivity of the cores adjacent to the IOP is not changed, to maintain regularity of the grid.

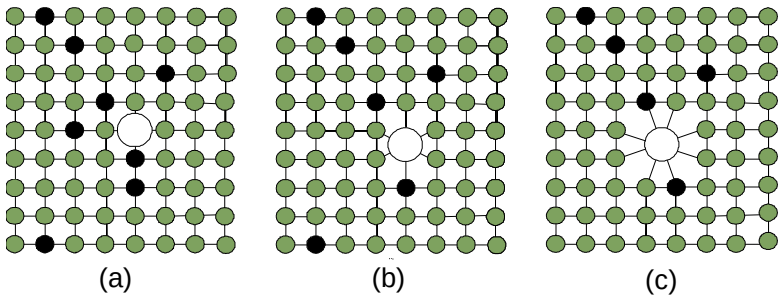


Fig. 58: Examples of local changes of the grid topology around the IOP increase the number of adjacent cores. The IOP connectivity is increased from 4 (in a) to 6 (in b), and to 8 (in c).

The constraint of maintaining high bandwidth leads sorting and validating chips as a function of the number of non-faulty adjacent nodes around each RIOP. The formula for the probability $p_L(k, n_C, p_{f,N})$ that at least k adjacent neighbors (out of n_C) are not defective around each RIOP reads:

Eq. 23

$$p_L(k, n_C, p_f) = \left[\sum_{i=k}^{n_C} \binom{n_C}{i} (1 - p_{f,N})^i \, p_{f,N}^{n_C - i} \right]^{N_{IOP}}$$

where $p_{f,N}$ is the node failure probability.

The probabilities $p_{W,IOP}$ and $p_L(k, n_C, p_{f,N})$ are plotted in Fig. 59 and Fig. 60 for a 4-port architecture (i.e., N_{IOP}=4).

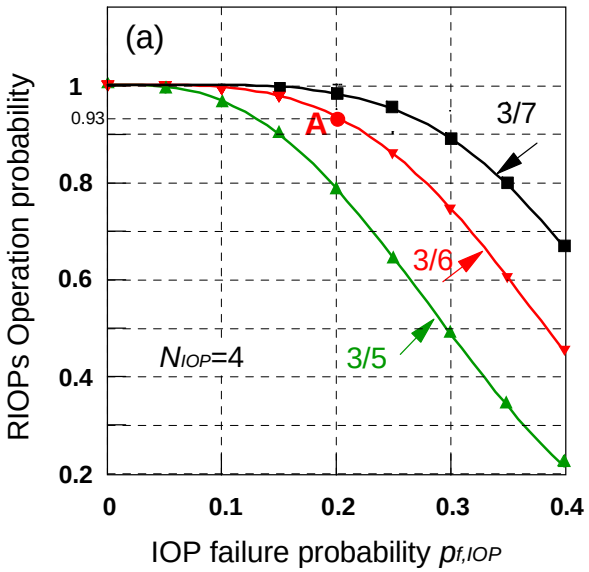


Fig. 59: Probability in a 4-port chip that at least 3 IOPs work in each RIOP versus the IOP failure probability.

Note that each curve is indexed with a label *N/D*. In the above figure the numerator *N* is always 3, which represents the number of fault-free IOPs in the RIOP and the denominator *D* represents the redundancy used to build the RIOP. For instance, point A in Fig. 59 (with coordinates X_A and Y_A) shows that when the redundancy is R=6 and the IOP failure probability $p_{f,IOP}=X_A$=0.2, the probability $p_{W,IOP}$ that each RIOP works correctly (i.e., that at least 3 out of 6 IOPs are fault-free) is approximately Y_A=0.93.

Similarly, in Fig. 60 the fraction numerator is the number of fault-free cores adjacent to the RIOP and the denominator $D=n_C$ is the RIOP connectivity. We studied three cases D=4, D=6 and D=8 in accordance with the topologies shown in Fig. 58. For instance, point C in Fig. 60 means: When the RIOP connectivity is $n_C = 8$, then the probability to have at least N=3 fault-free cores adjacent to each RIOP (out of 8) is equal to 0.95, when the core failure probability $p_{f,N}$ is 0.25.

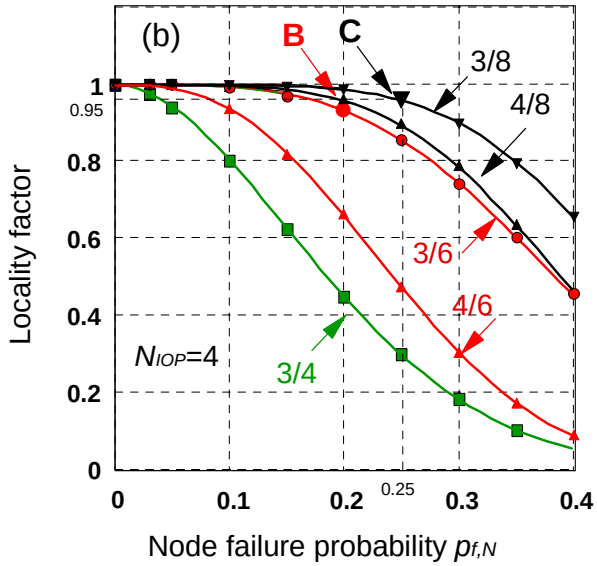


Fig. 60: Probability in a 4-port chip that each port is surrounded by a minimum number of fault free cores, versus the core failure probability $p_{f,N}$.

Ultimately, the results presented in the two above figures may be combined to estimate the overhead (in terms of silicon area) necessary to protect the IOPs and their environment using redundant techniques. For example, points A (in Fig. 59) and B (in Fig. 60) jointly mean: when the failure probabilities of IOPs and cores are $p_{f,IOP}= p_{f,N}$=0.2 and when the grid has 4 ports, each one being connected to 6 neighbors, the probability that 3 out of 6 IOPs work in each RIOP is p_1=0.93 and the probability that at least 3 out of 6 direct adjacent neighbors of each RIOP work is approximately p_2=0.93. Thus, the probability to fabricate a chip fulfilling these constraints is approximately $p_1 \cdot p_2 \approx 0.865$. The protection penalty, i.e., the fraction of additional silicon area sacrificed to protect the RIOPs, is approximately:

Eq. 24

$$Q = \frac{(R-1)N_{IO} \cdot A_{IO}}{N \cdot A}$$

where N is the number of cores, A_{IO} the the IOP module size and A the core size. Considering the above equation, we have calculated the chip area overhead as a function of the ratio A_{IO}/A. We have analyzed the chip with the following parameters: 300 cores, 4 RIOPs, each connected to 8 adjacent cores. Additionally, we assumed that:

- At least 3 (out of total 8) IOP neighbours must be fault-free to protect communication bandwidth, for each IOP.
- Port validation yield calculated from Eq. 22 and Eq. 23 must be at least 80%.
- The failure probability of each IOP module or each node is proportional to its size. So the ratio of areas A_{IO}/A is also the ratio of failure probabilities.

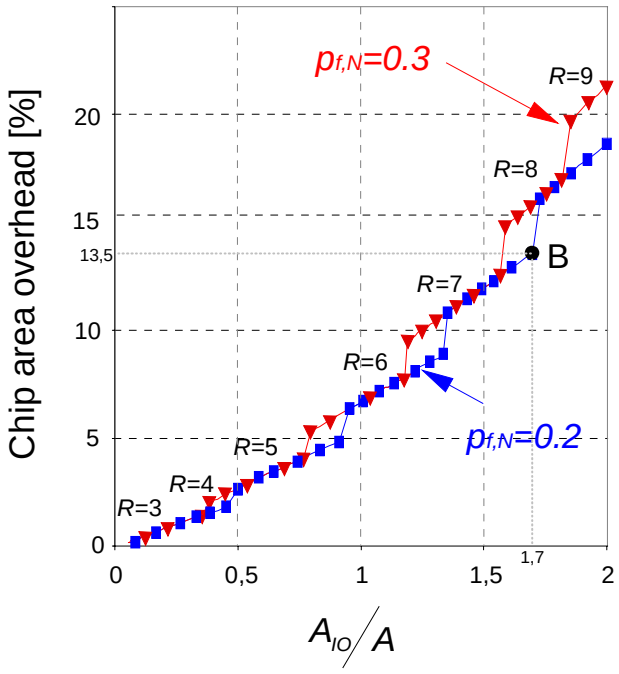


Fig. 61: Chip area overhead versus the relation between IOP and core sizes for two node failure probabilities $p_{f,N}$=0.2 *and* $p_{f,N}$=0.3

The X axis in Fig. 61 represents the relation A_{IO}/A, the Y axis is the chip area overhead and the letters *R* above both curves state the redundancy needed to protect the ports considering the assumptions stated above. Let us now consider a particular point in the above figure. Point B means that if the node failure probability $p_{f,N}$ is 0.2 and the IOP module is 1.7 times larger than a core, 13.5% of chip area is sacrificed to protect the ports. In general, we can see that the IOP module size should be preferably smaller than a core to keep the area overhead below 7%. If the IOP module is larger than the core, the overhead quickly become prohibitive, reaching 20% when A_{IO}/A=2. Surprisingly, the difference between two curves $p_{f,N}$=0.2 *and* $p_{f,N}$=0.3 are quite small.

Note that protecting the IOPs (using redundancy) seems to become extremely difficult when the failure probability exceeds $p_{f,N}$ = 0.3, especially if the IOP area is larger than that of the node (for instance, when the ratio of areas $A_{IO}/A>2$). The extrapolation of the data displayed in Fig. 61 shows that more that 20% (perhaps 30%) of the chip area would have to be sacrificed. This estimation confirms our previous conclusion that it would be difficult to fabricate chips with a high validation yield, when the node failure probability exceeds typically 0.2.

4.4.2 Positioning of the IOPs

The integration of several IOPs poses a natural question: what should be the positions of the ports in the grid. In a single IOP grid the natural choice is the center of the network whereas choosing the IOP locations in a multiple-IOP grid is not an obvious decision.

Where to place the IOPs is technology dependent, and many solutions are possible. For a long time, the most common solution consisted in placing the IOPs on the chip perimeter as shown for instance in Fig. 62, where 4 IOPs labeled with the letter A are positioned symmetrically in the middle of each edge.

However, this solution is not satisfactory because for regular grid topologies (square, rectangle), the number of places on the perimeter scales as the square root of the number of nodes in the network with the consequence that the traffic per IOP inexorably grows when increasing the number of nodes! Today, due to the development of microelectronics, ports may be placed anywhere in the grid. For instance in Fig. 62, which shows an example of a 11x11 grid, we consider three other IOPs positions (labeled with letters B,C and D). Of course, positioning the IOPs in the corners is not a suitable solution but has been included in our study for comparison purposes.

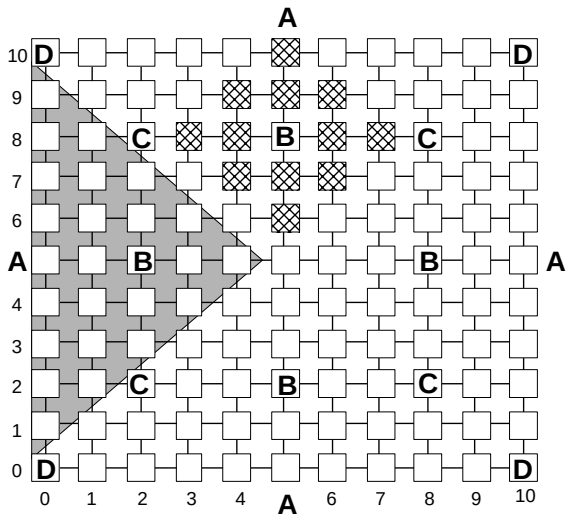


Fig. 62: Example of multiport grid architecture. The node connectivity is 4. The four points labelled A, B, C or D show four possible positions of the IOPs.

We studied in the four next figures the core accessibility in a multiport 2-D mesh. In each iteration, the simulator first randomly generates a fraction $p_{f,N}$ of holes in the network, which represent the faulty cores logically disconnected following the mutual-test process (see section 3.2.3). Then, each IOP emits a route request message (RM), which is broadcast across the defective network. The simulator sorts the cores as a function of the number of RMs they receive, at most 4 when the core is shared by all IOPs, and none in the opposite limit when it is inaccessible. Then the program calculates the average fraction of cores simultaneously accessible by 4, 3, 2, 1 or no IOP.

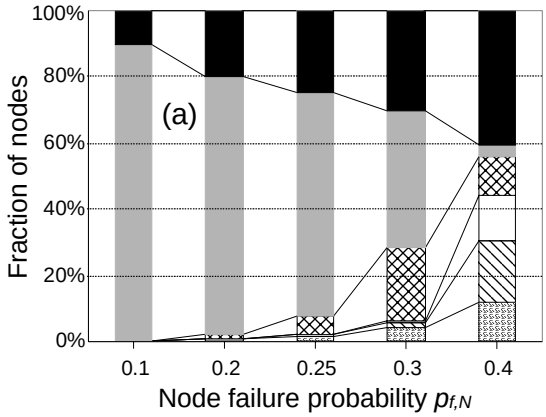


Fig. 63: Fraction of cores shared by the different IOPs in a 11x11 grid. ■ : Defective cores, ▒ : valid cores shared by 4 IOPs, ▩: valid cores shared by 3 IOPs, □ : valid cores shared by 2 IOP, ▨: valid cores contacted be one sole IOP, ▦: inaccessible cores. The IOPs are placed in positions A as shown in Fig. 62.

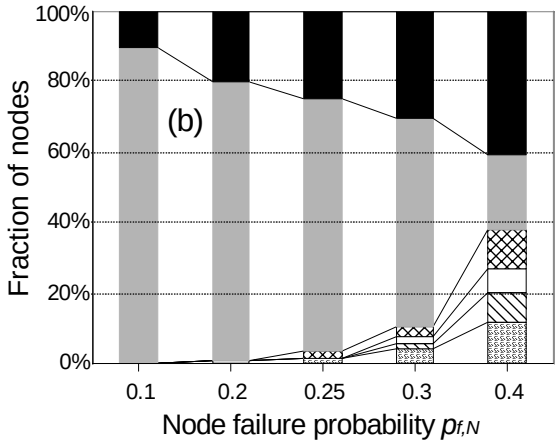


Fig. 64: Fraction of cores shared by the different IOPs in a 11x11 grid. ■: Defective cores, ▒ : cores shared by 4 IOPs, ▩: cores shared by 3 IOPs, □ : cores shared by 2 IOP, ▨: cores contacted by one sole IOP; ▦: inaccessible cores. The IOPs are placed in positions B as shown in Fig. 62

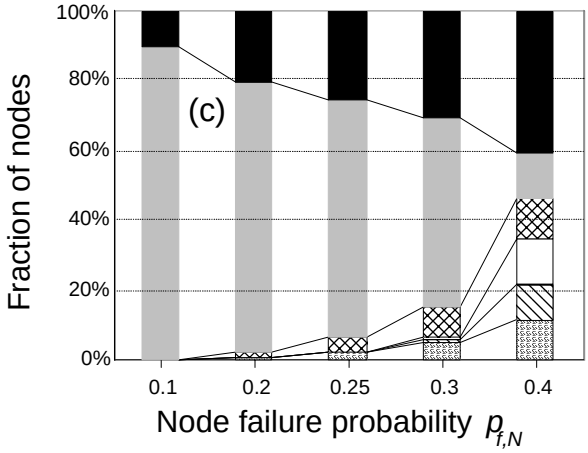


Fig. 65: Fraction of cores shared by the different IOPs in a 11x11 grid. ■: Defective cores, ▒ : cores shared by 4 IOPs, ▩: cores shared by 3 IOPs, □ : cores shared by 2 IOP, ▨: cores contacted by one sole IOP; ▦: inaccessible cores. The IOPs are placed in positions C as shown in Fig. 62.

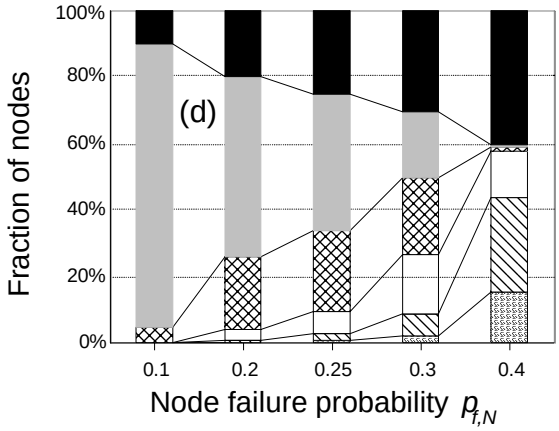


Fig. 66: Fraction of cores shared by the different IOPs in a 11x11 grid. ■: Defective cores, ■: cores shared by 4 IOPs, ▨: cores shared by 3 IOPs, □: cores shared by 2 IOP, ▧: cores contacted by one sole IOP; ▩: inaccessible cores. The IOPs are placed in positions D as shown in Fig. 62.

In Fig. 63 to Fig. 66 the four IOPs are placed in the positions labeled with the letters A, B, C and D, respectively. The description of the different bar patterns in terms of core sharing is integrated in the captions of these figures. It turns out that for positions A, B and C**,** all valid cores can be contacted by all IOPs as long as the fraction of defective nodes is typically less than 25%. Above this threshold, the position of the IOPs in the grid influences the core sharing. The results for positions D are very bad, as already for $p_{f,N}$=0.3 half of the valid cores are contacted by less than 3 IOPs. Placing the IOPs in the positions B or C (as shown in Fig. 62) very significantly increases the probability that a core is contacted by several IOPs when compared to positions A. This proves the importance of the initial IOP connectivity: in position A each port has only 3 adjacent nodes and in positions B and C each ports has 4 neighbours. Interestingly, the positions B provide slightly better core accessibility than positions C, especially for high values of $p_{f,N.}$ Consequently, in a mesh network with 4 IOPs, the positioning of ports according to the letters B should be considered as the best one and consequently, we performed the simulations with this positioning in the next section 4.4.3.

Additionally, this data confirm that above typically 20-25% of defective isolated cores in multiport network, there is a quick degradation of the connectivity between the IOPs and the valid cores.

4.4.3 Route discovery in multiport architectures

Defining metrics to characterize route discovery in multiport chips may (and should) be a complex multi-criteria optimization problem, in particular because the number of nodes which may be used by each IOP for executing tasks depends on the workload of each port. Therefore, we introduce two reachability metrics, based on two opposite operation modes. We define the reachability of an IOP as the fraction of nodes which the IOP can contact. The first metrics, which we call the Global Reachability Metrics, is in fact a generalization to what we have already described for uniport architectures (see chapter 4.2), where we consider sorting chips as a function of the number of cores that can be contacted for processing by the IOPs.

The second one, which we call the Closest-Cores Reachability Metrics, takes into account that every IOP has its own “allocation zone”. More precisely, in case of multiport operation, each IOP typically allocates its tasks to the cores which are the closest to it. The reason is that the IOPs compete for shared processing resources and the access to close cores is privileged because of the low access latency. For instance, in the defect-free 2D-mesh displayed in Fig. 62 the IOP labelled B with coordinates (2,5) would preferentially allocate its tasks to the cores inside the greyed isosceles triangle because they are farther from the other B IOPs.

Global Reachability Metrics

Let N_N be the total number of nodes in the network and $N_M(i)$ be the top number of valid cores that can be reached by the IOP of index i. Of course, this value changes from one chip to another, as it depends on the random distribution of defective nodes in the network. We characterize each chip by its reachability R_1 defined by the following ratio:

Eq. 25 $$R_1=Min\{N_M(i)|1\leq i\leq 4\}/N_N$$

Consequently, R_1 is in fact equal to the smallest fraction of nodes contacted by one of the IOPs. Therefore, in a chip with reachability R_1 each IOP is able to reach a fraction of cores in the array equal or larger than R_1. This Global Reachability Metrics (denoted RM1) is particularly adapted to the case of single-IOP operation, when there is no competition to access processing resources.

We used the simulator MASS to analyze the accessibility of nodes in the multiport defective network. We measure the node accessibility as the probability $p(R_1>R)$, i.e., the probability that a defective chip has a reachability R_1 exceeding a preset value R. To calculate $p(R_1>R)$, the simulator first generates a fraction of defective nodes in the network, then it calculates the number of accessible nodes $N_M(i)$ which can be contacted by each IOP, and finally R_1 using Eq. 25. This procedure is typically repeated several thousands times. In these simulations, we analyzed a very large 31x31 network with 4 IOPs. The IOPs are placed inside the network, similarly to the squares marked “B” in the Fig. 62. Moreover, we considered two networks: first with the IOP connectivity equal to 4 (Fig. 58a) and second with the IOP connectivity increased to 8 (Fig. 58c). Increasing the number of nodes adjacent to each IOP enables protecting the node accessibility against failures of IOP neighbours. As it has been already pointed out, failures of nodes close to the RIOP will dramatically limit the ability to contact the nodes whereas defective nodes far from the RIOP will produce no such effect.

The simulation results displayed in Fig. 67 essentially show how the increase of the fraction of defective nodes in the network (ranging from 10 to 40%) reduces the probability to fabricate chips with a pre-defined minimum global reachability.

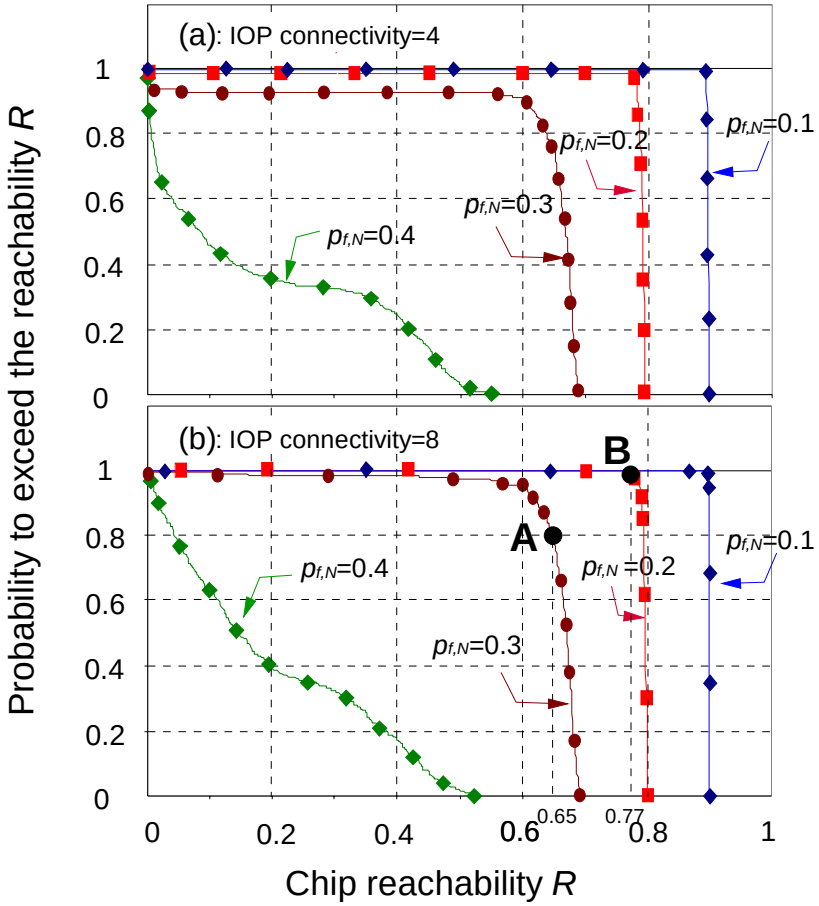


Fig. 67: Analysis of node accessibility using RM1 in the case of single IOP operation. Upper part (a): IOP connectivity $n_{C,IOP}$ = 4; lower part (b) : IOP connectivity $n_{C,IOP}$ = 8. The grid topology is a 2D-mesh.

First, note that the upper and the lower parts in Fig. 67 are almost identical. This comparison simply shows that the IOP connectivity (which is important to preserve the communication bandwidth of the IOP) has only a small impact on the long-range accessibility of nodes. A small improvement is visible in highly defective grids ($p_{f,N}$=0.3 or $p_{f,N}$=0.4) as the probability $p(R_1>R)$ for small values of R is slightly higher when IOP connectivity is increased to 8. Consequently, we may conclude that increasing IOP connectivity improves the probability of contacting close cores only.

Now let us consider one point to explain the data in detail. For example, point A means that when 30% of nodes are defective and isolated in the grid, the probability that the chip reachability exceeds 0.65 is approximately 0.8. In short, $p(R_1>0.65)=0.8$. In other words, in case of single-IOP operation, 80% of fabricated chips would be such that each IOP would be able to contact more than 65% (out of 70%) of valid nodes. Remember that an IOP cannot contact more than 70% of nodes because 30% of nodes are defective. Point B means that when 20% of cores are defective and isolated in the grid, the probability is 0.98 that the chip reachability exceeds 0.77, i.e., $P(R_{Min}>0.77)=0.98$. Thus, Fig. 67 depicts in fact the node accessibility versus the user-defined threshold. It is obvious from this figure that the node accessibility dramatically decreases when the fraction of defective and isolated cores in the network exceeds 30%.

Closest-Cores Reachability Metrics

The metrics presented in this paragraph is more realistic than the previous one because we consider the simultaneous operation of all nodes and that the workload is balanced among the IOPs.

Let $N_C(i)$ be the number of cores, which are closer to the IOP of index i than to other IOPs. For instance, in Fig. 62, the closest nodes to the IOP with coordinates (2,4) are inside the greyed isosceles triangle. Just like in the case of the previous metrics, this value depends on the random distribution of defective nodes in the network. We characterize each chip by the reachability number R_2 defined by the following formula:

Eq. 26 $$R_2 = Min\{N_C(i) | 1 \leq i \leq 4\} / N_N$$

R_2 is in fact the smallest fraction of nodes contacted by one of the IOPs which are closer to it than to any other IOP. Therefore, each IOP is able to contact a fraction of close cores in the array equal or larger than R_2. This reachability metrics (denoted RM2) is particularly adapted to the case of heavy workload when the IOPs compete for locking the processing resources following the mechanism described in section 5.1. The simulations to calculate $P(R_2 > R)$ are similar to those we just described in the previous section: the simulator generates a fraction of defective nodes in the network, then it calculates the number of valid nodes $N_C(i)$ which are the closest to each IOP, and finally the R_2 using Eq. 26. Simulation results are displayed in Fig. 68. Again, this figure shows how the increase of the fraction of faulty nodes in the grid reduces the probability to fabricate chips having a reachability higher that the sorting value R.

First, we may observe a small but clearly visible improvement when the IOP connectivity is increased from 4 to 8. However, it is fairly apparent that this improvement is not worth changing the grid topology around IOP as shown in Fig. 58c. Therefore, it turns out that increasing IOP connectivity is not an efficient solution to increase the accessibility of nodes in large networks, although it surely contributes to preserving high communication bandwidth. Second, Fig. 68 also shows that it is not so simple to guarantee a balanced workload among the IOPs in case of heavy processing. For instance, point A means that when 20% of cores are defective and isolated in the grid, the probability that R_2 exceeds 0.16 is about 0.8, i.e., $p(R_2>0.16)=0.8$. In other words, 80% of fabricated chips will be such that each IOP allocates more than 16% of grid cores. Remember that in the 4-IOP grid with 20% of defective cores, the maximum value of R_2 is $(1-p_{f,N})/N_{IO}$, i.e., 0.2. Therefore, we may conclude that the workload is reasonably balanced between the IOPs. However, the workload becomes quickly unbalanced when the node failure probability reaches 0.3. For instance, the simulation results prove that only 30-40% of manufactured chips (depending on IOP connectivity) with 30% of defective cores would be such that 12.5% of close cores are accessible by each IOP.

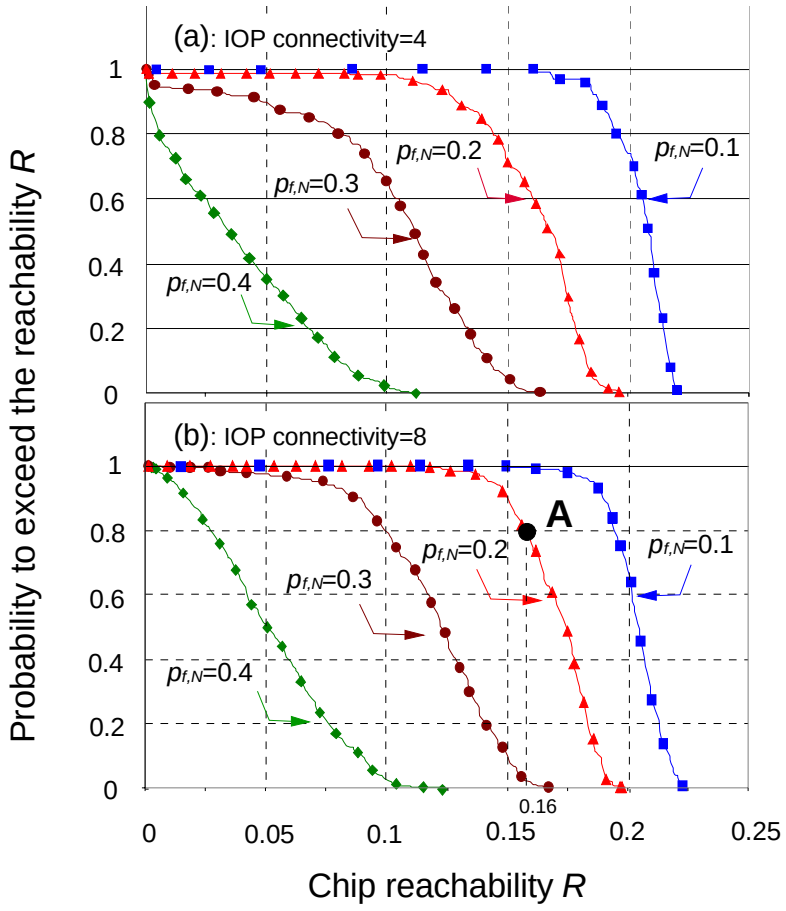


Fig. 68: Analysis of node accessibility using RM2 in the case of heavy workload. (a): IOP connectivity $n_{C,IOP}$ = 4; (b) : IOP connectivity $n_{C,IOP}$ = 8. The grid topology is a 2D-mesh.

The same metrics may be used to sort chips with faulty interconnects. In what follows, we consider a grid with no defective cores and various fractions of defective links. Consequently, we can compare the impact of defective nodes and defective links on chip reachability. As we showed earlier that the IOP connectivity has little impact on the number of contacted cores (see Fig. 67 and Fig. 68), we only studied the IOP connectivity $n_{C,IOP}$=4.

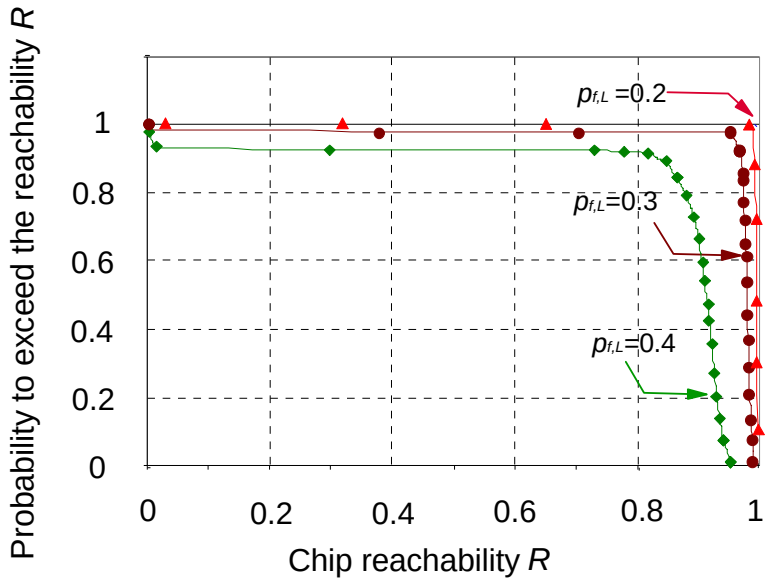


Fig. 69: Analysis of node accessibility using RM1 in the case of single IOP operation for various fractions of defective interconnects. The grid topology is a 2D-mesh.

Fig. 69 shows the results for single IOP operation. Even in the extreme case of 40% of defective links, the accessibility of the cores is satisfactory. For lower values of $p_{f,L}$, defective links have very little impact on the probability to exceed the predefined RM1.

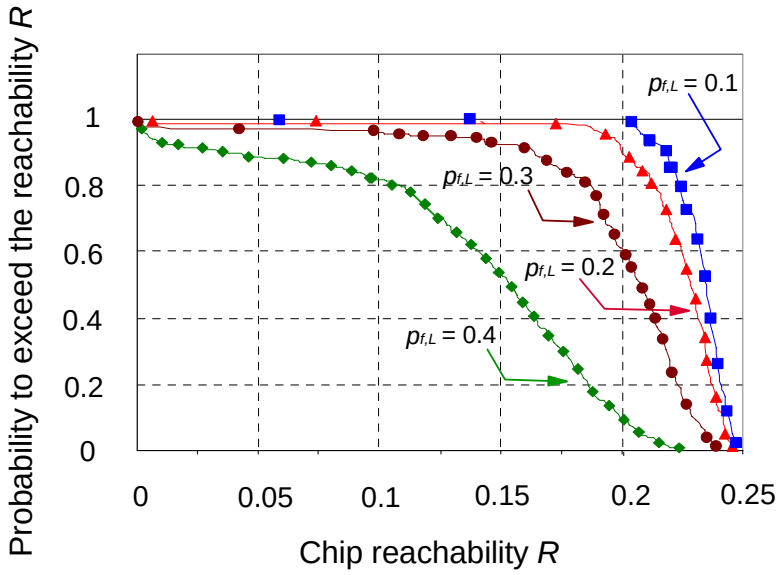


Fig. 70: Analysis of node accessibility using RM2 in the case of heavy workload for various fractions of defective interconnects. The grid topology is a 2D-mesh.

Contrarily, the impact of defective links is clearly visible in Fig. 70 which presents the results when all IOPs allocate tasks to their closest cores, as all IOPs compete for processing resources. Already for $p_{f,L}$=0.3 the probability that each IOP is able to contact at least 20% of cores is only 0.6 and for $p_{f,L}$ =0.4 it drops to 0.1. In other words, only 10% of manufactured chips with 40% of defective links guarantee that each IOP will be able to contact 20% of its closest cores in case of heavy workload in the grid. Let us now compare Fig. 69 and Fig. 70 with the figures obtained for various fractions of defective nodes (Fig. 67 and Fig. 68). In a mesh grid a defective node corresponds to 4 defective links and, additionally, it cannot participate in the processing. Clearly, a defective core should have a greater impact on network performance than a defective link. Indeed, the simulation results show that 10% of defective nodes have almost the same impact as 30% of defective interconnects in case of heavy processing in the grid. Therefore, we may conclude that defective links are even easier to tolerate than defective nodes in multiport architectures.

4.5 Summary

We have studied in this chapter the efficiency of route discovery mechanisms in defective grids. We considered both uniport and multiport architectures, as in large grids the presence of multiple ports is inevitable to ensure the scalability of the network. In most simulations we calculated the route discovery efficiency considering the reachability, i.e. the number of cores which may be contacted and used for executing tasks. This metrics may also constitute an approximation of chip performance degradation due to the presence of defective cores and links. We compared various network topologies and showed that the 2D mesh architecture should be considered as a reasonable solution when the core failure probability does not exceed 20%. Above this threshold, the node connectivity must be increased to maintain the desired chip performance.

We analyzed the effect of defective links and discovered that the presence of the fraction of defective links in the chip has a smaller impact on route discovery efficiency than the same percentage of defective cores in all topologies. Defective interconnects are then more tolerable in terms of maintaining chip performance. This conclusion may be ultimately used to balance the hardware overhead which should be implemented to protect the chip against core and link faults. Using the obtained simulation results we were able to estimate the production yield of future chips as a function of the core failure probability and network topology. The metrics that we used rests on a probabilistic guarantee to preserve a minimum reachability and to limit the number of defective neighbors adjacent to the IOP. The second constraint was introduced to guarantee high communication bandwidth between a chip and the external environment and it was shown that it dramatically reduces the production yield.

Since we discovered that the simple route discovery based on one broadcast may cause congestion in the network during chip operation, we have developed and simulated traffic balancing mechanisms. Our simple method proved to perform better that sophisticated method based on generating random numbers. Two analytical methods for chip sorting and validation have been also developed. The first method is based on estimating overall network connectivity and the second on takes into consideration the task allocation process and concentrates on IOP allocation capacity. Both methods give different results with respect to simple method based on calculating non-defective accessible nodes. It has been proved that chip validation based on fraction of accessible nodes may in some cases lead to a wrong decision.

Multiport architectures have also been studied. We calculated the chip area overhead necessary for protecting the ports which are critical elements of a chip. Moreover, simulations have been performed which allowed identifying the best positioning of the ports in a 2D mesh network. Similarly to uniport architectures, we simulated the route discovery efficiency and defined two sorting metrics based on the reachability of cores. It was shown that the modification of the port connectivity has only a small positive effect on the ability of the port to contact the cores. In the chips with multiple ports, various metrics for the ability of the ports to contact the cores may be considered, depending on the workload distribution among the ports. It has been shown that in case of heavy workload, it is very difficult to guarantee balancing the workload among the ports if the fraction of defective nodes in the chip exceeds 20%. The impact of defective links on node accessibility is even smaller (with regard to the impact of defective nodes) than in case of uniport architectures.

5 Dynamic allocation and redundant execution

In this chapter we analyze the impact of permanent and transient faults on the mechanism of task allocation and execution. We present a communication protocol which enables sharing the processing cores by multiple schedulers and we calculate the effect of defective cores on the communication latency. We also consider various protection techniques to ensure fault-free execution and we perform simulations which analyze the impact of both transient and permanent faults on chip performance.

5.1 Dynamic task allocation

Let us now introduce the common terms used in the task allocation theory. A task is simply a fragment of code which must be assigned to processing cores for execution. What is the size of the task? In fine-grained programming, tasks are quite small, comprising several hundred instructions. Contrarily, in coarse-grained programming, tasks are large, usually comprising over several thousands instructions. The choice of the task size depends heavily on the system architecture. In massively parallel systems with many cores fine-grained tasks should be preferred to take advantage of having many cores but on the other hand the task duration must be significantly longer than the time needed to allocate a task. Considering this latter constraint, we believe that it would be misleading to measure the task duration in instructions or processor cycles. Instead, we will measure it in network cycles, i.e., use as units the time needed to transfer a message across one link in the network (assuming synchronous transfer and including the processing time in the emitter and the receiver). An asynchronous transfer should be about 1.5 times slower (see section 2.5, page 16). In the 11x11 network that we consider in Fig. 73 page 97, the round-trip time between the IOP and the farthest node is below 20 network cycles. Consequently, in our analysis we shall assume that the average task duration is equal to about 100 network cycles.

The time ordering constraints occurring in the execution of a task group is often described by a precedence graph [77]. An example of such a structure is depicted in the figure below.

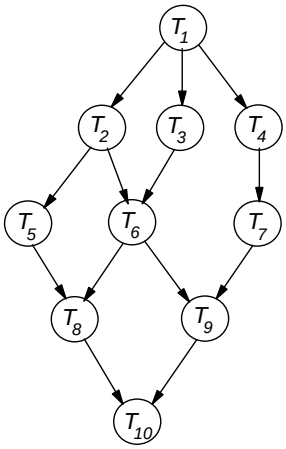


Fig. 71: Task precedence graph

T_i is the task of index *i*. The arrows connecting the tasks represent the precedence constraints. For instance, task T_6 cannot be started until both task T_2 and task T_3 are completed. We consider the graph to be acyclic, i.e., that there are no loops between the tasks (the program loops may of course be present inside a task). The task precedence graph is created at compile-time. The compiler parallelizes the application using for example Speculative Multithreading technique (see section 3.1, page 27), estimates task durations and deduces from the code the precedence constraints between tasks. The compiler is also supposed to ensure the consistency of execution. In the sequential consistency, it preserves the order of the READ-WRITE operations in the memory defined in a sequential uniprocessor execution. Next, the task graph must be mapped onto the processing resources: a schedule must be created. Schedules are usually presented as Gantt charts (see example below).

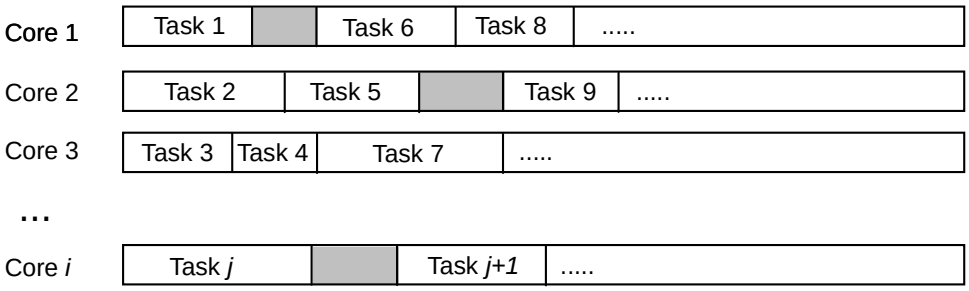


Fig. 72: Execution of tasks according to schedule represented as Gantt chart (greyed areas represent idle states)

There are two main types of scheduling:

- **Static scheduling**, performed at compile-time [78][79]
  The main advantage of the static approach is the ability to define the schedule using complex algorithms, as there is no time restriction regarding the duration of the algorithm (because it is not executed at runtime). Hence, some "quasi-optimal" schedule can be defined. The disadvantage is that the optimal schedule may be generated only if the compiler has full information about the system, for example about the number of available processing cores or on the inter-core transfer latency which may greatly change at runtime depending on possible traffic congestions. Thus,

static scheduling is not suitable in our approach because the compiler cannot know the number of accessible cores is the network. Perhaps, a more serious problem is the fact that in massively parallel systems, the number of processing cores may be greater than the parallelization level of an application which implies that the compiler cannot know which cores should be chosen to execute the application.

- **Dynamic scheduling**, executed at runtime [80][81]
  In this approach, the schedule is created at runtime which allows taking into account more system parameters. Its main drawback is that the scheduling algorithm must be fairly simple because too complex procedure may slow down the program execution.

Consider now the execution of an application with a multicore chip, as for example the 120 core array shown in Fig. 73 and let us assume that the execution may be parallelized on 20 cores. The compiler cannot know which 20 cores (out of total 120) should be chosen for executing the application, because it cannot know which cores are defective and which one will be free for processing at runtime! Consequently, the decision of "*which core should executed which task*" must be performed at runtime and take into account the current status of the processing in the network. Therefore, we naturally consider dynamic scheduling in this thesis. The task precedence graph is created statically at compile-time but the compiler neither assigns the tasks to cores (no schedule is created) nor defines the real degree of parallelization. When the program is executed, the scheduler must identify the pending cores and schedules the tasks according to the dynamic scheduling algorithm.

Let us analyze the on-chip multi-core network below. It is a 11x11 2D mesh with four IOPs in the positions marked by letters B. White and black squares represent good and faulty processing nodes, respectively. Remember that each node comprises a core and a router.

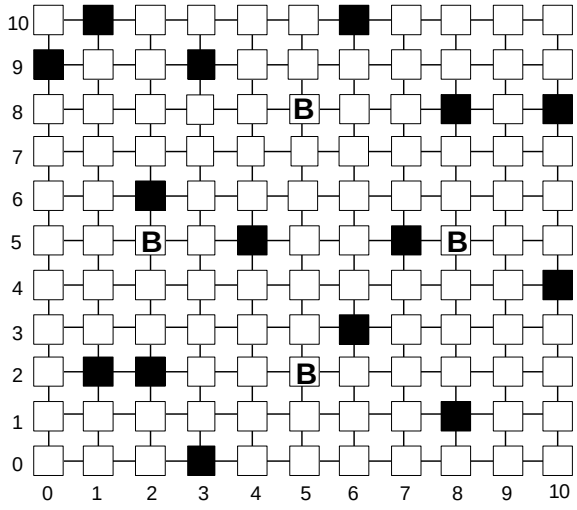


Fig. 73: Exemplary mesh network with 4 IOPs

In this thesis, we assume that each IOP also serves the role of scheduler; consequently, each one must compete with the other schedulers to allocate the incoming tasks to the nodes in the network. To allocate a task, each IOP should include in its local memory a special field which stores the last known state (*idle* or *locked*) of the accessible cores registered in its Valid Route Array (VRA, see

section 3.3.4, page 45). In our approach, an IOP should first scan its VRA to select a set of idle cores, and then, it should select an idle core in accordance with the scheduling algorithm. However, it must be stressed that even when assuming that each core informs the IOP when it switches from *idle* to *locked* (and conversely), there is no guarantee that the IOP knows the real state of the selected core! Actually, at time *t*, the IOP stores a "last known" state for each core, at best the state at time $t\text{-}T_L$, where T_L is the latency of communication with the core, which may be locked during that time by another IOP. Thus, the IOP must first check that the selected core(s) is (are) indeed *idle*. In the distributed approach, the following selection/locking mechanism (SLM) may be implemented based on the following 4-step protocol:

1. The IOP sends a *locking request message* (LRM) to one or more cores chosen for executing tasks. If the execution algorithm is parallel, the IOP should send several LRMs if it finds enough idle cores in its Valid Route Array.

2. Each contacted core responds:
   - If the core is *idle*, it accepts the LRM and: 1) It switches its state to *locked* and will not accept any further locking requests from other IOPs in the simple case when it executes only one task at a time; 2) It sends a *locking agreement message* (LAM) to the IOP which emitted the accepted LRM; 3) It sends a *locked state message* (LSM) to enable the other IOPs to switch the core state from *idle* to *locked* in their respective VRA.
   - If the core rejects the LRM, it sends a locked state message to the IOP which emitted the rejected LRM.

3. The IOP awaits the answer. If the answer is YES (because it received a LAM), it may allocate the task to the *locked* core. If the answer is NO (because it received a LRM), the IOP should update the node state to *locked* in its VRA and proceed with sending a new LRM to another *idle* node(s).

4. At the end of the task execution, each locked node sends an *idle state message* (ISM) to enable each IOP to update the core state to *idle* in its VRA for further task allocations in the network.

An interesting solution to minimize the latency of the core locking mechanism is the implementation of private cores. The principle consists in assigning some cores around each IOP for its exclusive use to eliminate the message round trip involved in the Selection/Locking Mechanism of shared cores. The private core will only respond to one IOP and, therefore, the state of the core in the Valid Route Array will be always correct. The number of cores which can be declared as private and assigned to each IOP should be seen as a user-defined property stored in the physical registers of the IOP, setup at start-up or even possibly updated at runtime. A simple mechanism to assign private cores in the network may be based on broadcasting a privacy request message (PRM) by an IOP. The message may have a small time-to-live (TTL) field so that it will be only received by a predefined number of cores around the IOP. Each core may have a "privacy bit" for each IOP. If a PRM is received, the privacy bit is set to "1" and the core becomes private for the IOP which emitted the privacy request. In further operation, the IOP will not need sending Locking Request Message to private cores. It can immediately send tasks to these cores.

For clarity, we grouped together in the table below the seven messages involved in the route discovery phase (see section 3.3.4, page 41), in the core locking mechanism and in the assignment of private cores.

Route Request (RRM)	Phase	Route Discovery (see Section 3.3)
	Emitter	IOP
	Destination	All Cores
	Description	First message broadcast to discover the routes (see Section 3.3)
Route Acknowledgment (RAM)	Phase	Route Discovery (see Section 3.3)
	Emitter	Core
	Destination	IOP
	Description	Sent to acknowledge a RRM
Locking Request (LRM)	Phase	Core Locking (see Section 5.1)
	Emitter	IOP
	Destination	Cores deemed as idle in the VRA
	Description	Emitted for selecting/locking an idle core
Locking Acknowledgment (LAM)	Phase	Core Locking (see Section 5.1)
	Emitter	Core
	Destination	IOP
	Description	Sent to the IOP which emitted the accepted LRM
Locked State (LSM)	Phase	Core Locking (see Section 5.1)
	Emitter	Core
	Destination	IOP
	Description	Sent in two cases: - To all IOPs when a core accepts a LRM (except the one which emitted the accepted LRM) to signal that the core state is now locked - To the IOP which emitted the LRM when it is rejected
Privacy Request (PRM)	Phase	Privacy Setup (see Section 5.1)
	Emitter	IOP
	Destination	Cores
	Description	Emitted to switch nodes to private. The message TTL is user-defined
Idle State (ISM)	Phase	End of processing (see section 5.1)
	Emitter	Core
	Destination	All IOPs
	Description	Emitted by a core to signal that its processing is complete and that its state is now idle

Table 4. Messages involved in the route discovery and core locking mechanisms

5.2 Calculation of communication latency

The communication latency is one of the crucial parameters to consider when estimating the efficiency of communications. Due to the presence of defects, the communication routes are longer because the messages have to move around the defective nodes and to avoid the defective links (see Fig. 54 page 75). The increase of the communication latency reduces the number of tasks n_T that can be allocated and executed in a given time, which of course has a direct impact on the chip performance. If we consider a limited number of processing nodes only, the decrease of n_T is of course proportional to the percentage of defective nodes (simply because there are fewer nodes that can accept incoming tasks). However, defective nodes also increase the network communication latency and, consequently, increase the task allocation time. As a result, in a given time interval, even fewer tasks may be allocated and executed.

We chose the number of hops (recall that a "hop" is a distance between two adjacent nodes in the network) as a way to measure the communication latency. In the simulations reported in Fig. 74 and Fig. 75 below, we measure the average communication penalty defined as the average communication latency minus the minimal communication latency at a given distance. For instance, if a node is located 10 hops away from an IOP in a perfect network and if the average communication latency in the simulations is estimated to 12.4, then the average communication penalty is 12.4-10=2.4 hops. The simulations were performed for various percentages of defective nodes, ranging from 10% to 40% and various network topologies (see Fig. 30 page 53). The simulation mechanism is similar to the one described in section 3.3.3, page 42.

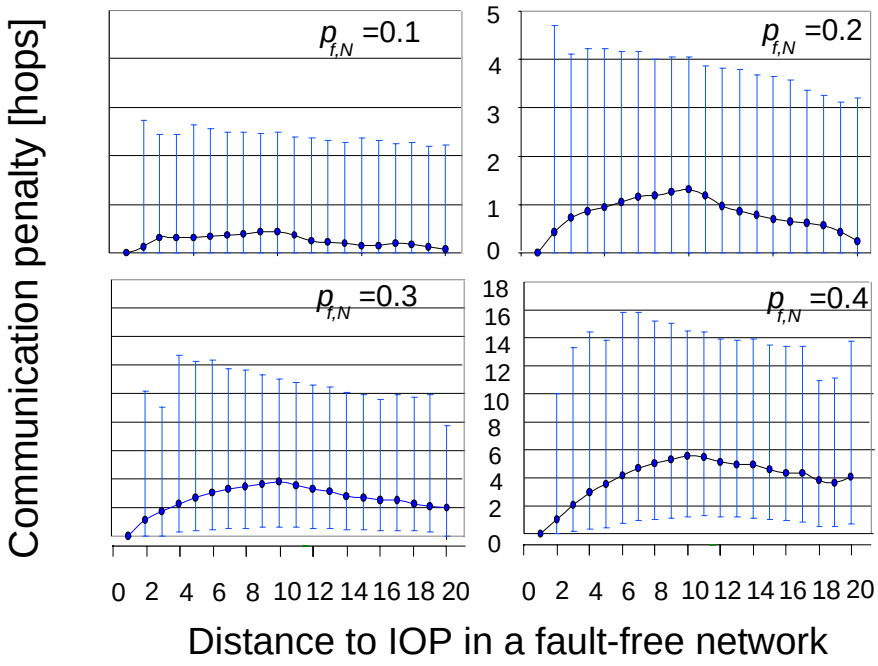


Fig. 74: Communication latency in 2D mesh network with 441 nodes for various fractions of defective cores

The figure above shows the results for a 2D mesh network. The X axis represents the node-IOP distance in hole-free network, which is *de facto* the shortest possible route between a

node and IOP. The Y axis displays the communication penalty. Vertical bars mark the upper and lower standard deviations from the average. Let us now analyze Fig. 74. Naturally, the communication penalty should gradually increase with the distance. In the figure, however, the penalty initially increases but then starts to decrease. This phenomenon is not at all surprising and can be easily explained as a consequence of the finite size of the network. If the distance between a node and the IOP is small and the shortest route is cut off by holes, there are still a lot of possibilities to move around the holes. Consequently, the IOP can contact the node by using a longer route. Contrarily, if the distance between a node and IOP is large and if the shortest route is unavailable, there could only be a few alternative routes since the network size is finite. Therefore, the IOP is not able to contact these nodes at all.

It is also noteworthy to point out that the standard deviation from the average could be very large. For example in Fig. 74, when $p_{f,N}$ =30% and when the distance is 10, the average communication penalty is 4 hops. However, the upper standard deviation is 8 hops! This indicates that some alternative routes are much longer than the nominal distance. For example, when the nominal distance is 5, the upper standard deviation is about 13, because in some (rare) cases, the IOP need to follow a route with 15-20 hops to reach the node.

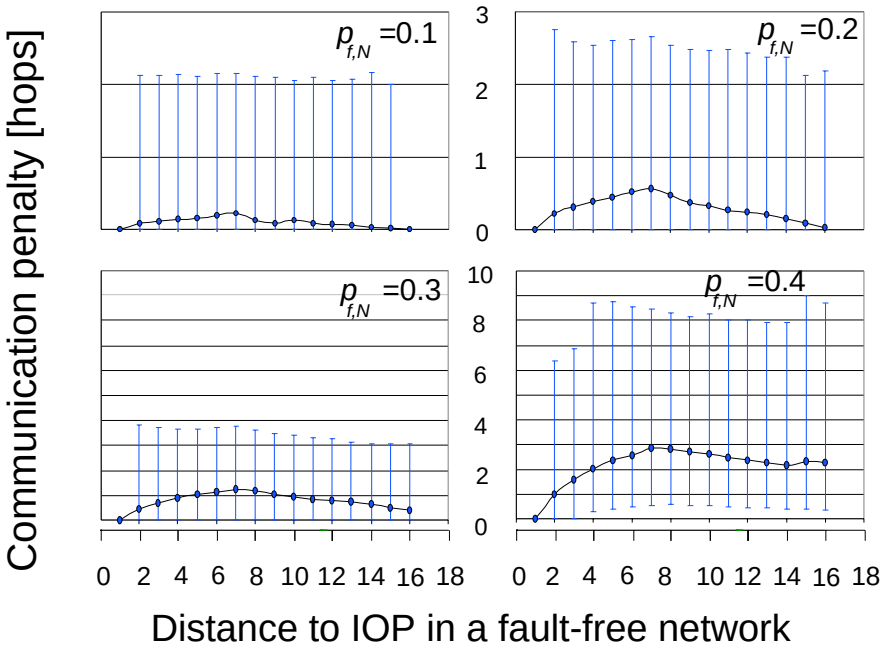


Fig. 75: Communication latency in the 2-layer mesh network with 450 nodes for various fractions of defective cores

The figure above represents the results for the 2-layer mesh network. We can see that the 2-layer mesh is much more resistant to defective nodes in terms of communication penalty. Up to $p_{f,N}$ = 0.3, the communication penalty stays below 2 hops, and even for 40% of defective nodes it is not very large, reaching 4 hops. The standard deviation increases rapidly if $p_{f,N}$ is above 0.3.

Again, we see that there exists a certain threshold of defective nodes above which the latency rapidly increases. The value of this threshold depends on the network topology and is about 0.25 for the mesh network and 0.35 for the 2-layer mesh network.

Let us now estimate the impact of the communication latency on the number of executed tasks. The analysis is similar to what has already been presented in section 4.3.2, page 75, where we calculated the top allocation capacity and therefore we neglected the execution time. Here we calculate the number of tasks executed in a given time, consequently both allocation and execution durations are taken into consideration.

During the time interval T_S, a node i can execute n_i tasks, where n_i can be approximated by the following formula:

Eq. 27
$$n_i = \frac{T_S}{T_{ai} + T_{ei}}$$

where T_{ai} and T_{ei} are the times of allocation and execution, respectively. Consequently, the average total number of tasks n_T that can be allocated and executed on N_N nodes is:

Eq. 28
$$n_T = N_N \cdot \frac{T_S}{T_a + T_e}$$

where T_a and T_e denote the average times of allocation and execution, respectively.

In the defective network, the same equation can be reformulated as follows:

Eq. 29
$$n_{Tf} = (1 - p_{f,N}) \cdot N_N \cdot \frac{T_S}{bT_a + T_e}$$

where $p_{f,N}$ is the node failure probability and b is a parameter which represents the increase in the allocation time due to increased latency needed for communication in the presence of defective nodes. Note that in the fault-free network b=1 and in a defective network b>1.

Hence, the network performance measured in executed tasks in a given time reduces to:

Eq. 30
$$\frac{n_{Tf}}{n_T} = \frac{(1 - p_{f,N}) \cdot N_N \cdot \frac{T_S}{bT_a + T_e}}{N_N \cdot \frac{T_S}{T_a + T_e}} = \frac{(1 - p_{f,N})(1 + h)}{1 + bh}, \qquad h = \frac{T_a}{T_e}$$

As we can see, the performance degradation is proportional to $p_{f,N}$, if we consider the reduced number of nodes only and discard the impact of defective nodes on latency (b=1). Of course, in a real system b>1.

Let us now evaluate the impact of the parameter h, i.e., the relation between T_e and T_a on network performance. Typically, the execution time should be much longer than the average allocation time, so we may assume that $T_e \cong 20T_a$. The above equation can be then reformulated to:

Eq. 31

$$\frac{n_{Tf}}{n_T} = \frac{(1-p_{f,N})\cdot(1+\frac{1}{20})}{1+\frac{b}{20}} = \frac{21\cdot(1-p_{f,N})}{20+b}$$

To evaluate the performance degradation, we still must calculate the parameter b as a function of $p_{f,N}$. The relation between $p_{f,N}$ and b can be empirically derived from the simulations displayed in Fig. 74 and Fig. 75. From these figures, we introduce the next phenomenological equation:

Eq. 32

$$b = 1 + c\cdot(p_{f,N})^{v}$$

where c is a constant, and v is a parameter depending on the network topology. For example, for the mesh network $c = 4$ and $v = 2$. Applying the above formula to Eq. 31, we obtain:

Eq. 33

$$\frac{n_{Tf}}{n_T} = \frac{21\cdot(1-p_{f,N})}{21+c\cdot(p_{f,N})^{v}} = \frac{21\cdot(1-p_{f,N})}{21+4\cdot(p_{f,N})^{2}} = \frac{(1-p_{f,N})}{1+0.19\cdot(p_{f,N})^{2}}$$

In the figure below we plotted the relation $\frac{n_{Tf}}{n_T}$ that we calculated above and, for the sake of comparison, we also plotted the same relation $\frac{n_{Tf}'}{n_T} = (1-p_{f,N})$ which holds only if we consider the reduction of the number of nodes and not the impact of increased communication latency. We may see that curves do not differ by much which proves that the increased latency due to defective nodes has small impact on network performance in comparison with the reduction of number of accessible nodes. For example, in a network with 30% of defective nodes, only 2% of the entire performance degradation is caused by increased communication latency. Why is the difference so large? Intuitively, one may see that the number of accessible cores in the network is critical for the performance, as it defines the number of tasks which can be executed simultaneously. Contrarily, the increase of the latency only slightly affects allocation phase which, additionally, has much shorter duration than execution time.

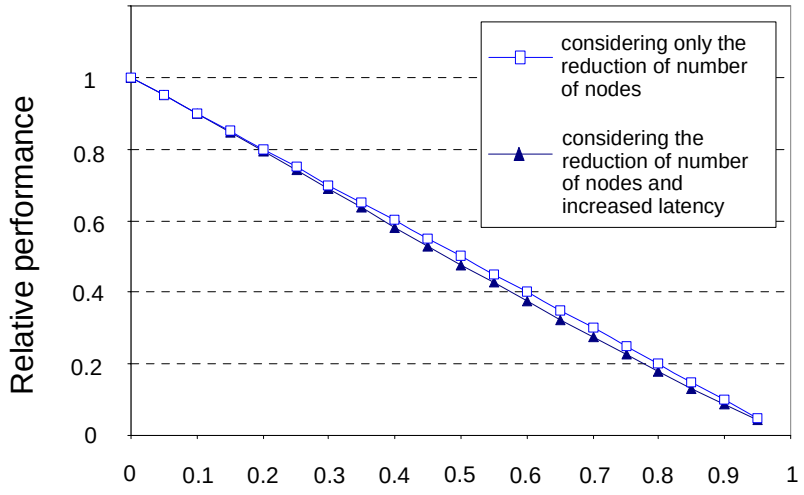


Fig. 76: Relative performance versus the fraction of defective cores when $T_e \approx 20T_a$

5.3 Redundant execution of tasks

5.3.1 Introduction

Task execution starts as soon as the scheduler sends the task information to a node. Recall that we assume that during execution all changes to memory variables are executed only in node's local memory, so the local memory should be large enough to contain all task data. We assume that only when the task is completed, a processing node is allowed to write the results to the main memory, so they can be visible to other nodes. The coherence of the main memory is ensured by the compiler by creating precedence constraints among the tasks (see Fig. 71). Each node may of course read the main memory and the shared L2 cache by sending memory read requests. Since there are many nodes which can perform reads and writes to each L2 cache, a coherence protocol must be implemented. As it has been already mentioned in section 3.1.1 page 30, the directory-based protocol is the most suitable method for maintaining cache coherency in the scalable multiprocessor chip. A variety of directory-based protocols have been proposed in the literature [82][83]. The detailed analysis of cache coherency problem is beyond the scope of this thesis.

In our approach, we suggest that the IOPs serve the role of the directories which store the knowledge about the copies of a memory block in the system. However, the important question which may be raised here is: if the data wanted by a processing node is found in the L2 cache of some other node, how is this data accessed? Recall that in the network there are defective nodes in unknown positions and a node does not know a priori the routes to other nodes. Let us now explain briefly the L2 cache access mechanism.

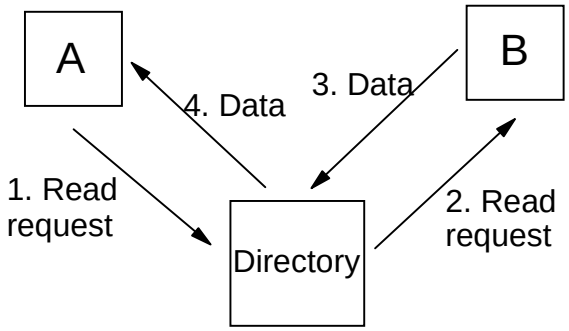


Fig. 77: L2 cache access mechanism

This method is called Intervention Forwarding. Consider that node A tries to access the data in the memory (Fig. 77). It first checks its private L1 cache and if the data is not present (L1 miss), sends a read request message to the IOP (directory). Note that all nodes know the route to the IOP (see section 3.3.4, page 45). Let us assume for the sake of analysis that the wanted data is present in the L2 cache. The directory checks the address in the request message and transfers the read/write request message to the node B which stores the data in its L2 cache. Similarly, after the data is read, node B sends the data to the directory which forwards it to the node A which originally sent the read request. The advantage of this solution is that no route

discovery between two nodes is needed. However, the drawback is that the traffic will concentrate around directories which may increase the memory access latency.

5.3.2 Transient faults in future nanoscale chips

Transient faults or soft errors in a logic circuit may be caused by many factors, as for instance:

- alpha particles,
- cosmic rays (neutrons),
- crosstalk.

Currently, the dominant factors are those based on radiation effects [84]. Let us describe for example alpha particle strike. If the particle hits the logic circuit, a charge is induced. If the energy of the particle is high enough, the induced charge may be higher than the critical charge Q_{crit} and the resulting voltage pulse may be large enough to change the logic state of a latch or memory cell. In the past, this problem was only experienced in large cluster systems. As the size of the transistors decreases, the critical charge Q_{crit} becomes smaller and the soft error rate inevitably increases. The recent studies have shown that although the number of errors due to alpha radiation stays at the same level, the error rate induced by neutron radiation significantly increases [85]. This problem will become more and more serious as the technology scales down.

Transient errors may produce different effects: corruption of messages in the network, modification of cache lines or registers or erroneous operation of combinatorial circuits. The messages can be protected by adding error detection and/or correction code to the message. Depending on the code used, one or more errors can be detected or even corrected. Furthermore, a reliable communication protocol can ensure that a corrupted message will be ignored by the system. Similarly, memories protected by Error Correction Codes (ECC) should be resistant to soft errors [60]. Therefore, in this thesis we only consider soft errors which result in bit flips in the processor’s logic.

5.3.3 Error detection methods

The most popular method for recovering from soft errors is backward error recovery (BER). The last known good processor architectural state is saved (a checkpoint is created) and when an error is detected, a rollback is performed (processor returns to the saved state) and all instructions since the checkpoint are executed again. Checkpoints are inserted by the compiler or the user. Probably the most interesting variation of this method is fingerprinting [86]. In this approach, instead of comparing the entire processor’s state, only the hash of changes to the processor’s architectural state (a fingerprint) is compared. In other words, a fingeprint is a “snapshot” of the changes to all processors registers and cache lines. Using fingerprints instead of the entire processor’s state makes the bandwidth and the time needed for checking the correctness of the execution several magnitudes lower.

Is is obvious that the mean time necessary to execute the code between two consecutive checkpoints should be much shorter than the mean time between two failures (MTBF). Otherwise, the probability would be high that the execution would need to rollback to the previous checkpoint. On the other hand, when error rate is negligibly low, frequent checkpointing causes unnecessary delay needed to compare the fingerprints. Therefore, the number of checkpoints in a program must be defined as a function of the error rate.

In what follows, we assume that when a task is allocated to a node (or a group of nodes) by an IOP, the node processor(s) create(s) immediately a checkpoint to re-execute the task if necessary. Error detection is performed at the end of the task by comparing fingerprints.

Most methods for transient error detection are based on temporal or spatial redundancy or on some combination of these two methods.

- Temporal redundancy is based on a simple idea: a portion of code is executed multiple times on a processor (core) to ensure that the obtained result is correct. Typically, a portion of a code is executed twice and possibly a three times when the results of the two previous executions do not match. Note that the result of the previous executions has to be stored for the comparison with the next execution.

- Spatial redundancy is based on another idea: the same portion of a code is executed on several processors (cores) quasi-simultaneously. Typically, three processors (cores) are used to enable comparison and voting.

Note that when spatial redundancy is applied, the error detection mechanism is capable of detecting the permanent faults which were not discovered during initial testing phase (see chapter 3.2, page 32) or which may have occurred at runtime, as for instance permanent aging faults. Simply, the IOP should keep track of the core failure statistics to remove from its Valid Route Array the processing cores which repeatedly signal execution failures. Consequently, they would no longer take part in the processing. Temporal redundancy cannot detect permanent failures, as the task is executed on only one processing core.

We must stress here that methods requiring that the same two or three processors always cooperate (processors are statically assigned to form groups of two or three processors depending on the execution method) are inflexible. Therefore, in our approach for each task different processors may be used for spatial redundant execution. Moreover, we consider a fingerprint to be some hash of task results rather that a hash of processor's architectural state. Note that whenever an error occurs causing a change in the processor's registers but not affecting the task results, there is no need to detect such an error.

Let us now describe more precisely the three execution modes that we considered in our calculations. Note that the mechanisms are simplified: for example we neglected the impact of irreversible operations (for example I/O accesses) on task execution.

Execution on 1 processor

The simplest method is based on temporal redundancy. At the beginning of a task, a checkpoint is created which enables future rollback to this point. The task is executed for the first time and the fingerprint is stored. Then the processor performs rollback, executes the task once more and compares the two fingerprints. If both fingerprints match, the execution is

considered to be completed successfully. If some mismatch is detected, it is not known *a priori* which execution failed and the task must be executed once more on the same processor. All fingerprints are compared following the third execution and if two out of three match, the result of the third execution is accepted. If all three fingerprints are different, at least two executions were corrupted and the task execution failed, because it is not possible to identify the right execution. The next actions depend on the implementation; in our approach the processing core informs the IOP that the execution was unsuccessful and, consequently, the IOP has to allocate the task again.

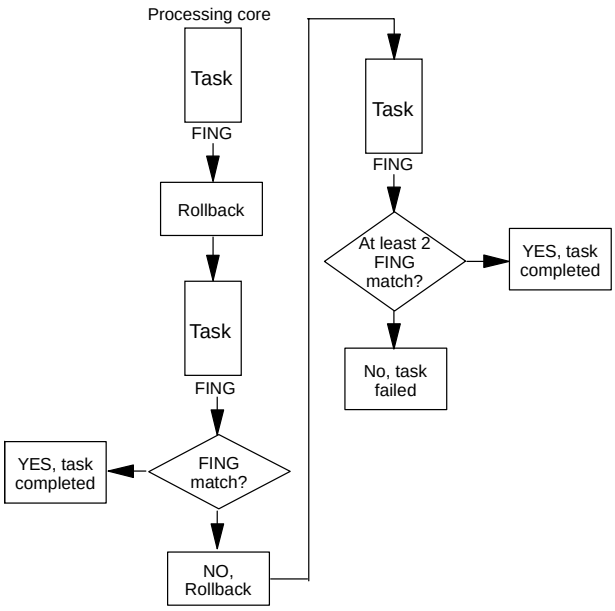


Fig. 78: Execution on one processor – decision tree

The advantages of this approach are: 1) only one processor is needed (it saves resources for other tasks) and 2) there is no communication between cooperating processors. The main disadvantage is that when there are no errors, every task is executed twice. Consequently, the average task execution time (assuming no errors) is roughly twice longer than the spatial-redundancy methods which use 2 or 3 processors.

Execution on 2 processors

This method is also called dual modular redundancy (DMR) [87]. The task is executed quasi-simultaneously on 2 processors (master and slave). Many types of mutual cooperation are possible; in our approach we consider the DMR execution to work as follows (see also Fig. 79):

1) Both processors execute the task.
2) When the slave finishes its task, it sends its fingerprint (FING) to the master, which compares it with its own fingerprint.
3) The master processor compares the fingerprints and if they match, it considers the task as completed successfully. Contrarily, if it detects a mismatch, the master performs a rollback and executes the task again.
4) The result of the third execution is accepted if it matches one of the fingeprints of two previous executions. If all three fingeprints are different, the master informs the IOP that execution failed and the tasks has to be allocated again.

Similarly, when the master finishes its task first without receiving the FING from the slave, it sends a request for fingerprint (REQ). A timeout may be implemented to ensure that if no FING is received from the slave in a given time period, then the master processor considers that the execution on the slave processor has failed. When the FING from the slave is received, the master follows steps 3 and 4 from the list above.

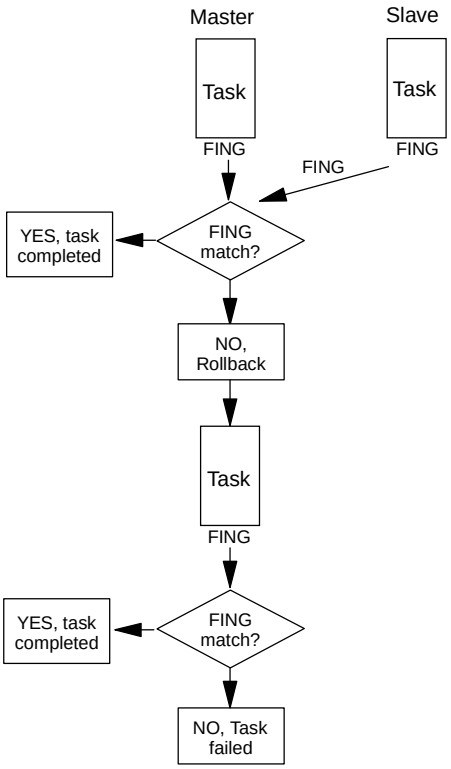


Fig. 79: Execution on two processors – decision tree

Execution on 3 processors

This method, in which the task is executed on three processors, is also called triple modular redundancy (TMR). One of the processor is assigned as master and two other ones as slaves. The procedure is similar to the one presented above for the DMR method: after the task execution, the two slaves send their fingerprints to the master who compares all three fingerprints. When at least two out of three fingerprints match, the task is considered as completed successfully by the master. If all three fingeprints are different, the master informs the IOP that the execution failed and the IOP must allocate the tasks again. Note that when the master has the only incorrect fingerprint, it must assign one of the slaves as new master who will complete the task, i.e., write the task results into the memory.

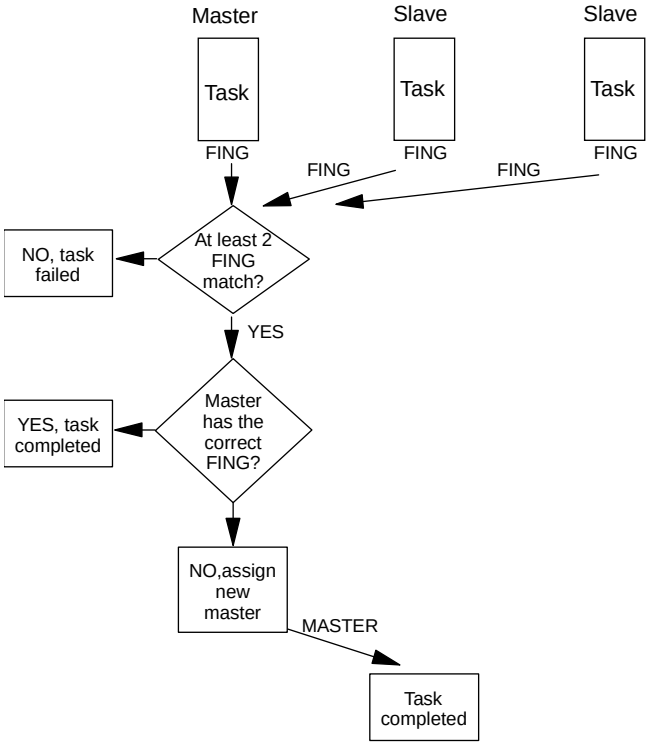


Fig. 80: Execution on three processors – decision tree

5.3.4 Comparison of redundant execution methods

To compare the efficiency of the methods described in section 5.3.3, we performed an analysis in which we estimate the number of completed tasks in a multiprocessor network in a given time. To verify the correctness of our calculations, we performed the simulations of the allocation and execution process using MASS simulator (see Appendix B). We simulate the execution of tasks on 2 processors.

Principle of simulations

We analyzed a 2D 10x10 mesh network with N_N=100 processing nodes. Time is measured in network cycles. A network cycle is in fact an inter-node latency (the time needed to transfer a message between two adjacent nodes). This approach makes our analysis independent from the inter-node latency. We also assumed that the network is saturated: the number of tasks in the queue is infinite and there is a task ready to be allocated at each network cycle. As the aim of this analysis is only to compare the allocation methods, we used simplified allocation mechanism.

The single input/output port (IOP) located in the centre of the network is in charge of the task allocation process. Let us recall that:

- Following the route discovery phase (see section 3.3.4 page 45), the IOP has in its valid route array (VRA) one route per reachable fault-free node.

- During the Locking-Allocation phase, the IOP and the nodes exchange the messages described in Table 4 page 99. The IOP sends LRM messages to the nodes (Locking Request Messages). An idle node which receives a LRM becomes busy and it is locked to start the execution of a task. Note that the node's state is saved by the IOP in its VRA on reception of a LAM, so that the busy nodes (or the nodes considered as busy by the IOP in a multiport chip) are not taken into consideration.

- If all nodes are busy, the IOP stops allocating tasks until it receives an ISM (Idle State Message) from an idle node in the network. The tasks are executed following the rules described in section 5.3.3, page 105.

In case of DMR, the IOP sends the same task to two processors. The communication between two processing cores is performed as the L2 cache access, described in section 5.3.1, page 104. The master compares the results and if no error is found (i.e., if both results match), it informs the IOP that the task is finished (i.e., it sends an ISM) and both processors become idle. If the master detects a mismatch in the result, it executes the task one more time and compares all three results. If the same result has been obtained twice, the task is completed successfully and the master sends an ISM to the IOP. If all three results are different, the master processor must execute a recovery decision. In our approach, it informs the IOP that the task execution failed and that the task must be allocated and executed again.

In our simulator, faults are injected randomly. The probability of fault occurrence per cycle is the inverse of the MTBF (Mean Time Between Failures). For example, if the MTBF=1000 cycles, the probability that there is a transient fault in a given network cycle equals 0.001. We assumed that the task duration is T_D=100 network cycles. We calculate the number of tasks completed successfully by all processors in T_S=1000 cycles. Let us now proceed to analytical calculations.

Analytical calculations (saturated processing regime)

First of all, we made an assumption that we can analyze separately every processing group (one processor, DMR pair or TMR group, depending on the execution method).

During a time interval T_S, the average number of tasks k that can be executed by one processing group of index i (PG) reads:

Eq. 34

$$k_i = \frac{T_S}{T_{ei} + T_{ai}}$$

We have to stress that we must count only the tasks which are completed during the time interval T_S and not those that are still in progress at time T_S. For example if at time T_S the processing group A executed k_A=8.6 tasks and the processing group B executed k_B=10.4 tasks, the average value of completed tasks is (10+8)/2=9 tasks. We have to count only the integer

value of k_A and k_B. Consequently, we have to subtract 0.5 from the formula above to get the correct result.

Eq. 35
$$k_i = \frac{T_S}{T_{ei} + T_{ai}} - 0.5$$

Thus, the average number of tasks completed by all processing groups is (assuming heavy processing when each processing group continuously executes tasks):

Eq. 36
$$k = N_{PG} \cdot \left(\frac{T_S}{T_e + T_a} - 0.5 \right)$$

where:

- Note that T_e is not equal to T_D. T_e is the average execution time considering the occurrence, detection and solution of errors. It depends on the structure of the processing group which possibly includes one, two or three processors and is also a function of p (the probability of error during execution). The average T_e is estimated in Appendix E.
- T_a is the average allocation time (the number of cycles needed for communication between IOP and PG and within PG itself).

However, we must remember that we must take into account the non-zero probability that the entire redundant task execution could fail. This will happen, if the results of three executions are different (there is an error in at least two of three executions). The probability $p_{3diff\ results}$ can be defined as:

Eq. 37
$$p_{3diffresults} = p^3 + 3 \cdot p^2 \cdot (1 - p)$$

where p is the probability of an error in one execution. Hence, the number of tasks completed successfully is:

Eq. 38
$$k_S = N_{PG} \cdot \left(\frac{T_S}{T_e + T_a} - 0.5 \right) \left(1 - p_{3diffresults}\right) = N_{PG} \cdot \left(\frac{T_S}{T_e + T_a} - 0.5 \right) (p-1)^2 (2p+1)$$

In what follows, we apply this formula to calculate the number of completed tasks for each of three execution methods described in section 5.3.3. The detailed calculations are presented in Appendix E.

Results

The chart below presents number of executed tasks for three different methods of execution for the following parameters:

N_N=100, T_D=100, T_S=1000

The X axis represents the relation between the Mean Time Between Failures (MTBF) and the task duration T_D. The Y axis represents the number of completed task during time T_S.

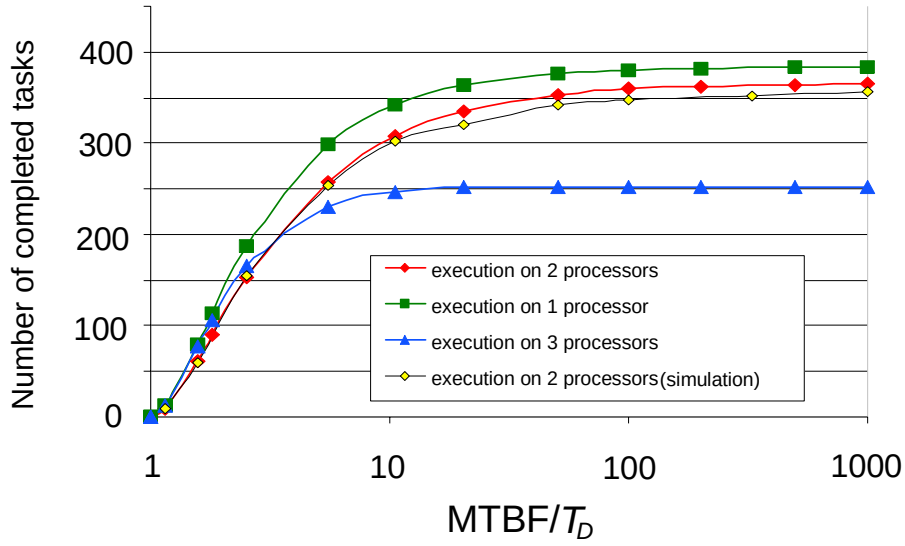


Fig. 81: Number of tasks executed in a given time interval as a function of the normalized MTBF

First, we can see that the results obtained by analytical calculation and numerical simulations are very similar. This proves that our analysis should be correct.

We see that the number of completed tasks increases with the parameter MTBF/T_D. Naturally, if MTBF≅T_D, the number of completed tasks is very low, as an error occurs during almost every execution. Contrarily, if MTBF>100T_D, then almost all tasks are completed successfully, because the probability of two erroneous executions is extremely low.

If we compare the three curves, we can see that, surprisingly, the execution on 1 processor has the best results, regardless of the failure rate. We have to remember though that the above analysis was performed in a completely saturated network. Fig. 81 suggests that the methods using spatial redundancy are less efficient in saturated network. The explanation of this phenomenon is fairly simple: in a saturated network, the number of tasks that can be executed simultaneously is much more important that the gain from spatial redundancy. In other words, we achieve maximum performance, when one processor performs one task.

However, the execution on 1 processor is not suitable for dealing with permanent faults which may occur during operation of the chip and above all, due to the incomplete fault coverage of the diagnostic phase described in section 3.2.3, page 37. Moreover, when a burst of errors occurs, this method is likely to fail because the tasks are executed on a single processor. Two errors in two consecutive executions cause a critical error, as we will not be able to

distinguish which result is correct even after the third execution. The entire task will have to be restarted and the time spent on three executions is wasted.

We must also emphasize that the efficiency of the three methods depends heavily on the network saturation. More precisely, in the above simulations we assumed that, at any time, a new task is pending for allocation and execution. Naturally, this assumption is favorable to the temporal redundancy and therefore, the method based on the execution on 1 processor obtained the best results. Indeed, if a network is not saturated and there is little workload, spatial redundancy is clearly better. For example, if there are three idle processors in the network and we have only 1 task waiting to be executed, why not use all three processors to execute the task as fast as possible (using the method based on execution on 3 processors)? **Therefore, the best method would likely be to dynamically adapt the redundancy for each task. If a network is saturated, less critical tasks may be executed on 1 processor. Contrarily, if most of the nodes are idle, all tasks may be executed on three separate processing cores.**

5.3.5 Impact of permanent and transient faults on task execution

In the previous section, we concluded from analytical calculations that time redundancy enables the largest number of executed tasks per time unit when the processing is saturated, i.e., when tasks are permanently pending for execution in the IOP. Therefore, we implemented this method in the simulator MASS.

Moreover, we considered multiport architectures: the multicore network that we simulated is presented in the Fig. 62 page 84 with 4 IOPs positioned according to letters B. Obviously, the presence of multiple ports requires the implementation of the core sharing mechanism described in section 5.1. The entire core sharing protocol (see Table 4, page 99) has been simulated, including the locking requests. The task duration was modeled by a Gaussian distribution around the average value T_{Dav}=100 network cycles. Recall that the network cycle is the time needed to transfer a message between two cores in the network. We measured the tasks completed by all processing nodes in the simulation time (10000 cycles). The node failure probability $p_{f,N}$ was a parameter in our study, we performed simulations for four values of $p_{f,N}$ (see Fig. 82). We also included transient errors in our study: the parameter we used was MTBF which represents in fact the average time between two transient errors which cause execution failure. Please note that the simulations were cycle accurate, i.e., every transfer of message across every link in the network was simulated. We did not consider communication errors. Let us now briefly describe how the simulations were performed.

- The simulator randomly generates a fraction of defective nodes in the network, corresponding to parameter $p_{f,N}$.
- All four IOPs perform route discovery (see section 3.3.4).
- Following the route discovery phase, all IOPs start allocating tasks to the nodes which are present in their Valid Route Arrays.
- The allocation phase is performed using the distributed allocation mechanism based on core locking and described in section 5.1; consequently, the propagation of all messages in Table 4, page 99 is simulated.
- The task duration is between 75-125 cycles, modeled by a Gaussian distribution.

- During the execution of a task, the program may randomly generate an execution failure according to MTBF.
- Each task is executed twice by the same processing node. At the end of a task the processing node stores the "fingerprint" (see section 5.3.3) of its architectural state.
- At the end of the second task, the processing node compares the two fingerprints of two executions. If the signatures match, the task is believed to be completed successfully. Contrarily, if the signatures are different, one or several execution errors occurred during the execution and the node must re-execute the task for the third time**.** If no error is generated during the third execution, the task is assumed to be completed successfully.
- If one more error occurs during the third execution, the execution of task is considered to be unsuccessful. The node sends an error message to the IOP and the same task must be allocated and executed again.
- After 10000 cycles, the program counts the number of completed tasks in the multicore network.

The simulation results are presented below in Fig. 82 in case of heavy workload in the network. We assume that in each IOP a new task is pending in the task queue on every network cycle. Moreover, we considered completely independent tasks. More precisely, the tasks are not associated with any precedence constraints.

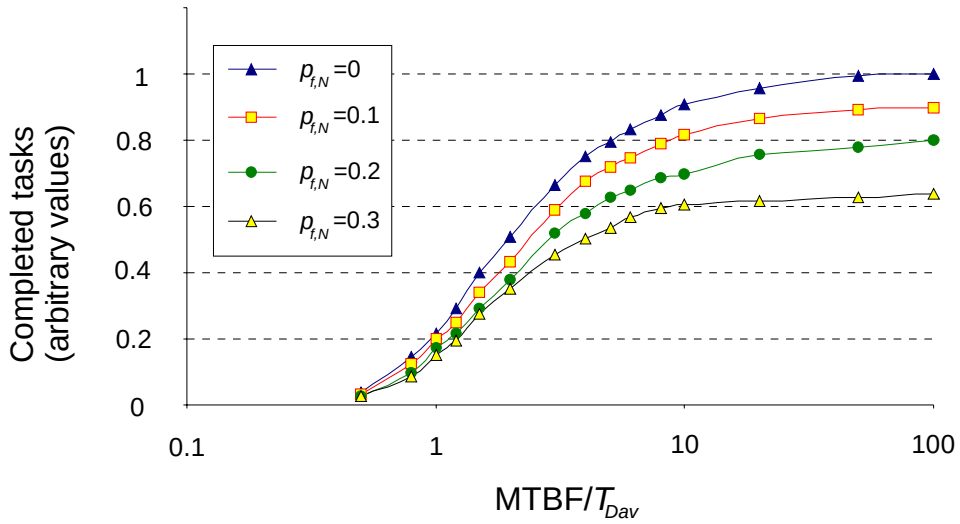


Fig. 82: Number of completed tasks during the simulation time versus the normalized MTBF for various node defect probabilities and heavily saturated network

Let us now analyze the results in detail. On the X axis we see the parameter MTBF, normalized versus average task duration T_{Dav}. The Y axis represents the number of completed tasks presented as arbitrary values, where 1 represents the number of tasks completed in the chip with no permanent and no transient errors. For clarity, we will explain the figure starting from high values of MTBF (low probability of transient errors). For instance, when MTBF/T_{Dav}=100, the number of completed tasks depends heavily on $p_{f,N}$. In fact, we may see that the number of completed tasks is proportional to the number of non-defective nodes. Interestingly, when $p_{f,N}$ reaches 0.3, the performance drops by almost 40% (instead of expected 30%). This shows that if the fraction of defective nodes exceeds 20%, the

performance is affected also by other factors like, for example, good nodes blocked by defective ones or increased communication latency needed for moving around the defective nodes. As the rate of transient errors increases, we see that the absolute differences between the curves decrease: in fact, they become negligible when MTBF/T_{Dav}<1.

Thus, when the rate of transient errors is very high, the performance is primarily degraded by transient errors, regardless of the fraction of permanently defective nodes. In other words, if the transient error rate is high enough, it does not matter whether the chip is fault-free or comprises 30% of defective nodes! Of course, in real systems such high error rate is extremely unlikely, but it shows very well how the performance degradation due to transient errors may become dominant over performance degradation due the reduction of valid cores not affected by permanent faults.

It is important to point out that when MTBF/T_{Dav}=10, the number of completed tasks drops by 10% which may be reasonably considered as an acceptable degradation of the performance level. However, for MTBF/T_{Dav}<10, we observe quick degradation of performance. Already for MTBF/T_{Dav}=2 the multi-core chip can only provide half of the maximum performance.

Fig. 83 presents the results for moderate workload in the network. In this case we considered tasks grouped in applications according to the task precedence graph shown in the figure below. Again, on the Y axis the value 1 represents the number of tasks completed in the chip with no permanent and no transient errors.

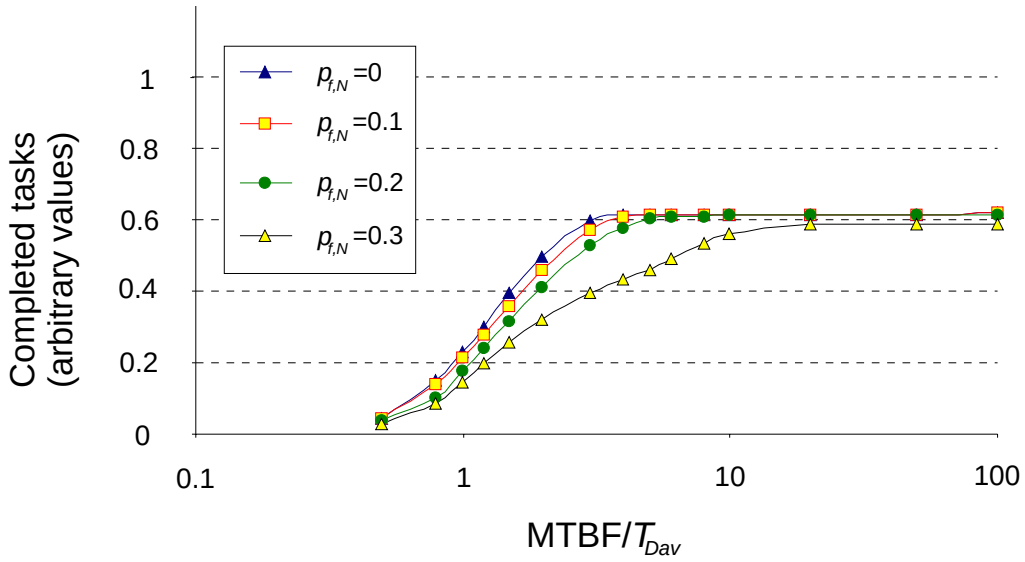


Fig. 83: Number of completed tasks during simulation time versus normalized Mean Time Between Failures for various node defect probabilities and moderately saturated network

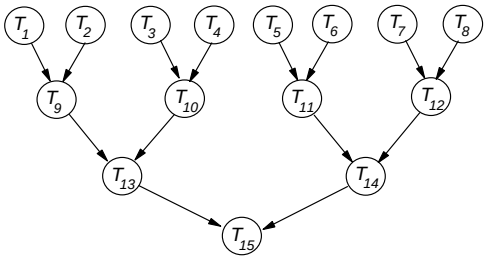


Fig. 84: Model of task precedence graph used in simulations

Each application consist of 15 tasks, mutually linked with precedence constraints, as shown in Fig. 84. In our simulations, we assumed that, in each IOP, a new application is ready to be allocated and executed every 100 cycles. How does this assumption influence the task execution in the network? First, there are periods of time, when there are no tasks in the task queue. Second, if there are more idle processing nodes than pending tasks, some of the nodes will be idle for a certain period of time. Let us now analyze the results in details. Again, we plotted on the X axis the parameter MTBF, calculated in average task duration T_{Dav} units. The Y axis represents the number of completed tasks presented as arbitrary values.

The fact that the workload in the network is quite low has an important impact on the results (see Fig. 83). For instance, for $10<\text{MTBF}/T_{Dav}<100$, the fraction of defective nodes does not change the chip performance. This effect is reasonable: if 70% of the cores can handle the entire workload, it does not matter which fraction of the remaining 30% is defective. Similarly, the transient error rate also does not affect the performance: it does not matter if the task is executed three times or two times because the gained time cannot be used to execute another task. Just like in Fig. 82, we may observe considerable performance degradation due to transient faults when MTBF/T_{Dav} approaches 1.

5.4 Summary

In this chapter we have analyzed the task allocation and redundant execution mechanism. Dynamic task allocation must be implemented in massively defective grids because the compiler cannot know which cores are defective and because idle cores can only be identified at runtime, especially when several IOPs compete to lock resources. Thus, we have presented no scheduling algorithm. However, we have developed a task allocation protocol which allows sharing the processing nodes by multiple IOPs which work as schedulers. We have performed simulations which calculate the communication latency due to the presence of defective nodes. However, it has been discovered that the performance degradation caused by increased latency is negligibly small in comparison with the degradation caused by the reduction of number of non-defective nodes.

We have also presented and compared three methods of redundant execution based on temporal and spatial redundancy. The best simulation results (within the accuracy of our models) are provided by the method based on executing the task twice on the same processor. However, it must be emphasized that this method is not capable of discovering permanent faults which may occur during chip operation or due to the incomplete fault coverage of the diagnostic phase described in section 3.2.3, page 37. Therefore, the method based on Dual Modular Redundancy (DMR), which gives slightly worse results but detects permanent faults, should also be envisaged.

The impact of transient and permanent faults on chip performance has been analyzed. It was shown that the fraction of defective nodes is the most important factor, but performance degradation due to transient errors is dominant when the MTBF becomes comparable with the task duration. Moreover, we have shown that it is very difficult to estimate the impact of faults on the functioning of a chip, because the performance degradation depends largely on chip operation parameters and particularly on the workload.

6 Conclusion

In the history of microprocessor design, there were already several moments when it seemed that the designers faced some barrier which could not be overcome. However, solutions have always been found and processors have continued to evolve, reaching extremely high levels of processing power. The range of problems that chip designers had to cope with today is very wide. However, it seems that a completely new kind of threat has emerged. Rapidly progressing miniaturization has enabled chip manufacturers to produce transistors so small that they can be measured in atomic layers. Consequently, the device parameters (as for example the doping density) can no longer be viewed as continuous since they have become discrete quantities because of the granularity of matter. These granularity effects are still unexplored, although they become more and more significant with the downsizing of transistor dimensions. As a result, in the future nanoscale technologies the designer will probably face a dramatic rise of the number of on-chip defects. As the reduction of the dimensions seems irreversible (the demand for increasing processing power will surely force the reduction of dimensions), designers will have to invent and implement protection and mitigation paradigms which will ensure the correct operation of VLSI chips even in the massive presence of faults. Note that exactly the same problems of fault tolerance will emerge in the embryonic technologies derived from the molecular electronics.

How to protect the extremely complex nanoscale chips? Future nanoscale technologies might be so defective that the protection on only one level would not be sufficient. Therefore, the cooperation among designers working on all system levels, from the transistor-level to the application-level will become necessary to win the dependability battle in fault-tolerant nanochips. We believe that each layer should contribute to fault tolerance. More precisely, the role of the protection implemented in each layer is to only reduce the impact of faults. Ultimately, the combined fault tolerance techniques at all levels will allow true fault-tolerant chip operation.

This thesis is a contribution to how to protect the future nanoscale chips at the architectural level. In this work, we analyze general-purpose processing with multiple cores integrated on a single chip. Using replicative architectures, i.e., cores connected to build a regular scalable array, solves the complexity issue of future chips and ensures the constant increase of the chip processing power. It also constitutes a natural solution to tolerating permanent faults in a chip, as the failure of a fraction of cores does not generally implicate the failure of the entire multi-core chip. We consider both manufacturing (permanent) and transient faults in future chips. Therefore, in this thesis a complete self-configuration methodology is presented which contributes to increasing the dependability of massively parallel multicore chips. The idea of self-configuration is not new, as it is for example employed in neural networks, and follows the natural trend of increasing the chip autonomy as its complexity grows.

It must be emphasized that the self-configuration is not physical as no spare elements are used and there is no physical reconfiguration of the chip, but logical, i.e., the physical chip architecture do not change. Self-configuration is based on the following mechanisms.

First, the chip should be able to detect any permanent faults that occurred during the manufacturing process and to isolate the defective cores. In our approach, we suggest using mutual testing of on-chip processing cores, based on Software-Based Self-Testing (SBST). This method requires the external test of the input/output ports only, and its crucial advantage is that the isolation of defective cores is independent on the actions or decisions of defective cores. As a result, all cores comprising detected faults are logically disconnected from the fault-free part of the chip and do not interfere with the subsequent chip operation.

Second, as the chip comprises a fraction of isolated defective cores, a communication mechanism must be developed which enables routing messages around defective cores in the chip. In this thesis, we suggest that the routes between cores in the multicore array are discovered during the start-up phase and then stored in some internal non-volatile memory so that they can be used thereafter during regular chip operation. The route discovery mechanism is based on broadcasting messages which incorporate the route that they follow into their header. To study the efficiency of the route discovery, we have conducted numerous simulations using dedicated software and we have shown that the popular 2D mesh topology may be envisaged as long as the fraction of defective cores is below 20%. Above this threshold, topologies with higher connectivity, like the 2-layer mesh, should be considered. Defective interconnects also cause higher latency and lower the production yield, but the impact of a fraction of defective links is approximately 2 times smaller than the impact of the same fraction of defective cores in a mesh network. The higher the network connectivity, the less important the defective links in comparison with defective cores. The route discovery efficiency, the estimated production yield and the communication latency were also investigated for various network topologies.

Third, to cope with transient faults which may corrupt chip operation, fault tolerance mechanisms must be implemented to protect the chip at runtime. Therefore, several solutions based on the redundant execution have been analyzed and compared. We have shown that temporal redundancy provides the best results in the conditions of heavy processing. However, dual modular redundancy should be privileged (even if its results are slightly worse than for temporal redundancy), because it additionally enables detecting the runtime errors due to the presence of permanent faults undetected during the initial testing or which may occur during operation. Moreover, we have also presented in this thesis some simulation results of a multi-core network for a wide range of operation parameters, which estimate the impact of both permanent and transient faults on chip performance. At low rate of transient errors, permanent faults constitute a dominant factor in performance degradation, but when the MTBF becomes typically shorter than 10 task durations, transient faults become dominant. When the MTBF becomes approximately equal to the task duration, the chip performance suffers from a considerable degradation, practically making the chip inoperable. It must be also emphasized that the simulations showed that the estimation of the performance degradation due to faults heavily depends on the workload.

Appendix A: A brief reminder on communication

Multicast protocols

Multicasting messages is crucial to the route discovery in the massively defective core arrays. We have studied two multicast protocols: flooding and gossiping as well as various modifications of these two protocols. We have evaluated them with respect to the efficiency, simplicity and on-chip implementability.

Let us now describe what we mean by multicast efficiency. The goal of every multicast is to deliver the message to several destination nodes. The efficiency of the multicast may be evaluated by comparing the number of nodes which receive the message with the amount of used network resources. More precisely, the goal of multicast is to deliver the message to multiple nodes using as few transmissions as possible. By simplicity and implementability we mean that the protocol must be simple enough to minimize the chip surface needed for router implementation. Moreover, the simpler the multicast the faster the message can be processed by a router which decreases communication latency.

The first multicast that we have analyzed is flooding, which is, in fact, a broadcast because the message is delivered to all accessible nodes. The rule of flooding is very simple: the node which receives a message sends it to all of its neighbors, except the neighbor which was the source of the received message. Note that to avoid back propagation, only the first copy of the message is transmitted and other arriving copies are ignored. Flooding succeeds in delivering the message to all nodes in the network. However, flooding is considered to be the most redundant and, therefore, the less efficient method. Indeed, the rule mentioned above induces considerable redundancy in terms of the number of message copies in the network. Consider simple example presented in Fig. 85.

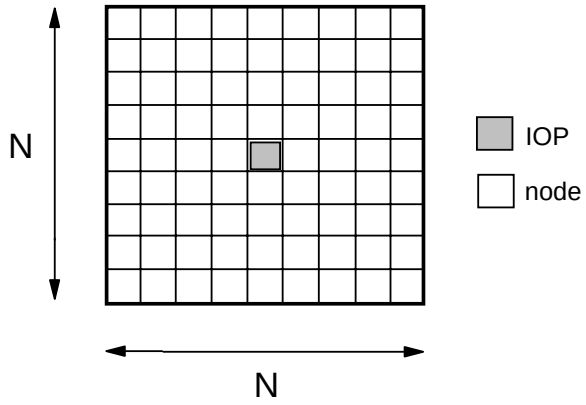


Fig. 85: Sample NxN mesh network with IOP located in the center

In the *N*x*N* mesh network, an IOP is surrounded by N^2-1 nodes. Suppose that the goal of the IOP is to send the message to all nodes. How many transmissions are needed? Ideally, we could accomplish this goal using only N^2-1 transmissions. According to the rules of flooding, every node retransmits the message to 3 neighbors. The number of transmissions in case of flooding will be then about three times greater than the number of nodes, which is $3(N^2-1)$ transmissions. It can be seen then that flooding uses a lot of redundant message copies.

Nevertheless, it is worth considering because it is also the most robust multicast technique. More precisely, flooding guarantees that the message will be delivered to all contactable nodes, even if there is a very large fraction of defective nodes in the network.

Limited flooding is a variation of this method, which simply includes additional TTL (Time-To-Live) field in the message. This field comprises the number of maximum message retransmissions. As the message propagates across the network, the TTL is decremented at each node and when it reaches zero the message is no longer retransmitted. Hence, the TTL defines the range of message propagation measured in hops. For example, a message emitted by IOP with TTL equal to 5 will propagate in the five-hop radius from IOP and then it will vanish.

Gossiping was mainly invented as an alternative to flooding [88]. The idea was to reduce the redundancy (i.e., to increase the efficiency) while maintaining comparable robustness. The gossiping is based on an idea similar to flooding: every node which receives a message, performs its re-emission to all of his neighbors. However, gossiping additionally introduces an element of randomness in the re-emission mechanism. The rule is simple: each message is re-emitted with some probability, called REP (re-emission probability). For example, if REP=0.8, the probability of performing each re-emission is 0.8 and 0.2 of discarding each re-emission. Note that since some of the re-emissions are omitted, the number of message copies in the network is smaller than in case of flooding. The re-emission probability may vary in very large range. The lower the REP, the fewer copies of message circulate in the network and the more efficient is the protocol. However, as there are few message copies in the network, some nodes may not even receive the message, so gossiping is less robust than flooding in terms of number of contacted nodes.

The number of total transmissions for gossiping in a NxN network cannot be easily calculated. Therefore, we have performed numerical simulations to study the efficiency of gossiping (see section 3.3.3, page 42). It must also be emphasized that gossiping with REP=100% corresponds to flooding. Limited gossiping, like limited flooding, introduces TTL field which allows limiting the message propagation to the specified range.

Route Field Forwarding

The request message, which is broadcast across the network, includes a special route field in its header. When the request message propagates through the network, each node fills the message's route field with the direction from which the message was received. We propose to use a simple two bit coding scheme: 00 for message arriving from north, 01 for message arriving from east, 10 for message arriving from south and 11 for message arriving from west. After *n* hops, the route field indicates *n* directions which specify exactly the route back to the IOP.

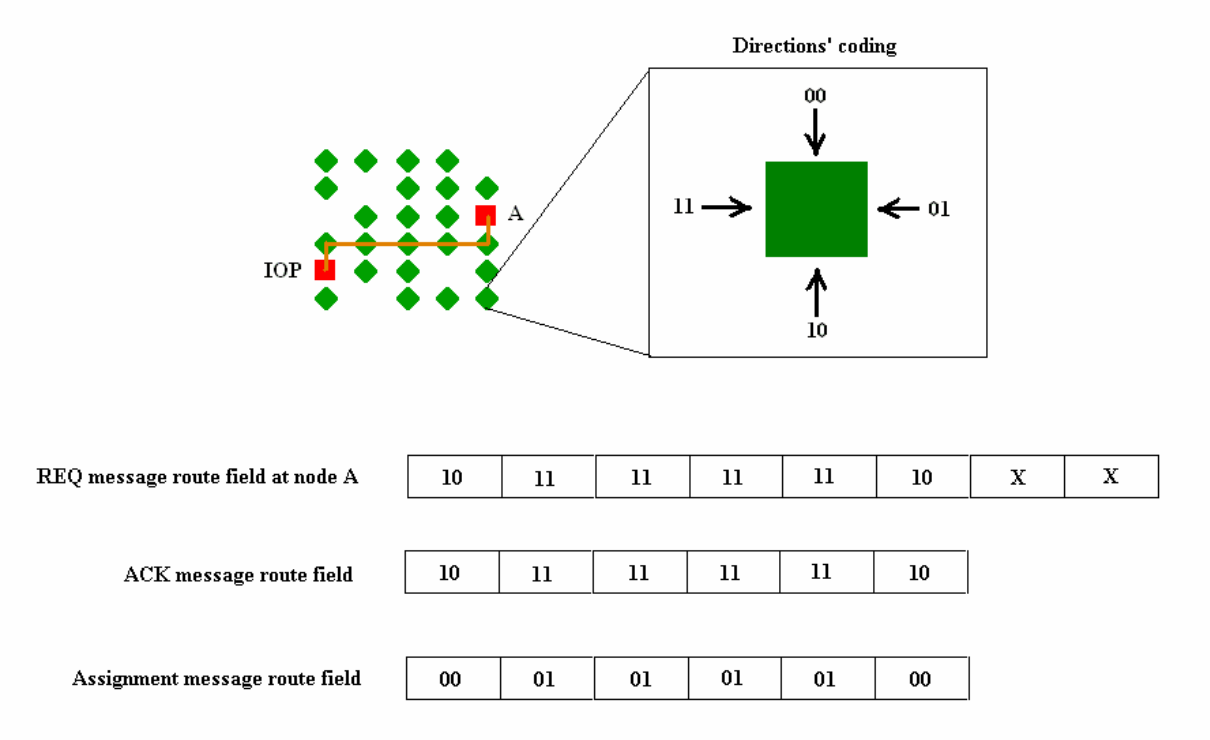


Fig. 86: Explanation of route field forwarding and the direction coding

Let us now analyze the example presented in the Fig. 86. The straight line connecting the IOP and node A indicates a sample propagation of one broadcast message copy. The first node sees that the message was received from South, so “10” is added to the route field. The next node sees that the message was received from West so it adds “11” to the route field. Thus, any node which receives the message has the complete coding of the route back to the IOP. For example, coding for the route between node A and IOP is shown in the figure above.

Thus, any node can store this route in its local memory. Since each node has de facto a complete route to the IOP, the ACK message can be simply unicast. More precisely, the destination node includes in the ACK message the same route which was received with REQ message. Then, the ACK message is routed according to the directions specified in the route field.

Then, the IOP receives an ACK message from the node and, similarly, it stores it in its local memory. Note that when the IOP receives an ACK message, it cannot directly use the route to contact the destination node because this field specifies the path from the node to the IOP, not from the IOP to the node. However, the IOP can easily derive the correct route code by:

- inverting the two-bit code to its complement in the terms of direction (00 becomes 10 and vice versa, 11 becomes 01 and vice versa) and,

- inverting the order of the route fields.

Appendix B: MASS simulator

MASS [89] is an integrated development environment (IDE), written in C++, to study the coexistence of large populations of agents. The basic structure of the application is presented in the following figure.

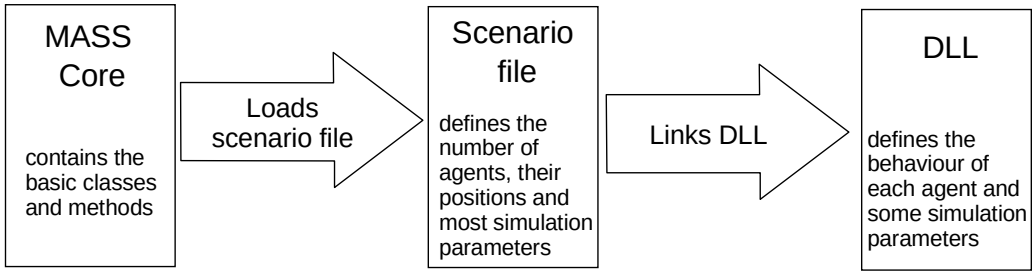


Fig. 87: The basic structure of MASS simulator

The core of MASS contains the basic classes and methods. The scenario file describes the agents' environment, the simulation parameters and the links with DLL files. In a DLL, the behaviour of each agent type is described, enabling communication and interaction between agents.

The application MASS is composed of two threads, namely the interface thread and the calculation thread. The interface thread is further decomposed into 5 quasi-independent modules (see Fig. 89), namely the basic Win32 MFC application classes, a C-like interpreter, a vectorial tool for drawing the initial agent distributions, a chart tool and some functions necessary for graphical display of agents. The calculation thread contains the routines necessary to evolve the agent populations, to build the representation of the environment of each agent, to identify the state of each agent, to evolve the agent policy by reinforcement learning and to execute actions.

One of the most important elements of MASS is the global scheduler (GS), whose role consists in activating sequentially the agents. Each agent is associated to a next action time (NAT) which is the time the agent is to be activated. The GS identifies the lowest NAT and triggers the ACTION cycle of the corresponding agent as shown in Fig. 88. In this example, the scheduler manages 5 agents and identifies the lowest-NAT agent; say for instance t2 for Agent2. Consequently, it triggers the ACTION cycle of Agent2. Then, the agent increments t2, and returns it to the scheduler, which retrieves the next lowest NAT for another agent cycle, and so on. The scheduler provides an asynchronous-operation mode.

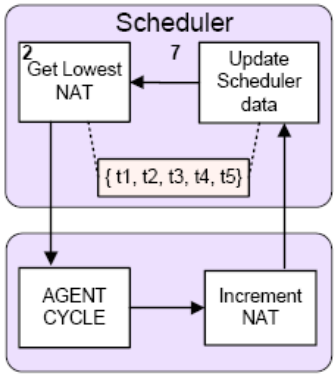


Fig. 88: Asynchronous scheduler in MASS simulator

In the ACTION mode, the agent scans its environment and its internal variables to decide an action. Moreover, a communicating agent also asks its node to sort the messages in the input buffers before deciding and action. Finally, it may also decide to send new messages as a consequence of its policy. In the NODE mode, the communicating agent scans its input buffers to process the incoming messages. The node processing consists in reemitting the message for other nodes, in transferring to a buffer for local processing, in detecting errors in the message, etc.

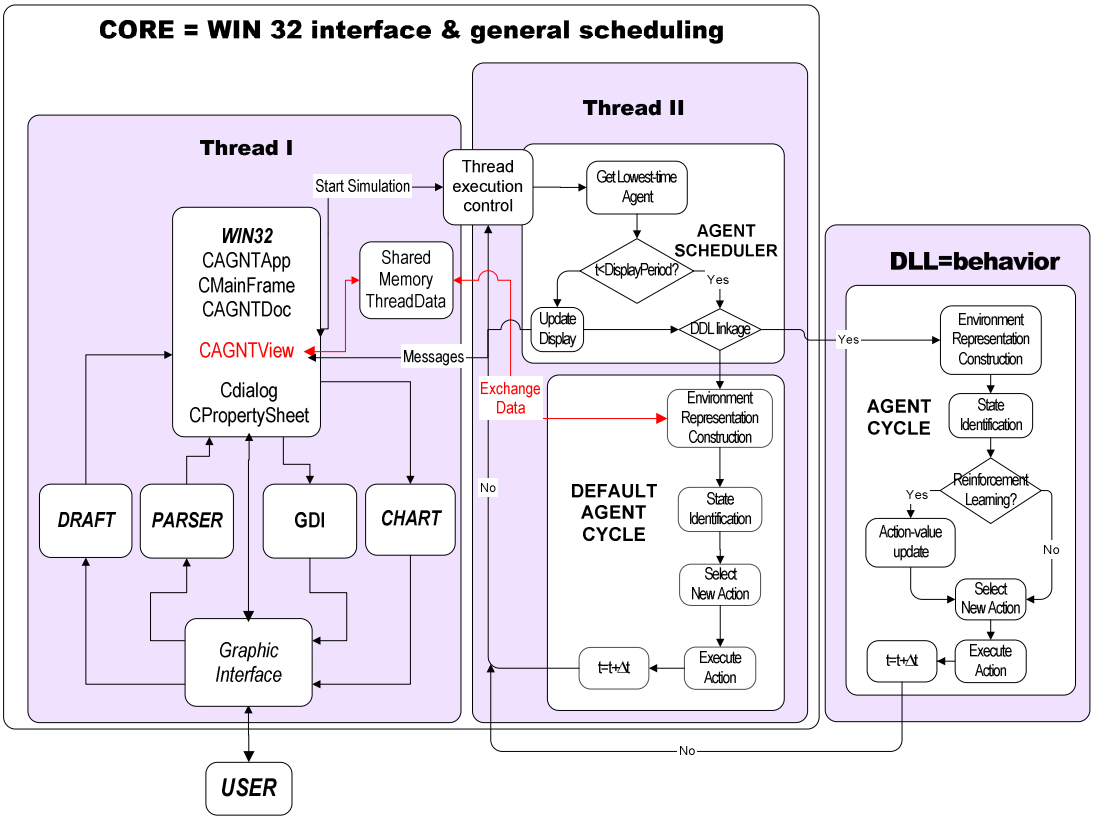


Fig. 89: The cooperation of two threads in MASS simulator

The core has been written in LAAS-CNRS laboratory in Toulouse under the supervision of J.H. Collet. During my work, I did not make significant changes to the core, as the agent actions may be entirely described in the DLL code. Moreover, many core functionalities

(from a wide range that MASS provides) were not used. For example, since I applied MASS to simulate processor network, I didn't use any of the functions implementing the agent movement. My role consisted of:

- creating a scenario in MASS,
- writing a code which represents the behavior of a processing core in a multiprocessor network,
- compiling the DLL and running a simulation.

In my work, the scenario describes of course a network of immobile communicating agents. However, depending on the network configuration (see section 4.2, page 52 for example), the network type may be created in many different forms. The scenario also defines many simulation parameters: the simulation time, the number of agents of each type, the size of communication disk, and other less important parameters like colors, chart type etc. Typically in the simulations that have been performed, the network consists of many agents of type 0 which represent processing cores and one or several agents of type 1 which represent the IOP(s).

Perhaps the most important role of the scenario is that it defines the DLL for each agent type. More precisely, it defines the actions of agents according to the DLL code. The functions in the DLL are called by the core. We must stress here that they are two types of agent actions: agent ACTION cycle and agent NODE cycle. In the NODE cycle, agent performs all only communication-related operations (receiving and reemitting messages, executing routing protocol). In the ACTION cycle, other operations necessary for the correct functioning of the agent are performed (analysis of messages received by the agent, generating new messages, executing tasks etc).

This execution method is extremely well designed for simulation of multiprocessor networks. Let us remind now the multiprocessor network architecture. Basically, it is composed of identical "tiles" and each tile is composed of a processor, memory and a router (see section 3.1, page 27).

This architecture clearly shows that the processor and the router are separate entities and therefore their actions have to be simulated separately. It can be easily seen that all router actions may correspond to an agent NODE cycle and all processor actions may correspond to an agent ACTION cycle. As a result, we have a natural solution for simulating the independent operation of the processor and the router. We present below a quick summary of the agent actions.

- The NODE cycle starts (controlled by the global scheduler).
- Message(s) is/are received in the NODE input buffers.
- If a message is intended to the agent, it is copied to the local buffer for further processing in the ACTION cycle.
- Each message is reemitted according to the rules of the routing protocol
- The NODE cycle ends.
- The ACTION cycle starts.
- If there is a new message in local buffer, it is decoded and an appropriate action is taken following the agent policy.
- If there is no new message, the agent proceeds with its previous work
- ACTION cycle ends.

My role as a programmer was to write only the code necessary for proper processor and router operation. As I already mentioned, I mainly worked with the code of a DLL. I had to write two DLLs, one to simulate the behavior of a processing core and one to simulate the behavior of IOP as well as the code to globally control the simulations.

For an extended description of MASS, see different manuals available from the URL http://www.laas.fr/~collet/.

Appendix C: Multiprocessor architectures

Parallel computing is no new discovery. Already in 1966, a taxonomy was proposed by Flynn [90] which divided the computer systems with regard to data and instruction streams. He distinguished:

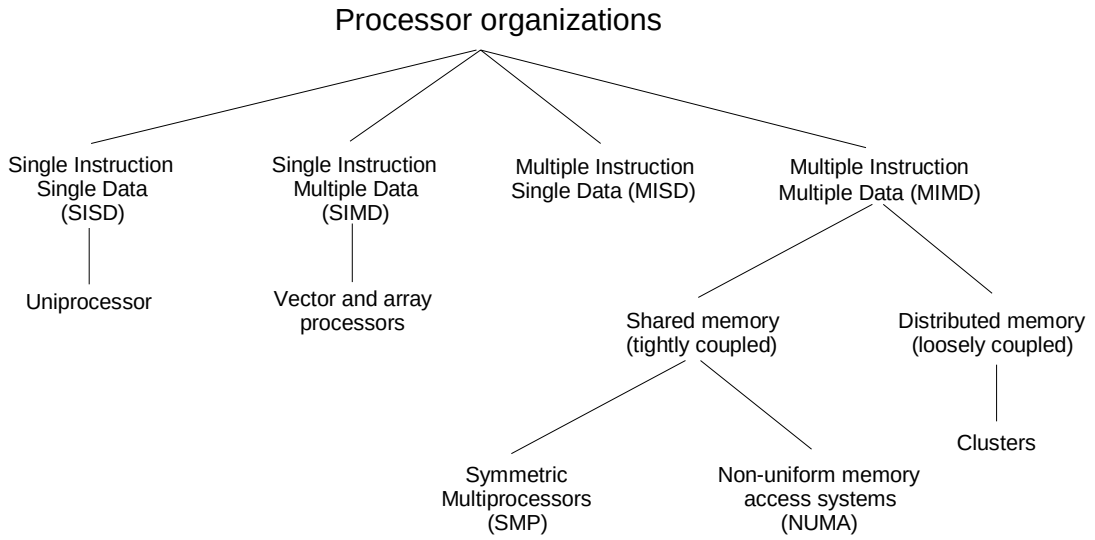


Fig. 90: Flynn's taxonomy of computer systems

- Single Instruction Single Data (SISD). At any clock cycle, CPU executes only one single instruction stream and uses only one data stream. Uniprocessors fall into this category.

- Single Instruction Multiple Data (SIMD). All processing units execute the same instruction during any clock cycle, but they use different data streams. The examples of such architectures are Array Processors (Maspar MP-1, MP-2) and Vector Processors (NEC SX-2, IBM 9000, Cray C90).

- Multiple Instruction Single Data (MISD). Processing units execute different instructions during any clock cycle, but they use the same data stream. This category is somewhat artificial and few implementations are known.

- Multiple Instruction Multiple Data (MIMD). At any clock cycle, the processing units execute different instructions using different data streams. Symmetrical multiprocessors (SMPs) like Pentium Core 2 Duo are the most common example of this category. Other examples include clusters and NUMA systems.

MIMD architectures can be further subcategorized into two main groups with regard to their memory architecture [91]:

- shared memory systems:

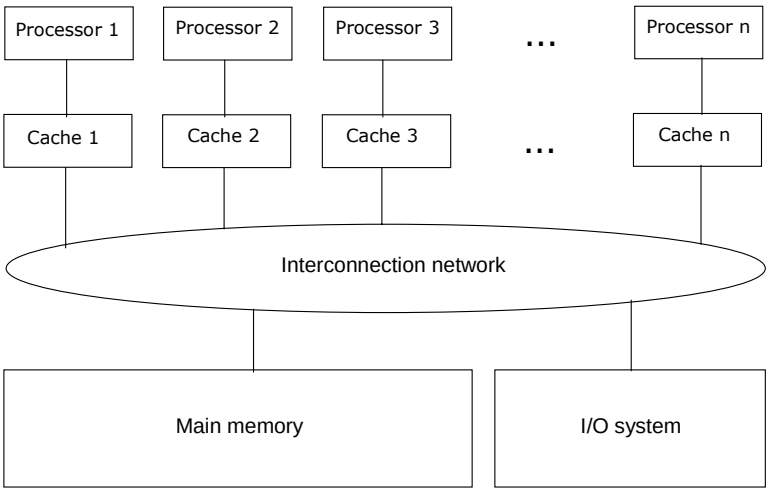


Fig. 91: Shared-memory system architecture

All processing cores share a common global memory address space. Naturally, the communication is performed via global address space because the changes made to the memory by any core are visible by all other cores. The advantages of such solution are easy programming and fairly easy inter-task communication. The drawback is that these systems are not scalable when it comes to CPU-memory communication. Adding more and more processors considerably increases traffic on the CPU-memory path so the overall chip performance does not scale proportionally with the number of processors. Moreover, cache coherency among the cores must be maintained.

We can distinguish two main types of shared memory architectures:

- UMA (Uniform Memory Access). This category is most commonly represented by Symmetrical Multiprocessors (see chapter 2.7.1).

- NUMA (Non-Uniform Memory Access). This category is usually represented by multiple computers connected via some high speed network like Ethernet. Each core can directly access the memory of any other core (global addressing). Note that the memory is shared among all processing units although it may be physically distributed. Therefore, each core has different access time to memory depending on its location. Generally, memory access time is slower than in shared memory architectures. Recently, as designers can integrate more and more transistors on chip, NUMA systems may be integrated in the chip, forming a chip multiprocessor (see section 2.7.4, page 24).

- distributed memory systems:

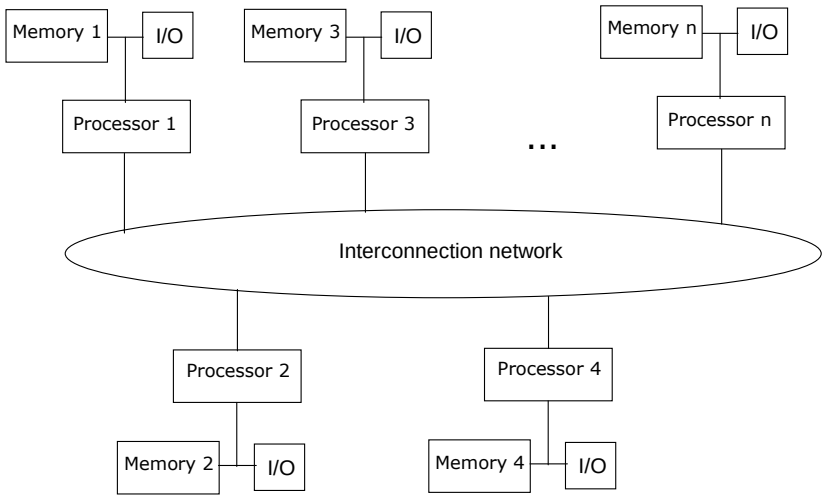


Fig. 92: Distributed memory system architecture

In these systems, there is no global address space because each core has its own memory. Naturally, this eliminates cache coherency problem. The main advantage of such systems is scalability: increasing the number of cores increases the memory size and the overall performance proportionally. However, the drawback of such systems is difficult programming, as it is programmer's responsibility to maintain the communication between tasks executed on different cores. This communication usually uses the Message-Passing Interface (MPI) so sometimes this processor architecture is called a Message-Passing multiprocessor.

Appendix D: Node importance factor

The node importance factor is a parameter that we introduce to evaluate the connectivity of the network (see section 4.3.1, page 71). First, let us calculate the number of routes in an *M*x*N* rectangle. More precisely, we calculate the number of possible routes from the upper left corner to the bottom right corner in rectangular *M*x*N* network. We will call the upper left corner point A, and the bottom right corner point B.

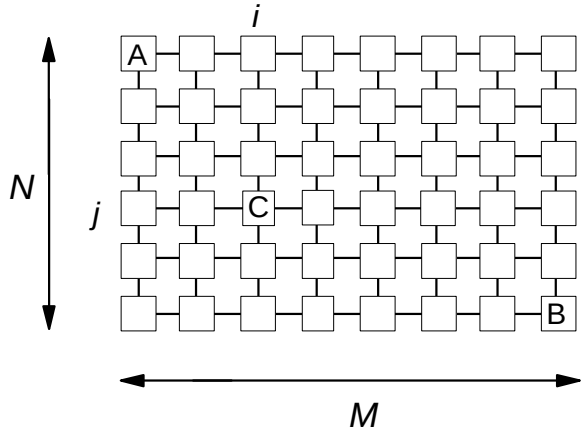


Fig. 93: Exemplary *M*x*N* network with node C at coordinates (*i*,*j*)

Note that we consider only the shortest routes, i.e. , we do not calculate routes that have more hops than the shortest possible route. In an *M*x*N* network the number of vertical hops is always *N*-1 and the number of horizontal hops is always *M*-1. Hence the minimum number of hops needed to go from point A to point B is *M*+*N*-2. A question arises: how many different paths are there from point A to point B? The answer is fairly simple. The number of possible routes is the number of (*M*-1) (or of course (*N*-1)) combinations from a set of (*M*+*N*-2) elements:

Eq. 39
$$r(M,N) = C_{M+N-2}^{M-1} = C_{M+N-2}^{N-1}$$

The above equation will be very useful in further calculations.

We introduce here a parameter which we call the node importance factor (*imp*). It represents the importance of a node to the network communication. Consider the rectangle *M*x*N* with node C located at coordinates (*i, j*). Now, what is the importance factor of node C to the routes from A to B? Of course, it is proportional to the network traffic pass through node C which in turn may be estimated as a percentage of routes passing through node C. In other words, the node importance factor can be defined as the percentage of routes which will vanish if the node is defective. For example in a one-dimensional network with nodes A and B at opposite ends, all nodes have the same importance factor 1 because if any one of them is defective, the communication between A and B will be impossible.

What is then the importance factor of node C?

The number of routes which pass through node C is the number of routes from A to C multiplied by number of routes from C to B (see Eq. 39):

Eq. 40
$$r_C = r_{A-C} \cdot r_{C-B} = C_{i+j}^{j} \cdot C_{M-i+N-j-2}^{N-j-1}$$

where the coordinates (*i*,*j*) represent the horizontal and vertical distance between node C and node A.

According to the statement above, the importance factor of a node is defined as:

Eq. 41
$$imp_C(i,j) = \frac{C_{i+j}^{j} \cdot C_{M-i+N-j-2}^{N-j-1}}{C_{M+N-2}^{N-1}}$$

For example, for the network shown in Fig. 93 with *M*=8, *N*=6, *i*=2, *j*=3 we have:

Eq. 42
$$r_C = C_5^3 \cdot C_7^2 = 210$$

Therefore the importance factor of node C is:

Eq. 43
$$imp_C(i,j) = \frac{C_{i+j}^{j} \cdot C_{m-i+n-j-2}^{n-j-1}}{C_{m+n-2}^{n-1}} = \frac{210}{792} \approx 0.265$$

Nevertheless, Eq. 41 represents the node's importance factor only in case when considering the communications between points A and B. It does not represent the overall node importance factor in the network. Consequently, a formula which describes global node important factor should be found. Consider now the network presented below.

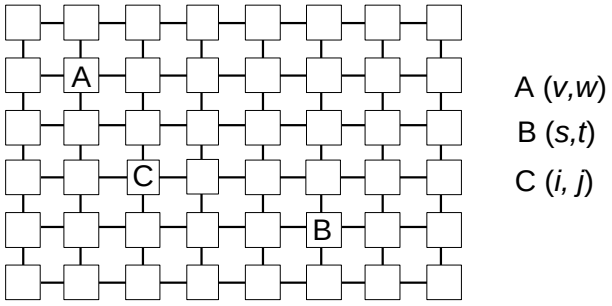


Fig. 94: Exemplary network with nodes A, B and C

We have already shown that the importance factor of node C located at position (*i, j*) to the communication between node A and node B is given by Eq. 41.

The fundamental question is: what is the global importance factor of node C in this network? To calculate it we have to sum up the importance factors of node C for all possible positions of A and B:

Eq. 44
$$imp_{OVERALL}(i,j) = imp(i,j) = \sum_A \sum_B imp_{A->B}(i,j)$$

The number of routes from the node A*(v,w)* to the node B*(s,t)* passing through node C*(i,j)* can be calculated as the number of routes from A to C multiplied by the number of routes from C to B. Hence, if we apply the formula from Eq. 39, we obtain the following:

Eq. 45
$$r_C = r_{A-C} \cdot r_{C-B} = C^{abs(w-j)}_{abs(v-i)+abs(w-j)} \cdot C^{abs(t-j)}_{abs(s-i)+abs(t-j)}$$

where *abs* (*x*) denotes the absolute value of variable *x*.

Obviously, the above equation is of valid only if node C is located in the rectangle created by points A and B, that is if

Eq. 46
$$min(v,s) \le i \le max(v,s) \quad \wedge \quad min(w,t) \le j \le max(w,t)$$

Accordingly, the importance factor can be defined as:

Eq. 47
$$imp_{A->B}(i,j) = \begin{cases} \dfrac{C^{abs(w-j)}_{abs(v-i)+abs(w-j)} \cdot C^{abs(t-j)}_{abs(s-i)+abs(t-j)}}{C^{abs(t-w)}_{abs(s-v)+abs(t-w)}} & min(v,s) \le i \le max(v,s) \quad \wedge \\ & min(w,t) \le j \le max(w,t) \\ 0 & otherwise \end{cases}$$

We have already stated that the global importance factor of a node is the sum of values of $imp_{A->B}(i,j)$ for every position of A and B:

Eq. 48
$$imp_{OVERALL}(i,j) = imp(i,j) = \sum_A \sum_B imp_{A->B}(i,j)$$

Hence, in the *M*x*N* network the importance factor of a node is (considering of course the conditions specified in Eq. 47):

Eq. 49
$$imp(i,j) = \sum_{v=1}^{M} \sum_{w=1}^{N} \sum_{s=1}^{M} \sum_{t=1}^{N} \frac{C^{abs(w-j)}_{abs(v-i)+abs(w-j)} \cdot C^{abs(t-j)}_{abs(s-i)+abs(t-j)}}{C^{abs(t-w)}_{abs(s-v)+abs(t-w)}}$$

It must be emphasized that Eq. 49 depends on network size. We can normalize it by dividing each importance factor *imp*(*i*,*j*) by the sum of all importance factors.

Consequently, the final formula for node's normalized importance factor is:

Eq. 50
$$imp_{NORMALIZED} = imp_N(i,j) = \frac{\sum_{v=1}^{M} \sum_{w=1}^{N} \sum_{s=1}^{M} \sum_{t=1}^{N} \frac{C^{abs(w-j)}_{abs(v-i)+abs(w-j)} \cdot C^{abs(t-j)}_{abs(s-i)+abs(t-j)}}{C^{abs(t-w)}_{abs(s-v)+abs(t-w)}}}{\sum_{i,j} imp(i,j)}$$

Appendix E: Comparison of execution methods

We analyze further Eq. 38 separately for each execution method described in section 5.3.3, page 105.

Execution on 1 processor

In this type of execution, the processing group is formed by only one processor. Therefore,

Eq. 51
$$N_{PG} = N_N$$

Let us now calculate the average task execution time T_e. It is equal to $2T_D$ when there is no errors and equal to $3T_D$ if there is at least one error during any execution:

Eq. 52
$$T_e = 2T_D(1-p)^2 + 3T_D(1-(1-p)^2) = T_D(-p^2 + 2p + 2)$$

where p is the probability that an error occurs during task duration.

Task allocation in this type of execution consists only of communication between IOP and the processor which executes the task. In the square network with N_N processors the average distance is:

Eq. 53
$$d = \frac{\sqrt{N_N}}{2}$$

Hence, the time needed to allocate the task is:

Eq. 54
$$T_a = 2 \cdot 1.5 \cdot \frac{\sqrt{N_N}}{2} = \frac{3\sqrt{N_N}}{2}$$

Therefore the final formula for the total number of tasks executed by N_N processors during time T_S yields:

Eq. 55
$$k_S = N_N \cdot \left(\frac{T_S}{T_D(-p^2 + 2p + 2) + \frac{3\sqrt{N_N}}{2}} - 0.5 \right) (p-1)^2 (2p+1)$$

Execution on 2 processors

In this type of execution, the processing group is formed by two processors. Therefore:

Eq. 56
$$N_{PG} = \frac{N_N}{2}$$

Let us now calculate the average task execution time T_e. It is equal to T_D when there is no errors and equal to $2T_D$ if there is at least one error during any execution. Note that we neglect here the time needed for comparison of fingerprints, as it is much smaller than the average task duration:

Eq. 57
$$T_e = T_D(1-p)^2 + 2T_D(1-(1-p)^2) = T_D(-p^2+2p+1)$$

The task allocation in this type of execution consists of communications between the IOP and the master processor which executes the task and additional communications between two processors within the group. Here, we simply assume that the average communication time (needed to compare results) between these two processors is equal to the average communication time between IOP and processors' pair. Note that we analyze the network without permanent faults so two processors may communicate using simple XY routing. Hence, the time needed to allocate the task is:

Eq. 58
$$T_a = 3 \cdot 1.5 \cdot \frac{\sqrt{N_N}}{2} = \frac{9\sqrt{N_N}}{4}$$

Therefore, the final formula for the total number of tasks executed by N_N processors during time T_S yields:

Eq. 59
$$k_S = \frac{N_N}{2} \cdot \left(\frac{T_S}{T_D(-p^2+2p+1) + \frac{9\sqrt{N_N}}{4}} - 0.5 \right) (p-1)^2 (2p+1)$$

Execution on 3 processors

In this type of execution, the processing group is formed by three processors. Therefore:

Eq. 60
$$N_{PG} = \frac{N}{3}$$

The average task execution time T_e is always equal to T_D. Note that we neglect here the time needed for comparison of fingerprints, as it is much smaller than the average execution time:

Eq. 61
$$T_E = T_D$$

Task allocation in this type of execution consists of communication between the IOP and the master processor which executes the task and the communication between three processors within the group. Again, we assume that the average communication time between processors (needed to compare results) is equal to the communication time between IOP and processing group. Note that we analyze the network without permanent faults so processors may communicate using simple XY routing.

Hence, the time needed to allocate the task is:

Eq. 62
$$T_a = 3 \cdot 1.5 \cdot \frac{\sqrt{N_N}}{2} = \frac{9\sqrt{N_N}}{4}$$

Therefore the final formula for the total number of tasks executed by N_N processors during time T_S yields:

Eq. 63
$$k_S = \frac{N}{3} \cdot \left(\frac{T_S}{T_D + \frac{9\sqrt{N_N}}{4}} - 0.5 \right) (p-1)^2 (2p+1)$$

References

[1] http://www.intel.com/technology/mooreslaw/index.htm

[2] A. Bhavnagarwala, X. Tang, and J.D. Meindl, "The impact of intrinsic device fluctuations on CMOS SRAM cell stability", *IEEE journal of Solid-state circuits* , vol. 36(4), pp. 658-665, (2001)

[3] A. Asenov, S. Kaya, and A. R. Brown, “Intrinsic parameter fluctuations in decananometer MOSFET’s introduced by gate line edge roughness", *IEEE Transactions on Electron Devices*, vol. 50, pp 1254–1260, (2003)

[4] A. Asenov, A.R. Brown, J.H. Davies, S. Kaya, and G. Slavcheva, "Simulation of intrinsic parameter fluctuations in decananometer and nanometer-scale MOSFETs", *IEEE Transactions on Electron Devices*, vol. 50(9), pp. 1837-1852, (2003)

[5] B. Cheng, S. Roy, and A. Asenov, "Impact Of Intrinsic Parameter Fluctuations On Deca-nanometer Circuits, And Circuit Modelling Techniques," *Proceedings of the International Conference on Mixed Design of Integrated Circuits and Systems*, pp. 117-121, (2006)

[6] The Semiconductor Reporter, Sept (2003)

[7] K. Nikolic, A. Sadek, and M. Forshawn "Fault tolerant techniques for nanocomputers", *Nanotechnology*, vol. 13, pp. 357-362, (2002)

[8] Y. Chen and T. Chen, “Implementing Fault-Tolerance via Modular Redundancy with Comparison,” *IEEE Transactions on Reliability*, vol. 39(2), pp. 217-225, (1990)

[9] S. Mukherjee, J. Emer, and S.K. Reinhardt, "Radiation-induced soft errors: an architectural Perspective", *Proceedings of 11th International Symposium on High-Performance Computer Architecture*, (2005)

[10] K.W. Harris. “Asymmetries in soft-error rates in a large cluster system” *IEEE Transactions on Device and Materials Reliability*, vol. 5(3) , pp. 336-342, (2005)

[11] Wei-Fen Lin, S.K. Reinhardt, and D.Burger “Reducing DRAM latencies with a highly integrated memory” *The Seventh International Symposium on High-Performance Computer Architecture*, pp. 301-312, (2001)

[12] D. Mallik, K. Radhakrishnan, J. He, C-P. Chiu, T. Kamgaing, D. Searls, and J.D. Jackson, "Advanced Package Technologies for High Performance Systems." *Intel Technology Journal* September (2005)

[13] S. Savastiouk, O. Siniaguine, J. Reche, and E. Korczynski, “Thru-silicon interconnect technology “ *Twenty-Sixth IEEE/CPMT International Electronics Manufacturing Technology Symposium*, pp. 122-128, (2000)

[14] http://www.semiconductor.net/article/CA6445435.html

[15] G. Lowney “Why Intel is designing multi-core processors”, *Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures*, pp. 113-113, (2006)

[16] R. Patel, S. Rajgopal, D. Singh, F. Baez, G. Mehta, and V. Tiwari, "Reducing Power in High-Performance Microprocessors," *35th Conference on Design Automation Conference (DAC'98),* pp. 732-737, (1998)

[17] http://www.epson.co.jp/e/newsroom/2005/news_2005_02_09.htm

[18] J. Cortadella, L. Lavagno, P. Vanbekbergen, and A.Yakovlev, „Designing asynchronous circuits from behavioural specifications with internal conflicts” *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 106-115, (1994)

[19] E. Beigné, F. Clermidy, P. Vivet, A. Clouard, and M. Renaudin, "An Asynchronous NOC Architecture Providing Low Latency Service and Its Multi-Level Design Framework," *11th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'05*), pp. 54-63, (2005)

[20] M, Krstić, E. Grass, F.K. Gürkaynak, and P. Vivet, "Globally Asynchronous, Locally Synchronous Circuits: Overview and Outlook," *IEEE Design and Test of Computers*, vol. 24(5), pp. 430-441, (2007)

[21] R. Dobkin, R. Ginosar, and C.P. Sotiriou “Data synchronization issues in GALS SoCs” *Proceedings of 10th International Symposium on Asynchronous Circuits and Systems*, pp. 170-179, (2004)

[22] Y. Zhiyi, and B.M. Baas, "Performance and power analysis of globally asynchronous locally synchronous multiprocessor systems" *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, vol. 00, pp. 6, (2006)

[23] L. Kleeman and A. Cantoni, "Metastable behavior in digital systems." *IEEE Design and Test of Computers* vol. 4(6), pp. 4-19, (1987)

[24] R. Mullins, and S. Moore, "Demystifying Data-Driven and Pausible Clocking Schemes," *13th IEEE International Symposium on Asynchronous Circuits and Systems*, pp.175-185, (2007)

[25] R. Dobkin, R. Ginosar, and C.P. Sotiriou "High Rate Data Synchronization in GALS SoCs**"** *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14(10), pp. 1063-1074, (2006)

[26] S. Moore, G. Taylor, R. Mullins, and P. Robinson "Point to point GALS interconnect" *Proceedings of Eighth International Symposium on Asynchronous Circuits and Systems*, pp. 69-75, (2002)

[27] S. Vangal, J. Howard, G. Ruhl, S, Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar, „An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS", *IEEE International Solid-State Circuits Conference, Digest of Technical Papers*, pp. 98-589, (2007)

[28] B.R. Gaeke, P. Husbands, X.S. Li, L. Oliker, K.A. Yelick, and R. Biswas."Memory-Intensive Benchmarks: IRAM vs. Cache-Based Machines" *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, (2002)

[29] H.F. Li, and C.C. Lau "A distributed multiprocessor traffic control system" *The IEEE Computer Society Second International Computer Software and Applications Conference*, pp. 259-264, (1978)

[30] C. McNairy, D. Soltis, "Itanium 2 processor microarchitecture" *IEEE Micro,* vol. 23(2), pp. 44-45, (2003)

[31] http://www.intel.com/cd/ids/developer/asmo-na/eng/95581.htm?page=2

[32] http://www.intel.com/products/processor/coresolo/

[33] A. Jantsch and H. Tenhunen (Eds.), "Networks on Chip," *Kluwer Academic Publishers*, (2003)

[34] L. Benini, and G. De Micheli, "Networks on Chips: A New SoC Paradigm," *Computer*, vol. 35(1), pp. 70-78, (2002)

[35] W. J. Dally, and B. Towles "Route Packets, Not Wires: On-Chip Interconnection Networks" *Proceedings of the 38th Design Automation Conference (DAC)*, (2001)

[36] J. Henkel, W. Wolf, and S. Chakradhar "On-chip networks: a scalable, communication-centric embedded system design paradigm" *Proceedings of 17th International Conference on VLSI Design*, pp. 845-851, (2004)

[37] Hongyu Chen, Chung-Kuan Cheng; A. B. Kahng, I.I. Mandoiu, Qinke Wang, and Bo Yao, "The Y architecture for on-chip interconnect: analysis and methodology" *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24(4), pp. 588-599, (2005)

[38] H. Abachi, and Al-Junaid Walker, "Network Expandability and Cost Analysis of Torus, Hypercube and Tree Multi-Processor Systems," *28th Southeastern Symposium on System Theory (SSST '96)*, pp. 426, (1996)

[39] L. Bononi, and N. Concer, "Simulation and analysis of network on chip architectures: ring, spidergon and 2D mesh,", *Proceedings of the Design Automation & Test in Europe Conference* vol. 2, pp. 30, (2006)

[40] W. J. Dally „Performance Analysis of k-ary n-cube Interconnection Networks**"** *IEEE Transactions on Computers*, vol. 39(6), pp. 775-785, (1990)

[41] L. Hammond, B. A. Nayfeh, and K. Olukotun, "A Single-Chip Multiprocessor," *Computer*, vol. 30(9), pp. 79-85, (1997)

[42] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, "A Case for Intelligent RAM," *IEEE Micro*, vol. 17(2), pp. 34-44, (1997)

[43] www.semiconductor.net/article/CA6445435.html

[44] http://techon.nikkeibp.co.jp/article/HONSHI/20071127/143093/cs-f3.jpg

[45] G. S. Sohi, and A. Roth, "Speculative Multithreaded Processors," *Computer*, vol. 34(4), pp. 66-73, (2001)

[46] http://www.intel.com/technology/magazine/research/speculative-threading-1205.htm

[47] C.G. Quinones, C. Madriles, J. Sanchez, P. Marcuello, A.Gonzalez, and D.M. Tullsen “Mitosis compiler: an infrastructure for speculative threading based on pre-computation slices” *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, vol. 40(6), pp. 269-279, (2005)

[48] F. J. Villa, M. E. Acacio, and J. M. Garcia „On the Evaluation of Dense Chip-Multiprocessor Architectures”, *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*, pp. 21-27, (2006)

[49] L. Shen, and S.Y.H. Su, **“**A functional testing method for microprocessors”, *IEEE Transactions on Computers*, vol. 37(10), pp. 1288-1293, (1988)

[50] J. Han and P. Jonker, "A defect- and fault-tolerant architecture for nanocomputers", *Nanotechnology*, vol. 14 pp. 224–230, (2003)

[51] K. D. Wagner "Robust Scan-Based Logic Test in VDSM Technologies," *Computer*, vol. 32(11), pp. 66-74, (1999)

[52] V. D. Agrawal, C. R. Kime, and K. K. Saluja, "A Tutorial on Built-in Self-Test. I. Principles," *IEEE Design and Test of Computers*, vol. 10(1), pp. 73-82, (1993)

[53] A. Giani, S. Sheng, M. S. Hsiao, and V. Agrawal, "Novel Spectral Methods for Built-In Self-Test in a System-on-a-Chip Environment," *19th IEEE VLSI Test Symposium*, pp. 163, (2001)

[54] Shyue-Kung Lu, and Shih-Chang Huang, "Built-in Self-Test and Repair (BISTR) Techniques for Embedded RAMs," *Records of the 2004 International Workshop on Memory Technology, Design and Testing (MTDT'04)*, pp. 60-64, (2004)

[55] L. Chen and S. Dey, “Software-Based Self-Testing Methodology for Processor Cores,” *IEEE Transactions on Compter-Aided Design of Integrated Circuits and Systems*, vol. 20(3), pp. 369-380, (2001)

[56] N. Kranitis, A. Paschalis, D. Gizopoulos, and Y. Zorian, "Instruction-Based Self-Testing of Processor Cores," *Journal of Electronic Testing: Theory and Applications*, no. 19, pp. 103-112, (2003)

[57] A. Krastic, L. Chen, W.C. Lai, K.T. Cheng, and S. Dey, “Embedded Software-Based Self-Test for Programmable Core-Based Designs,” *IEEE Design and Test of Computers*, pp. 18-26, (2002)

[58] D. Gizopoulous, A. Paschalis, and Y. Zorian "Embedded Processor-Based Self-Test", *Kluwer Academic Publishers*, (2004)

[59] L. Anghel, N. Achouri, and M. Nicolaidis "Evaluation of memory built-in self repair techniques for high defect density technologies," *Proceeding. on 10th IEEE Pacific Rim International Symposium Dependable Computing*, pp. 315-320, (2004)

[60] J. Maiz, S. Hareland, K. Zhang, and P. Amstrong, "Characterization of multi-bit soft error events in advanced SRAMs", *Proceedings of IEEE International Electron Device Meeting*, pp. 21.4.1 - 21.4.4, (2003)

[61] M.M. Bae, and B. Bose, "Spare processor allocation for fault tolerance in torus-based multicomputers," *Proceedings of Annual Symposium on Fault Tolerant Computing*, pp.282-291, (1996)

[62] B.M. Maziarz, and V.K. Jain, "Automatic reconfiguration and yield of the TESH multicomputer network," *IEEE Transactions on Computers*, vol.51(8), pp. 963-972, (2002)

[63] J. Han and P. Jonker, "A defect- and fault-tolerant architecture for nanocomputers", *Nanotechnology*, vol. 14, pp. 224–230, (2003)

[64] L.E. Laforge, K. Huang, and V.K. Agarwal, "Almost sure diagnosis of almost every good elements", *IEEE Transactions on Computers*, vol 43(3), pp. 295-305, (1994)

[65] A.D. Singh "Interstitial redundancy: An area fault-tolerant scheme for larger area VLSI processors arrays", *IEEE Transactions on Computers*, vol. 37(11), pp. 1398-1410, (1988)

[66] I. Koren and Z. Koren, "Defect Tolerant VLSI Circuits: Techniques and Yield Analysis," *Proceedings of the IEEE*, vol. 86, pp. 1817-1836, (1998)

[67] L. M. Ni, and P. K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *Computer*, vol. 26(2), pp. 62-76, (1993)

[68] Y. K. Dalal, and R. M. Metcalfe, “Reverse Path Forwarding of Broadcast Packets”, *Communications of the ACM*, vol. 21(12), pp.1040-1048, (1978)

[69] M. Chrobak, L. Gasieniec, and W. Rytter, "Fast broadcasting and gossiping in radio networks," *41st Annual Symposium on Foundations of Computer Science*, pp. 575, (2000)

[70] Jie Wu, Fei Dai, "A Generic Broadcast Protocol in Ad Hoc Networks Based on Self-Pruning," *International Parallel and Distributed Processing Symposium (IPDPS'03)*, pp. 29a , (2003)

[71] J. Lipman, P. Boustead, and J. Chicharo "Reliable optimised flooding in ad hoc networks" *Proceedings of the IEEE 6th Circuits and Systems Symposium onEmerging Technologies: Frontiers of Mobile and Wireless Communication*, vol. 2, pp. 521-524, (2004)

[72] Y.B. Kim, and Y-B. Kim, "Fault Tolerant Source Routing for Network-on-chip" *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*, pp. 12-20, (2007)

[73] L. Chunsheng, Z. Link, D.K. Pradhan, "Reuse-based test access and integrated test scheduling for network-on-chip" *Proceedings of Design, Automation and Test in Europe Conference*, vol. 1, pp. 6, (2006)

[74] J. Song, E. Li, H. Wei, S. Ge, L. Chunrong, Z. Yimin , Z. Xuegong, C. Wenguang, and Z. Weimin "Parallelization of Bayesian network based SNPs pattern analysis and performance characterization on SMP/HT," *Proceeding of Tenth International Conference on Parallel and Distributed Systems*, pp. 315-322, (2004)

[75] H. Hatano, "Radiation hardened high performance CMOS VLSI circuit designs" *IEE Proceedings G Circuits, Devices and Systems*, vol. 139(3), pp. 287-294, (1992)

[76] I. Koren, and S.Y.H. Su, "Reliability Analysis of N-Modular Redundancy Systems with Intermittent and Permanent Faults," *IEEE Transactions on Computers*, vol. 28(7), pp. 514-520, (1979)

[77] R.L. Graham, "Bounds for certain multiprocessing anomalies", *Bell System Technical Journal*, vol. 45(9), pp. 1563-1581, (1966)

[78] Y.-K. Kwok, and I. Ahmad "Static scheduling algorithms for allocating directed task graphs to multiprocessors" *CM Computing Surveys (CSUR)*, vol. 31(4), pp. 406-471, (1999)

[79] S. Ronngren, and B.A. Shirazi, "Static multiprocessor scheduling of periodic real-time tasks with precedence constraints and communication costs," *Proceedings of the Twenty-Eighth Hawaii International Conference on System Sciences*, vol. 2, pp.143-152, (1995)

[80] G. Manimaran, and C. Siva Ram Murthy, "An Efficient Dynamic Scheduling Algorithm for Multiprocessor Real-Time Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9(3), pp. 312-319, (1998)

[81] V.K. Madavarapu, M. Franklin, and K.K. Sundararaman, "A study of dynamic scheduling techniques for multiscalar processors," *Third International Conference on High-Performance Computing (HiPC '96)*, pp. 413, (1996)

[82] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy, "The directory-based cache coherence protocol for the DASH multiprocessor," *Proceedings of 17th Annual International Symposium on Computer Architecture*, pp.148-159, (1990)

[83] J.A. Brown, R. Kumar, and D. Tullsen "Proximity-aware directory-based coherence for multi-core processor architectures" *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, pp. 126 -134, (2007)

[84] R.C. Baumann "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on Device and Materials Reliability*, vol.5(3), pp. 305-316, (2005)

[85] F. Ruckerbauer, and G. Georgakos, "Soft Error Rates in 65nm SRAMs--Analysis of new Phenomena" *13th IEEE International On-Line Testing Symposium (IOLTS)*, pp. 203-204, (2007)

[86] J. C. Smolens, B. Gold, J. Kim, B. Falsafi, J. Hoe, and A. G. Nowatzyk, "Fingerprinting: Bounding Soft-Error-Detection Latency and Bandwidth," *IEEE Micro*, vol. 24(6), pp. 22-29, (2004)

[87] C. LaFrieda, E. Ipek, J.F. Martinez, and R. Manohar, "Utilizing Dynamically Coupled Cores to Form a Resilient Chip Multiprocessor,", *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pp. 317-326, (2007)

[88] Z. Haas, J.Y. Halpern, and L. Li, "Gossip-based Ad Hoc Routing", *Proceedings of IEEE INFOCOM*, vol. 3, pp. 23-27, 1707-1716 (2002)

[89] www.laas.fr/~collet

[90] M. Flynn "Some Computer Organizations and Their Effectiveness", *IEEE Transactions on Computers*, vol. C-21, pp. 948, (1972)

[91] W. Stallings "Computer Organization and Architecture: Designing for Performance" 5th edition, *Prentice Hall*, (1999)

Tolérance aux fautes par auto-configuration dans les futurs processeurs multi-coeurs

Cette thèse est une contribution au niveau architectural à l'amélioration de la tolérance aux fautes dans les puces multi-cœurs massivement défectueuses fabriquées à partir de transistors nanométriques. L'idée principale de ce travail est qu'une puce devrait être organisée en une architecture réplicative et devenir aussi autonome que possible pour augmenter sa résilience contre les défauts permanents et les erreurs transitoires apparaissant en opération. C'est pourquoi nous introduisons une nouvelle méthodologie d'autoconfiguration de la puce qui permet de détecter et isoler les coeurs défectueux, de désactiver les coeurs isolés, de configurer les communications et de diriger l'allocation et l'exécution des tâches. L'efficacité des méthodes est étudiée en fonction de la fraction de coeurs ou d'interconnections défectueux et du taux d'erreurs transitoires.

Mots-clefs : nanotechnologie, multi-cœur, tolérance aux fautes, auto-configuration, sûreté de fonctionnement, multiprocesseur sur puce

Fault tolerance through self-configuration in the future nanoscale multiprocessors

This thesis is a contribution at the architectural level to the improvement of fault-tolerance in massively defective multi-core chips fabricated using nanometer transistors. The main idea of this work is that a chip should be organized in a replicated architecture and become as autonomous as possible to increase its resilience against both permanent defects and transient faults occurring at runtime. Therefore, we introduce a new chip self-configuration methodology, which allows detecting and isolating the defective cores, deactivating the isolated cores, configuring the communications and managing the allocation and execution of tasks. The efficiency of the methods is studied as a function of the fraction of defective cores, of defective interconnects and soft error rate.

Keywords : nanotechnology, multi-core, fault-tolerance, self-configuration, dependability, chip multiprocessor