Prototyping methodology of image processing applications on heterogeneous parallel systems

Jinglin Zhang

Thèse

THESE INSA Rennes
sous le sceau de l'Université européenne de Bretagne
pour obtenir le titre de
DOCTEUR DE L'INSA DE RENNES
Spécialité : Electronique et télécommunications

présentée par
Jinglin ZHANG
ECOLE DOCTORALE : Matisse
LABORATOIRE : IETR CNRS UMR 6164

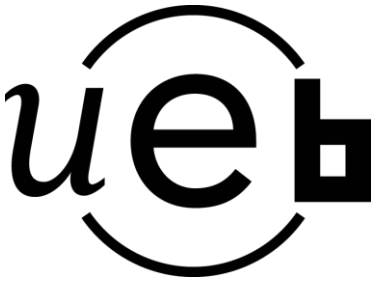
Prototyping methodology of image processing applications on heterogeneous parallel systems

Thèse soutenue le 19.12.2013
devant le jury composé de :

Michel PAINDAVOINE
Professeur des Universitésà l'Université de Bourgogne / Président
Dominique HOUZET
Professeur des Universités Grenoble-INP / Rapporteur
Guillaume MOREAU
Professeur des Universités Centrale Nantes / Rapporteur
Jean-Gabriel COUSIN
Maître de conférence à l'INSA de Rennes / Co-encadrant
Jean-François NEZAN
Professeur des Universités à l'INSA de Rennes / Directeur de thèse

Prototyping methodology of image processing applications on heterogeneous parallel systems

Jinglin ZHANG



To My Parents

Thank Prof. Jean-François Nezan!

Thank Dr. Jean-Gabriel COUSIN!

Thank China Scholarship Council!

Thank my father, ZHANG Dazeng!

Thank my mother, WANG Xiuling!

Thank my wife, TAN Huiwen!

Thank my son, ZHANG Tanyixi!

Thank all my friends!

Acknowledgments

The work of this thesis is one three-years work and life in INSA-Rennes. It is also a very important part in my life. In here, i have learned how to make research. First, i would like to thank my advisors Prof. Jean-François Nezan and Associate Prof. Jean-Gabriel Cousin for their help, suggestion and encouragement, for the time they spent to correct my papers and this PHD thesis. I don't forget that Jean-François revise my first conference paper again and again although he is very busy. I also remember that Jean-Gabriel teaches me to make some slides and to present my work. In particular, i thanks Matthieu Wipliez and Mickaël Raulet who helped me to debug the code when i was falling down with programming. I thanks Jean-François give me such a opportunity to study and make research here. I also thank China Scholarship Council to support my scholarship of 48 months living in Rennes, France.

Three year almost thousand days in our laboratory is quite a long time. I thank all the colleague Matthieu Wipliez, Nicloas Siret, Maxime Pelcat, Jérôme Gorin, Mickaël Raulet, Karol Desnos, Hervé Yviquel, Antoine Lorence, Khaled Jerbi, Julien Heulot, Erwan Raffin, who help me in the research and daily life. I enjoyed the laboratory life and its friendly environment, although i didn't communicate too much with you owing to my poor french speaking. Thanks you Hervé, have helped me making a phone call for house-hunting and something else. Maxime Pelcat, thank you being a listener and advisor in my presentations, also helping me to present the slides in the Dasip13 conference. Thanks Alexandre MERCAT's work on stereo matching on MPPA platform. Thanks to Aurore Gouin and Jocelyne Trenmier for managing administrative works.

I am grateful to all my friends, my Chinese colleagues. Yi Liu, Wenjing Shuai thank you for being my models in my experiment tests. Cong Bai, Ming Liu thank you for giving me the suggestions to revise papers and the templet of Ph.D thesis. Fu Hua thank you for helping me to write the abstract of French. Thank you all

the guys who invited me and prepared the meals almost every weekend when i was alone without my families.

Thank you, my wife Huiwen Tan, for taking care of our son Yixi. I apologize that i cannot live with you and take care of you in the more than three years. Although you always quarrel with me, i thank a lot because of your effort for our family, i am not a good husband and father until now. For my parents, father Dazeng Zhang, mother Xiuling Wang, thank you raising me up and teaching me when i was young. I can only to say i love you.

Contents

1 Introduction

1.1 Context

Inspired and motivated by the conclusion made by Gordon Moore in 1965, the density of transistors in a chip was doubled every 18 months. More transistors allow more complex chip-designs, and smaller transistor allows a higher frequency per watt consumed. But there is no more increasing about clock rates because of power consumption. In order to keep the performance increment, modern processors apply a many-cores strategy instead of frequency increment.

In the meantime, image and video applications such as video coding and ultra-resolution display with novel features whose aim is to recover a *real world* for users from the *digital world* are becoming more and more complex. Thereby, the need for computational power is rapidly increasing. The accuracy and the time-efficiency are two key criterions in the image and video processing applications. Better quality can be achieved by using more sophisticated algorithms, which also means the more computational power.

All these changes make electronic manufactures developing parallel embedded heterogeneous systems which combine with different subsystems optimized to execute different workload. The tradeoff between accuracy and time-efficiency will benefit from technology revolutions of the embedded heterogeneous system and parallel computing. Some heterogeneous systems consist of Central Processing Unit (CPU), Graphics Processing Unit (GPU), and Field-programmable gate array (FPGA). GPU with herds of Processing Element (PE) is the typical case which

benefits form many core revolutions that considered as special-purpose hardware for graphical computations until the turn of the century. In fact, GPU could employ general-purpose computation in many arithmetic computation domains. There are some successful attempts for general-purpose computing in heterogeneous systems. Even if they are so less mature and much more difficult to use for terminal users and to develop for researchers. With their immense processing power, GPU can be orders of magnitude faster than CPUs for numerically intensive algorithms that are designed to fully exploit the parallelism available. Lately, the interest in parallel programming has grown even more in popularity because of the availability of many core devices. The General-Purpose Graphics Processing Unit (GPGPU) research field has extended to the fields as diverse as artificial intelligence, medical image processing, physical simulation and financial modeling.

When the heterogeneous computing system became the trend of the hardware design and development in the domain of image and video processing, some parallel embedded System on Chips (SoC) like NVIDIA's Tegra, MediaTex's MT and Qualcomm's Snapdragon series comes up with a high speed development of personal portable devices like smartphone, TabletPC (Tablet Personal Computer) and so on. For these terminal users, parallel embedded heterogeneous systems bring better performance of applications and better user experience; For these researchers and programmers, most of the heterogeneous systems allow to select the best architecture for different demands of task or the best performance for fixed tasks. However developing software for heterogeneous parallel system is considered to be a non-straightforward task.

With so much heterogeneities, developing efficient solutions and applications for such a wide range of architectures faces a great challenge. In such heterogeneous systems, there are a huge diversity in hardware architectures and configurations. Meanwhile, these electronic manufacturers provide their standard and development environment for their heterogeneous systems like AMD's Streaming Programming Language: Brook, NVIDIA's Compute Unified Device Architecture (CUDA) and IBM's Unified Parallel C (UPC) and so on. But these tools and corresponding programming languages only support on their specified devices, are not compatible

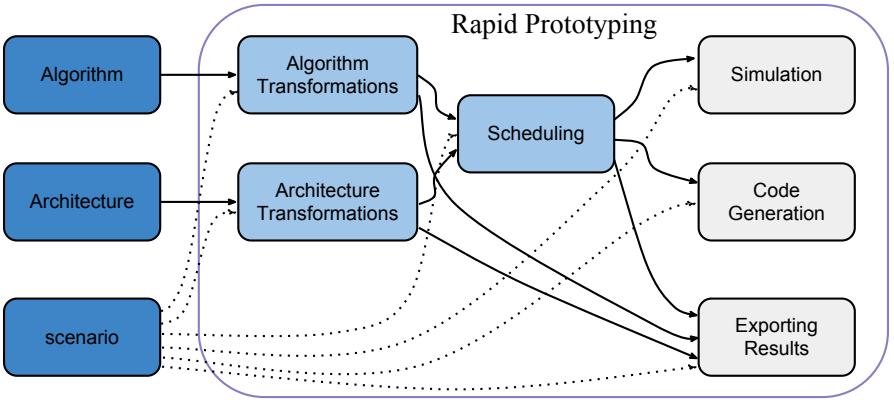


Figure 1.1: Rapid Prototyping framework

for other company's devices. In order to support different devices and different levels of parallelism, Open Compute Language (OpenCL) was proposed by Apple, NVIDIA, AMD, and IBM since 2009 year. OpenCL's standard provides the same Application Programming Interface (API) to manage multi-devices in the combined context and to distribute the global workload to these different devices. Although there are parallel computing programming language: CUDA and OpenCL, desinging the kernel code of CUDA and OpenCL is still a complex and troublesome procedure.

Implementing complex applications on such the same complex platforms have already been proven to be a daunting task. Under this context, there is a growing interest in adaptive/reconfigurable platforms that can dynamically adapt themselves to support or provide more flexible configuration and better performance for the application. A possible answer is to follow a model based approach, where both the hardware platform and the application are abstracted through models. These models are used to capture a given domain-specific knowledge into a formal abstract representation. Models of Computations (Kahn Process Network (KPN), Synchronous Dataflow (SDF), etc.) specify the behavior of a system and are a perfect example of such an abstraction. Similarly, platform models which abstract the hardware components of a system (processing resources, communication, etc.) are also a common abstraction for embedded platform designers. However, in spite of some early work done in this direction, there is currently still no modeling approach taking run-time adaptation into consideration (from both a hardware and software point of view).

A prototyping methodology is a software development process which allows developers to create portions of the solution to demonstrate functionality and make needed refinements before developing the final solution. The goal of rapid prototyping framework is to propose generic models for adaptive multi-processors embedded systems. Figure 1.1 describes a global view of rapid prototyping framework, there are lots of attempts and research works around this goal. Basically, our rapid prototyping framework contains:

1. Dataflow Models.
2. Model to Models transformations.
3. Friendly Graphical User Interface (GUI).
4. Backend for code generation

As a consequence, Conception Orientée Modèle de calcul pour multi-Processeurs Adaptables (COMPA) project is proposed by Institut d'Electronique et de Télécommunications de Rennes (IETR), Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA), Modaë Technologies, CAPS Entreprise, Laboratoire des Sciences et Techniques de l'Information-de la Communication et de la Connaissance (Lab-STICC), Texas Instrument France in 2011. The project addresses several issues related this challenge:

- It proposes to rely on a target independent description of the application, particularly focusing on dataflow Model of Computations (MoC) and especially the CAL language [1], developed in the Ptolemy project (Berkeley). The COMPA project will extend it with constructs for efficient modeling of multi-dimensional dataflow networks.
- To offer additional opportunities for optimizing the application implementation, it proposes to develop a static analysis toolbox for detecting underlying MoCs used in a given CAL description. This analysis will be coupled to optimizing transformations on CAL models.
- It specifies and develops a "Runtime Execution Engine" which will be in charge of the execution of the CAL network on the platform. Providing runtime execution and reconfiguration services imply solving many problems including

task-mapping, scheduling, etc. These problems themselves will not only take advantage of knowledge of the target architecture but also from meta data embedded in the CAL network description.

In the COMPA, three tools are proposed: the Open RVC-CAL Compiler (Orcc) [2], the Parallel and Real-time Embedded Executives Scheduling Method (Preesm) [3], and Hybrid Multicore Parallel Programming (HMPP) [4]. Orcc includes an RVC-CAL textual editor, a compilation infrastructure, a simulator and a debugger. Lots of works have been done with the Orcc and many backends are supported in the Orcc like *C, C++, VHDL, HMPP* and so on [5] [6] [7]. PREESM tool offers a fast prototyping tool for parallel implementations used in many applications like LTE RACH-PD algorithm and so on [8] [9] [10]. HMPP is a directive-based compiler to build parallel hardware accelerated applications. Previous research works with Orcc and Preesm was to evaluate the full prototyping framework developed in COMPA with video decoders. But video decoders are usually based on the same dataflow, the same structure with a lot of data dependencies which are very hard to map with parallelism and to implement on many core heterogeneous systems. Recently investigated motion estimation and stereo matching algorithm have the high nature of parallelism which can fully make use of the parallelism of target architectures. They are much more suitable to evaluate the efficiency of the rapid prototyping framework developed in COMPA.

1.2 Contributions

The goal of my Ph.D thesis was to evaluate and to improve the prototyping methodology for embedded systems, especially based on the dataflow modeling approach (high level modeling of algorithm) and OpenCL approach (intermediate level of algorithm). The first contribution of this thesis is to participate to the development of the rapid prototyping framework of COMPA project, as shown in Figure 1.2. I described the proposed motion estimation and stereo matching methods with RVC-CAL language. This rapid prototyping framework mainly contains three levels from up to down view: *high level programming model, intermediate level program-*

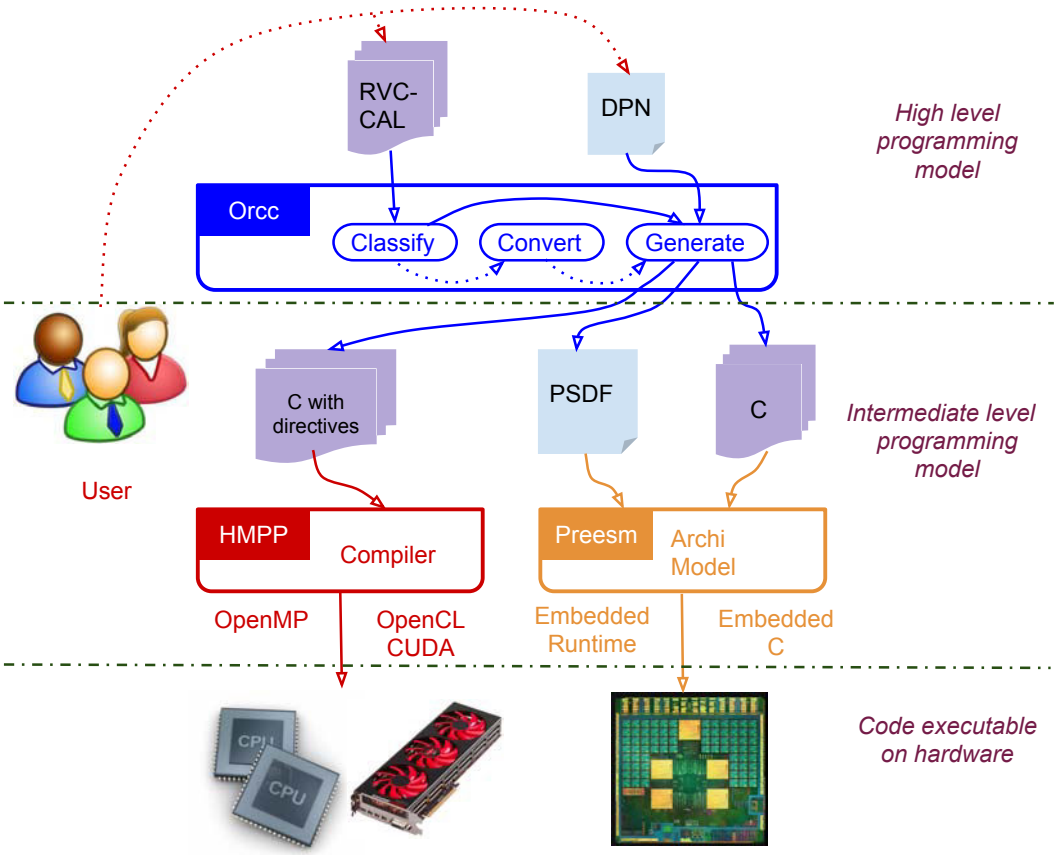


Figure 1.2: Proposed prototyping methodology in COMPA project

.

ming model, code executable on hardware which will be detailed in the next chapter of background. With the aid of the Orcc, Preesm, and HMPP, it can generate and verify C/OpenCL/CUDA code on heterogeneous platforms based on multi-core CPU and GPU platforms. Obviously there are two branches in this framework. One branch is $CAL \rightarrow Orcc \rightarrow HMPP \rightarrow GPU(OpenCL/CUDA)$ based on the HMPP backend research of Antoine Lorence and Erwan Raffin, the another one is $CAL \rightarrow Orcc \rightarrow Preesm \rightarrow DSP/ARM(EmbeddedC)$ based on research work of Maxime Pelcat and Karol Desnos. My research work as part of COMPA project was to rebuild the image and video algorithms with high-level descriptions, to verify the code generation. In the end, i target to the CPU, GPU and Multi-Purpose Processor Array (MPPA) of KALRAY as destination architectures to implement my applications.

The second contribution of this thesis contains three parts of improvement about

image and video processing algorithms:

- *The proposed parallelized motion estimation* (chapter 3) aims at heterogeneous computing system which contains one CPU and one GPU. I also developed one method to balance the workload distribution on such heterogeneous parallel computing system with OpenCL.
- *The proposed real-time stereo matching method* (chapter 4) adopts combined costs and costs aggregation with square size step to implement on an entry-level laptop's GPU platform. Experimental results show that the proposed method outperforms other state-of-the-art methods about tradeoff between matching accuracy and time-efficiency.
- *The proposed joint motion-based video stereo matching method* (chapter 5) makes use of the motion vectors calculated from our parallelized motion estimation method to build the support region for video stereo matching. Then it employs the proposed real-time stereo matching method to process these frames of stereo video as static paired images. Experimental results show that this method outperforms these state-of-the-art stereo video matching methods in the test sequences with abundant movement even in large amounts of noise.

1.3 Road Map

The content of this thesis are structured as follows: the fundamental concept of rapid prototyping framework and the images processing algorithms concerned in our approaches are introduced in chapter 2. Approach of parallelized motion estimation based on heterogeneous computing system is presented in chapter 3 while one new accurate method to distribute the workload in video applications based on heterogeneous computing system is proposed. Approach of real time local stereo matching is elaborated in chapter 4. Approach of joint motion-based video stereo matching is detailed in chapter 5 where one new method with joint motion vector to build the support region for stereo video matching is proposed. A conclusion will be given at chapter 6.

2 Background

This chapter gives an overview of our rapid prototyping methodology from down to up view. It describes the parallel embedded systems, parallel programming models, and the rapid prototyping methodology. The difference of embedded systems is discussed in the section of target architectures. OpenCL is presented as the intermediate level programming model in our rapid prototyping methodology. Dataflow approach is presented as the high level programming model in this methodology. Some tools used in this methodology are also introduced.

This chapter naturally begins by a presentation of target architectures in section 2.1; Section 2.2 introduces the intermediate level programming model - OpenCL; Section 2.3 presents the Dataflow approach in our rapid prototyping framework; section 2.4 presents some tools of rapid prototyping methodology. Section 2.5 introduces the concept of Motion Estimation and Stereo Matching algorithms; A brief conclusion is presented in section 2.6.

2.1 Target Architectures

Many core architectures like CPU, GPU, FPGA and DSP raised problems in terms of application distribution, data transferring and task synchronization. These problems become more and more complex and result in the loss of development time. Flynn's taxonomy [11] is the most popular classification of computer architecture which defines four categories of computer architecture according to the concurrency of instruction and data streams as shown in Figure 2.1. The categories are listed as

follows:

- Single Instruction, Single Data stream (SISD)
- Single Instruction, Multiple Data stream (SIMD)
- Multiple Instruction, Single Data stream (MISD)
- Multiple Instruction, Multiple Data stream (MIMD)

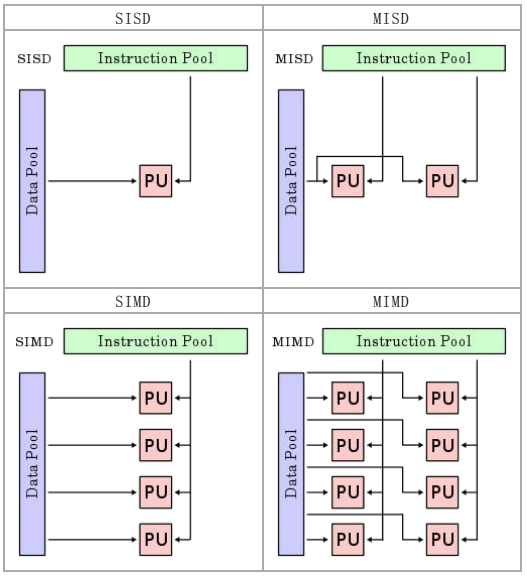


Figure 2.1: Flynn's taxonomy of classification of computer architecture, PU: Processing Unit

Classical single-processor systems belong to the category of SISD. A system of SIMD treats multiple data streams by using a single instruction stream, and it usually describes a processor array. In a system of MISD, multiple instructions operate on a single data stream. Modern parallel computing systems are mainly divided into two categories of SIMD (GPU, FPGA) and MIMD (Heterogeneous SoC System). MIMD is also divided into two groups: shared memory architecture and distributed memory architecture. These classifications are based on how MIMD processors access memory. Shared memory may be the bus-based, extended, or hierarchical type. Distributed memory may have hypercube or mesh interconnection schemes. We will introduce our shared memory MIMD architecture of heterogeneous system with GPU and CPU in chapter 3 of parallelized motion estimation.

2.1.1 CPU and GPU

The section begins by one most commonly comparing difference between CPU and GPU as shown in Figure 2.2. A CPU commonly has 4 to 8 fast, flexible cores clocked at 2-3 Ghz which favor threads of heavy workload with its instructions set like AMD's 3DNOW and Intel's Streaming SIMD Extensions (SSE), whereas a GPU has hundreds of relatively simple cores clocked at about 1Ghz that favor threads of light workload. Tasks that can be efficiently divided across many threads will see enormous benefits when running on a GPU. This highly parallel architecture is the reason why a GPU can process such large batches of copy number data so quickly.

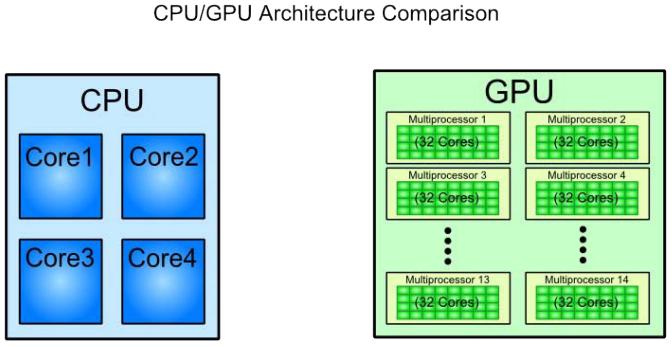


Figure 2.2: CPU/GPU architecture comparison

2.1.2 Platform MPPA Kalray

MPPA (Multi-Purpose Processor Array) of KALRAY is optimized to address the demand of high-performance, low-power embedded systems, making it the ideal processing solution for low to medium volume applications, requiring high processing efficiency at low power consumption. The first member of the MPPA MANYCORE family integrates 256 cores and high-speed interfaces (PCIe, Ethernet and DDR) to communicate with the external world. MPPA 256 provides more than 500 billion operations per second, with low power consumption, positioning MPPA MANYCORE as one of the most efficient processing solutions in the professional electronic market.

In fact MPPA has 256 VLIW cores per chip, organized in 16 clusters of 16 cores,

interconnected by a high bandwidth Network-on-Chip as shown in Figure 2.3 . Each cluster possesses 2M shared memory, and all these 16 cores of one cluster could directly access the shared memory. Each core has standard I/Os and interfaces to communicate with the outside. The parallelism of such a platform is very important in order to make the maximum cores work together.

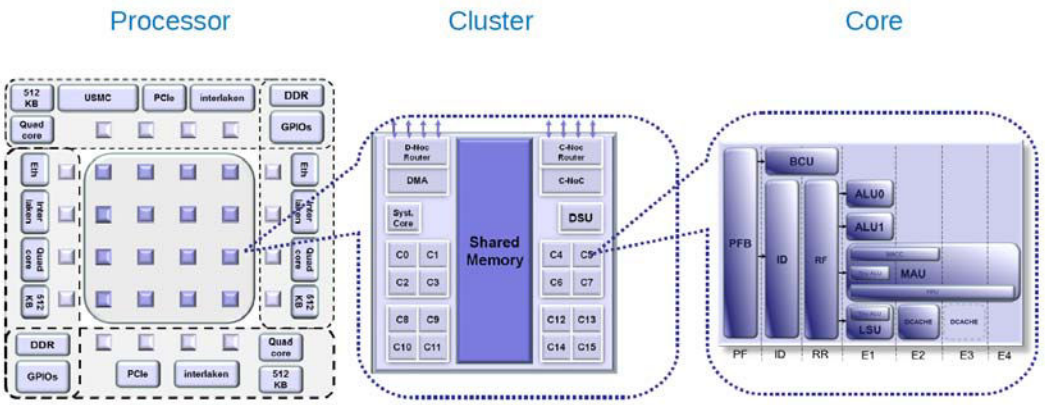


Figure 2.3: Structure of MPPA Platform

2.2 OpenCL: An Intermediate Level Programming Model

In fact we need to manage many core devices like CPU, GPU, SoC (System on Chip), to generate multiple target codes and to distribute suitable workload for different devices. CUDA is a parallel computing platform and programming model created by NVIDIA and implemented by the GPU that they produce. CUDA makes developers access to the virtual instruction set and memory of the parallel computational elements in CUDA GPUs. Different with CUDA, Open Computing Language (OpenCL) is appearing as a standard for parallel programming of diverse heterogeneous hardware accelerators. For existing CUDA projects or implementations of application, there are some useful tool like Swan [12] which could translate the kernel code from CUDA to OpenCL. It does several useful things:

- Translates CUDA kernel source-code to OpenCL.
- Provides a common API that abstracts both CUDA and OpenCL runtimes.

– Preserves the convenience of the CUDA kernel launch syntax by generating C source-code for kernel entry-point functions.

It can also be usefully used for compiling and managing kernels written directly for OpenCL.

OpenCL is such a programming language which expresses the application by encapsulating its computation into kernels. In our rapid prototyping framework, OpenCL is treated as an intermediate level programming model. The OpenCL compiler aims to parallelize the execution of kernel instances at all the levels of parallelism. Comparing with the traditional C programming language that is sequential, OpenCL enables higher utilization of parallelism available of hardware while still keeping familiar grammar with the C language. Whereas, OpenCL enables application portability but does not guarantee performance portability, eventually requiring additional tuning of the implementation to a specific platform or to unpredictable dynamic workloads. In OpenCL programming model, thread and thread group are

Table 2.1: Relationship between GPU device and OpenCL

Devices	**OpenCL**
thread	work-item
threads group	work-group
shared memory	local memory
device memory	global memory

equal to work-item and work-group respectively as described in Table 2.1. Work-item is the basic unit of OpenCL kernel execution, and all the work-items grouped in one work-group execute the same instruction at the same time. The *barrier*() function is required to ensure completion of reads and writes to local memory. All work-items in a work-group executing the kernel must execute *barrier*() function before any are allowed to continue execution beyond the barrier. This function must be encountered by all work-items in a work-group executing the kernel.

Local and global memories in OpenCL context correspond to the shared and device memories in specified device respectively. As shown in Figure 2.4, each work-group executes on one Compute Unit (CU), each work-item executes on one Processing Element (PE). Private memory is assigned to one work-item. Local memory is

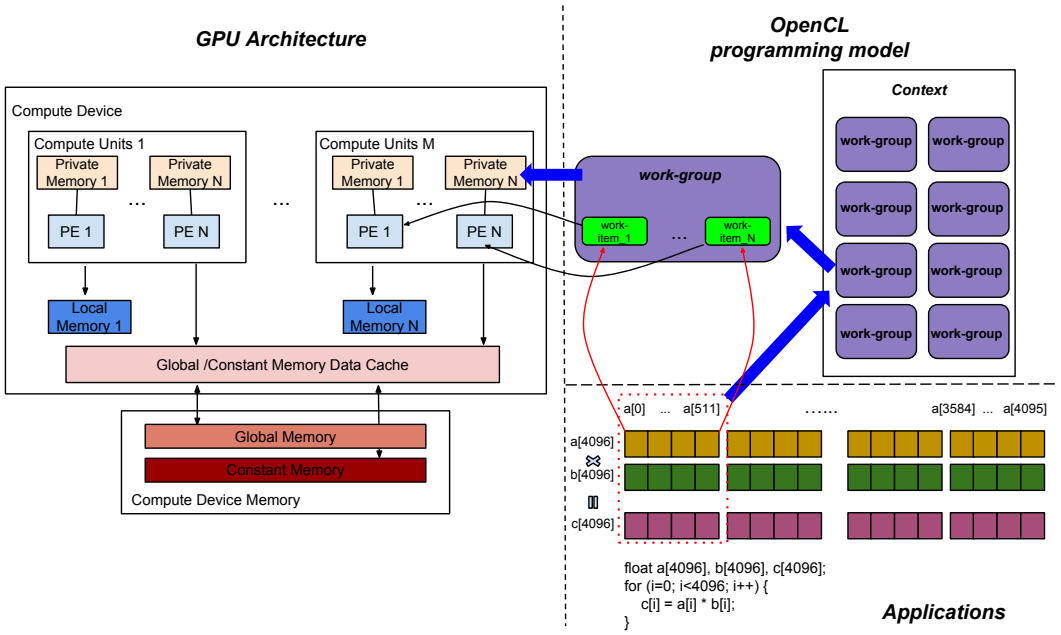


Figure 2.4: GPU device architecture and OpenCL programming model

shared by all the work-items in the same work-group. Global and constant memory are visible to all work-groups. OpenCL introduces an additional concept which is different with CUDA: Command Queues. Commands launching kernels and accessing memory are always issued for a specific command queue. A command queue is created on a specific device in a context.

One example of vector multiplication illustrates how the data map on the OpenCL programming model with GPU architecture in Figure 2.4. At first, all the data of array a[4096], b[4096] and c[4096] are stored in the global memory. Then 512 work-items in one work-group load a[0] to a[511], b[0] to b[511] and c[0] to c[511] to local memory before multiplication computation. Each work-item executes $c = a \times b$ until all the work-items reach the synchronous point with $barrier()$ function. The left will be the same until the execution complete.

Recently, there are so many programming tools and integrated development environment to use and evaluate OpenCL as illustrated in [13], [14], [15]. Du *et al.* [13] evaluated OpenCL as a programming tool for developing performance-portable applications for GPGPU They chose triangular solver (TRSM) and matrix multiplication (GEMM) as representative level 3 Basic Linear Algebra Subprograms (BLAS) routines to implement with OpenCL, profiled TRSM to get the time distribution

of the OpenCL runtime system, and provided tuned GEMM kernels for both the NVIDIA Tesla C2050 and ATI Radeon 5870. Experimental results described that nearly 50% of peak performance can be obtained in GEMM on both GPUs with OpenCL. Paone *et al.* [14] presented a methodology to analyze the customization space of an OpenCL application in order to improve performance portability and to support dynamic adaptation. They formulated their case study by implementing an OpenCL image stereo-matching application customized to the STMicroelectronics Platform 2012 [16]. They used design space exploration techniques to generate a set of operating points that represent specific configurations of the parameters allowing different trade-offs between performance and accuracy of the algorithm itself. Jinen *et al.* [15] described one methodology involved in applying OpenCL as an input language for a design flow of application-specific processors. The key of the methodology is a whole program optimizing compiler that links together the host and kernel codes of the input OpenCL program and parallelizes the result on a customized statically scheduled processor.

Since the programmable GPU make its way to mobile devices, it is interesting to study the new use-cases here. To this end, Leskela *et al.* [17] of Nokia Corporation created a programming environment based on the embedded profile of the OpenCL standard and verify it against an image processing workload in a mobile device with CPU and GPU back-ends. The early results on performance with CPU + GPU configuration suggested that there is enough room for optimization. Our experimental results based on heterogeneous systems also illustrate the performance enhancement based on the CPU + GPU combination in chapter 3. In the meantime, more and more portable device start to support OpenCL standard. The Mali OpenCL SDK [18] provides developers a framework and series of samples for developing OpenCL 1.1 application on ARM Mali based platforms such as the Mali-T600 family of GPUs. The samples cover a wide range of use cases that use the Mali GPU to achieve a significant improvement in performance when compared to running on the embedded CPU alone. QUALCOMM also release the Adreno SDK [19] for developing and optimizing OpenCL applications for Snapdragon 400, 600 and 800-based mobile platforms that include the Adreno 300 series GPU and Krait CPUs both.

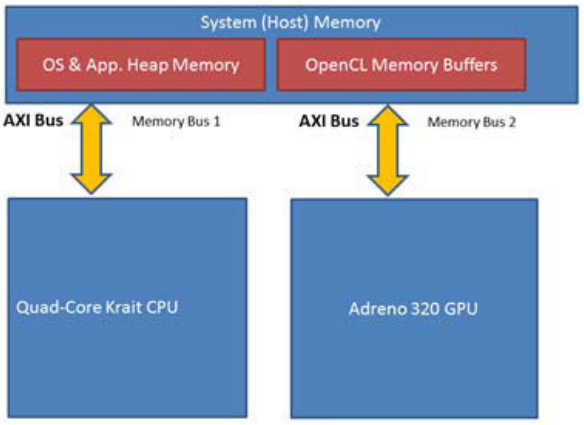


Figure 2.5: Snapdragon 600 (APQ8064)'s Shared Memory between Host and the Device

In such a SoC platform, CPU and GPU share the same external memory (global memory) which avoids enormous cost of communication and data exchanging like the communication between desktop-level CPU and GPU. It also means that we can fully exploit the potential of OpenCL on heterogonous parallel computing device to address some high computation applications. While the CPU and GPU within Snapdragon have different architecture and performance characteristics, OpenCL provides a programming environment that enables applications to accelerate data parallel computations that leverage both devices.

For example, in the Snapdragon 600 there is a Quad-Core Krait CPU and Adreno 320 GPU. Both the CPU and GPU access to system (host) memory where the input and output data for computations can be stored as shown in Figure 2.5. All calls to OpenCL API functions on the Snapdragon processor are performed on the CPU. The data parallel workloads are written in OpenCL C kernels. The kernels can be built (compiled) to run on either one or both of the Krait CPU and Adreno GPU. In the case of Snapdragon, the application can choose to create command queue(s) for either one or both devices (CPU or GPU). It is up to the application to decide how to partition work so that it can maximally make use of the compute resources of the CPU and GPU devices. Similarly we will discuss about the workload partition and propose our method about workload distribution in the chapter 3.

As above description, heterogeneous systems provide new opportunities to increase the performance of parallel applications on clusters with CPU and GPU architectures like [20], [21], [22]. Barak *et al.* [20] presented a package for running OpenMP, C++ and unmodified OpenCL applications on clusters with many GPU devices. Many GPUs Package (MGP) includes an implementation of the OpenCL specifications that allow applications on one hosting-node to transparently utilize cluster-wide devices (CPUs and/or GPUs). MGP provides means for reducing the complexity of programming and running parallel applications on clusters. Herlihy *et al.* [21] proposed a new methodology for constructing non-blocking and wait free implementations of concurrent objects whose representation and operations are written as stylized sequential programs with no explicit synchronization. Aoki *et al.* [22] proposed Hybrid OpenCL, which enables the connection between different OpenCL implementations over the network. Hybrid OpenCL consists of a runtime system that provides the abstraction of different OpenCL implementations and a bridge program that connects multiple OpenCL runtime systems over the network. Hybrid OpenCL enables the construction of the scalable OpenCL environments which enables applications written in OpenCL to be easily ported to high performance cluster computers; thus, Hybrid OpenCL can provide more various parallel computing platforms and the progress of utility value of OpenCL applications.

Different with the above OpenCL-based methodologies, our work in this thesis is to generate, to evaluate and to improve OpenCL applications with rapid prototyping methodology. The upper level of this methodology will be detailed in the next section.

2.3 The Dataflow Approach: An High Level Programming Model

The complexity introduced by the wide range of hardware architectures makes the optimal implementation of applications difficult to obtain. Moreover the stability of the application has to be early proved in the development stage to ensure the

reliability of the final product. Some tools such as PeaCE [23], SynDEx [24] aim at providing solutions to these problems by automatic steps leading to a reliable prototype in a short time. The aim of rapid prototyping methodologies is from a high-level description of these applications to its real-time implementations on target architecture as automatically as possible.

2.3.1 Definition of reconfigurable video coding

The Reconfigurable Video Coding (RVC) [25] defines a set of standard coding techniques called Functional Units (FUs). FUs form the basis of existing and future video standards, and are standardized as the Video Tool Library (VTL). A FU is described with a portable, platform-independent language called RVC-CAL. Video decoding process is described as a block diagram in RVC, also known as network or configuration, where blocks are the aforementioned FUs. To this end, RVC defines a XML-based format called FU Network Language (FNL) that is used for the description of networks. FNL is another name for the XML Dataflow Format (XDF). A FNL network may declare parameters and variables, has interfaces called ports, where a port is either an input port or an output port, and contains a directed graph whose vertices may be instances of FUs from the VTL or ports.

Figure 2.6 shows an example of the FNL network that represents sobel filter. The block "openImage" indicates the function of *load_file*, a FU that loads image file, "sobel" indicates the function of *sobel_filter*, a FU that executes the sobel filter algorithm and "dispImage" indicates the function of *display*, a FU that shows the final result in the screen. Edges carry data between a source port of the diagram or of an instance to a target port of the diagram or of another instance.

Figure 2.6: Sobel Filter XML Dataflow Format

Describing applications as a network of FUs rather than a monolithic C or C++

program has several advantages. First of all, it is no longer necessary to define profiles, rather a decoder may use any arbitrary meaningful combination of FUs. Additionally, this allows applications to be reconfigured at runtime by changing the structure of the network that defines the decoding process. This is especially interesting for hardware and memory-constrained devices. Finally, this makes RVC more *hardware-friendly* because dataflow is a natural way of describing hardware architectures.

2.3.2 Dataflow models of computation

A dataflow Model of Computation (MoC) defines the behavior of a program described as a dataflow graph. A dataflow graph is a directed graph whose vertices are actors and edges are unidirectional First-In-First-Out (FIFO) channels with unbounded capacity, connected between ports of actors. The networks of FUs described by the RVC standard are dataflow graphs. Dataflow graphs respect the semantics of Dataflow Process Networks (DPNs) [26], which are related to Kahn Process Networks (KPNs) [27] in the following ways:

1. Those models contain blocks (processes in a KPN, actors in a DPN) that communicate with each other through unidirectional, unlimited FIFO channels.
2. Writing to a FIFO is non-blocking, i.e. a write returns immediately.
3. Programs that respect one model or the other must be scheduled dynamically in the general case [28].

The main difference between the two models is that DPNs adds non-determinism to the KPN model, without requiring the actor to be non-determinate, by allowing actors to test an input port for the absence or presence of data [26]. Indeed, in a KPN process, reading from a FIFO is blocking: if a process attempts to read data from a FIFO and no data is available, it must wait. Conversely, a DPN actor will only read data from a FIFO if enough data is available, and a read returns immediately. As a consequence, an actor need not be suspended when it cannot read, which in turn means that scheduling a DPN does not require context-switching nor concurrent processes.

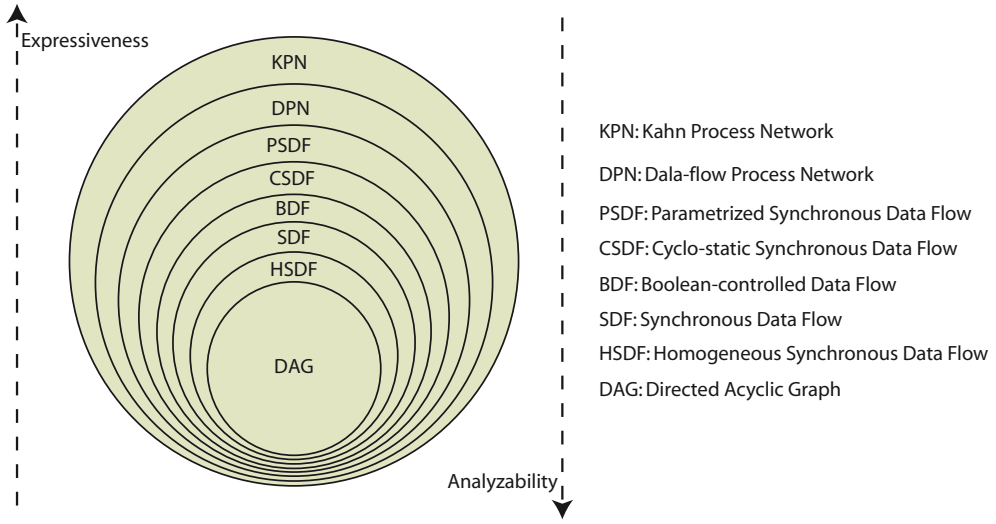


Figure 2.7: Data-flow Models of Computation (MoC) taxonomy

The CAL dataflow programming model is based on the DPN model, where an actor may include multiple firing rules. Many variants of dataflow models have been introduced in Figure 2.7: Data-flow MoC taxonomy. RVC-CAL is expressive enough to specify a wide range of programs that follow a variety of dataflow models. A CAL dataflow program can fit into those models depending on the environment and the target application.

Dataflow process network model

Each FIFO channel is a DPN carries a sequence of tokens $X = [x_1, x_2...]$, where each x_i is called a *token*. The sequence of available tokens on the P^{th} input port is X_p. An empty FIFO $\perp$ corresponds to the empty sequence. If a sequence X belong another sequence Y, for example $X = [1, 2, 3, 4], Y = [1, 2, 3, 4, 5, 6]$, we can define that $X \sqsubseteq Y$.

The set of all possible sequences is defined as S, and S^p is the set of p-tuples of sequence. In other words $[X_1, X_2, ..., X_P] \in S^p$.

An actor executes or fires when at least one of its firing rules is satisfied. Each firing consumes and produces tokens. An actor has N firing rules:

$$R = [\mathbf{R}_1, \mathbf{R}_2, ..., \mathbf{R}_N] \tag{2.1}$$

A firing rule $\mathbf{R}_i$ is a finite sequence of patterns, one for each of the p input ports of the actor:

$$\mathbf{R}_i = [P_{i,1}, P_{i,2}, ..., P_{i,p}] \in S^p \tag{2.2}$$

A pattern rule $P_{i,j}$ defines an acceptable sequence of tokens: if $P_{i,j} \sqsubseteq X_j$, the pattern is satisfied for the sequence of unconsumed (or available) tokens on the p^{th} input port; if $P_{i,j} = \perp$, the pattern is satisfied for any sequence, which is different from $P_{i,j} = [*]$ that defines a pattern satisfied for any sequence containing at least one token.

Synchronous dataflow model

The Synchronous Dataflow (SDF) [29] model is one of the most studied in our applications. SDF is the least expressive DPN model, but it is also the model that can be analyzed more easily. The SDF model is a special case of DPN where actors have static firing rules. They consume and produce a fixed number of tokens each time they fire. Any two firing rules $\mathbf{R}_a$ and $\mathbf{R}_b$ of an SDF actor must consume the same amount of tokens:

$$|\mathbf{R}_a| = |\mathbf{R}_b| \tag{2.3}$$

SDF may be easily specified in CAL constraining actions to have the same token consumption and production. Moreover, production and consumption rates may be easily extracted from an actor to check the SDF properties of actors at compile time. A key feature of this model is that a static code analysis detects if the program can be scheduled at compile time. A static analysis produces a static schedule (a predefined sequence of actor firings), if it exists, which is free of deadlock and that uses bounded memory.

Cyclo-static Dataflow (CSDF) [30] extends SDF with the notion of state while retaining the same compile-time properties concerning scheduling and memory consumption. State can be represented as an additional argument to the firing rules and firing function, in other words it is modeled as a self-loop. The position of the state argument (if any) is the first argument of a firing rule, i.e. it comes before patterns. The equations defined in the previous section for SDF can be naturally extended to

express the same restrictions (fixed production/consumption rates) for each possible state of the actor. Like SDF, CSDF graphs can be scheduled at compile-time with bounded memory. Our dataflow approach of SigmaC is based on CSDF model.

Parameterized SDF (PSDF) which was introduced in [31] that is a special case of SDF. This model aims at increasing SDF expressiveness while maintaining its compile time predictability properties. In this model a sub-system (sub-graph) behavior can be controlled by a set of parameters that can be configured dynamically. These parameters can either configure sub-system interface behavior by modifying production/consumption rate on interfaces, or configure behavior by passing parameters (values) to the sub-system actors.

2.3.3 Dataflow sigmaC

To exploit the parallelism available in the platform MPPA, Kalray has developed its own dataflow model: SigmaC. According to this model, the *block* in dataflow graph is called *agent*. Each *agent* is appointed and includes: the C code of the function executed, the number of incoming and outgoing data through the interface together with the set of production/consumption. All these information are assigned through a special syntax, specific SigmaC that illustrated as Figure 2.8.

Every *agent* define that it remains only to create the graph by interacting. To do this, a new syntax is created in order to create graph and sub-graph and thus to organize the work as cleanly as possible. Figure 2.9 illustrates the example of sub-graph syntax connecting two agents.

2.3.4 RVC-CAL language

This part presents the RVC-CAL language and covers the syntax, semantics with this kind of language. RVC-CAL is a Domain-Specific Language (DSL) that has been standardized by RVC as a restricted version of CAL. CAL was invented by Eker and Janneck and is described in their technical report [1].

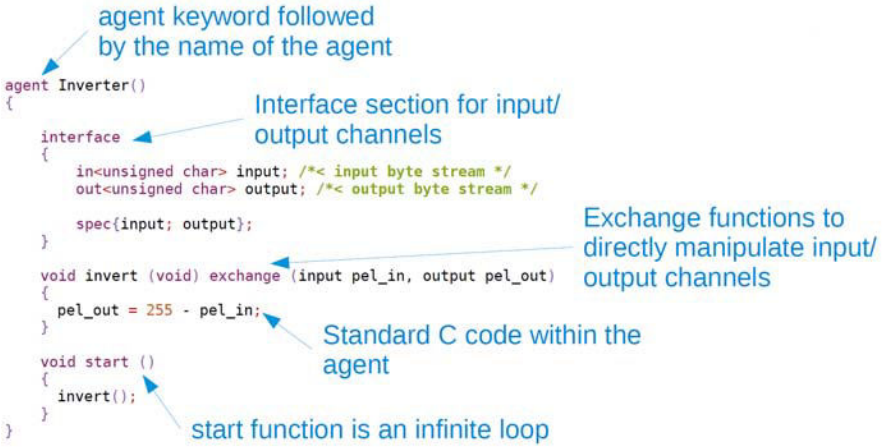


Figure 2.8: SigmaC: syntax of one agent

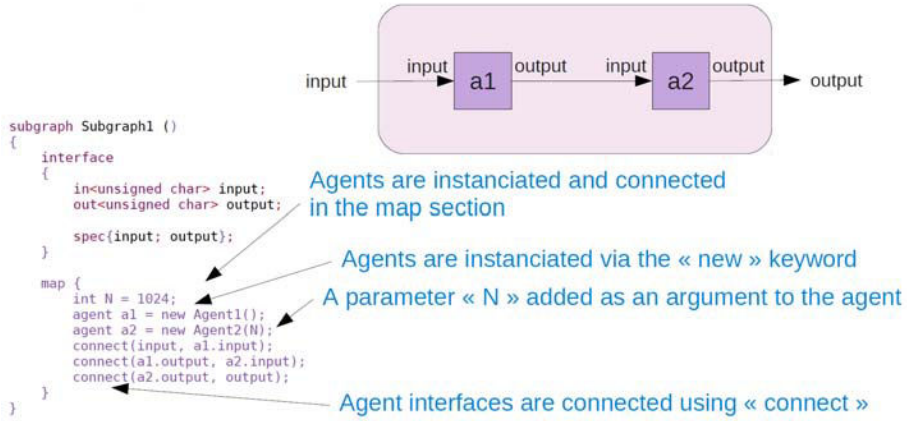


Figure 2.9: SigmaC: syntax of subgraph with two agents a1 et a2

Actor structure

An RVC-CAL actor is an entity that is conceptually separated into an header and a body. The header describes the name, parameters, and port signature of the actor. For instance, the header of the actor shown in the Figure 2.10 defines an actor called *select*. This actor takes two parameters, one boolean and one integer, whose values are specified at runtime, when the actor is instantiated, i.e. when it is initialized by the network that references it. The port signature of *select* is an input port *Cost* and one output ports *Disp*. The body of one actor may be empty, or contains state variables declarations, functions, procedures, actions, priorities, and at most one Finite State Machine (FSM).

Actors perform their computation in a sequence of steps called firings. In each

```
actor select(bool checkHeaderCRC, int accetpedMethods)
   float Cost ==> float Disp:

  //body

end
```

Figure 2.10: The header of an RVC-CAL Actor

of those steps:

1. the actor may consume tokens from its input ports,
2. it may modify its internal state,
3. it may produce tokens at its output ports.

Consequently, describing an actor involves describing its interface to the outside, the ports, the structure of its internal state, as well as the steps it can perform, what these steps do (in terms of token production and consumption, and the update of the actor state), and how to pick the step that the actor will perform next.

State variables

State variables can be used to define constants and to store the state of the actor. Figure 2.11 shows the three different ways of declaring a state variable. The

```
uint(size=16) Number = 0x1fff;

// the bits of   the byte read
uint(size=16) bits;

//   number of bits remaining in value
uint(size=4)  num_bits := 32;
```

Figure 2.11: Declaration of State Variable

first variable Number is a 16-bit unsigned integer constant. The bits variable is a 16-bit unsigned integer variable without an initial value. The num_bits variable is a 4-bit unsigned integer that is initialized to 32. The difference between the = used

to initialize a constant and the := used to initialize a variable. The initial value of a variable is an expression.

Function

As shown in Figure 2.12, a function may declare parameters such as n of bit_-number and local variables, like eof. The body of a function is an expression whose type must match the specified type of the function.

```
function bit_number(int n ) --> bool
var
   bool eof = get_eof_flag();
   if eof then false else num_bits >= n end
end
```

Figure 2.12: Declaration of a Function

Actions

So far, the only firing condition for actions was that there be sufficiently many tokens for them to consume, as specified in their input patterns. However, in many cases we want to specify additional criteria that need to be satisfied for an action to fire conditions, for instance, that depend on the values of the tokens, or the state of the actor, or both. These conditions can be specified using guards, as for example in the Split actor in Figure 2.13: An action may have firing conditions, called guards,

```
actor Split () Input ==> P, N:

  action [a] ==> P: [a]
  guard a >= 0 end

  action [a] ==> N: [a]
  guard a < 0 end
end
```

Figure 2.13: Declaration of a Action

where the action firing depends on the values of input tokens or the current state. Guards are included in scheduling information that defines the criteria for action to fire. The contents of an action, that are not scheduling information, are called its

body, and define what the action does. The difference is not so clear, for instance the expressions in the output pattern are part of the body, but the output pattern itself is scheduling information as it holds the number of tokens produced by the action. When an actor fires, an action has to be selected based on the number and values of tokens available and whether its guards are true. Action selection may be further constrained using a FSM, to select actions according to the current state, and priority inequalities, to impose a partial order among action tags.

Finite state machine (FSM)

An FSM is defined by the triple where S is the set of states, is the initial state, and is the state-transition function: Note that a state transition allows a set of actions to be fireable. Figure 2.14 presents an example of a simple actor that downsamples its input stream by two.

```
actor Downsample() bool R ==> bool R2:
a0 : action R:[r] ==> end
a1 : action R:[r] ==> R2:[r] end

schedule fsm s0:
  s0 (a0) --> s1;
  s1 (a1) --> s0;
end

end
```

Figure 2.14: A simple actor with an FSM

2.4 Tools of Rapid Prototyping Methodology

This section presents some tools integrated in our rapid prototyping methodology such as Orcc, Preesm, and HMPP.

2.4.1 Orcc, an Open RVC-CAL Compiler

Orcc include an RVC-CAL textual editor, a compilation infrastructure, a simulator and a debugger [2]. The primary purpose of Orcc is to provide developers with a compiler infrastructure to allow several languages and combination of languages (in the case of co-design) to be generated from RVC-CAL actors and networks. Orcc does not generate assembly or executable code directly; rather it generates source code that must be compiled by another tool. This tool can generate code for any platform, including hardware (VHDL), software (C/C++, Java, LLVM...), and heterogeneous platforms (mixed hardware/software) from a RVC-CAL description.

How we could obtain the target code for specified architecture from the RVC-CAL description of our applications. This is the procedure of compilation. This section describes basic concepts of compilation, and in particular the concepts that are necessary to understand our work as described in the next chapters. Compilation is the process by which a program in a source language is transformed to another semantically-equivalent program in a target language as shown in Figure 2.15.

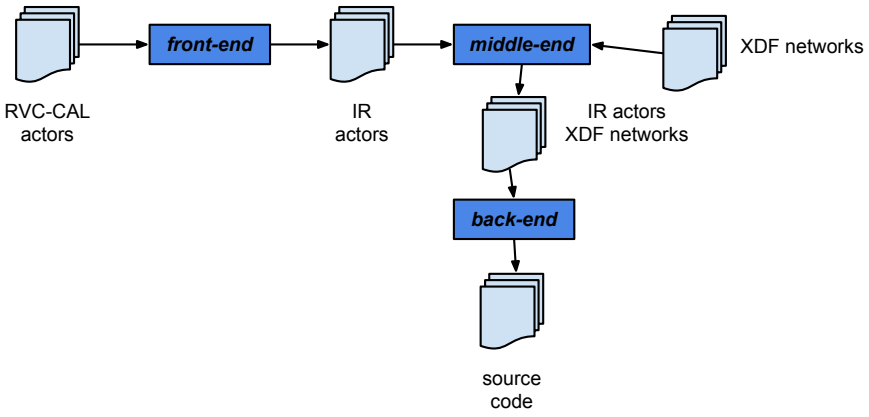


Figure 2.15: Compilation procedure of Orcc

The first stage of the compiler, called front-end, is responsible for creating an Intermediate Representation (IR) of RVC-CAL actors, the resulting actors being called IR actors. The middle-end is the component that analyzes and transforms the IR of actors and networks to produce optimized IR actors and networks. The last stage of the compiler is code generation, in which the back-end for a given language (C, LLVM, VHDL, etc) generates code from a hierarchical network and a set of IR

actors.

2.4.2 Preesm, a Parallel and Real-time Embedded Executives Scheduling Method

The PREESM tool offers a fast prototyping tool for parallel implementations [3]. The inputs of the tool are architecture and algorithm graphs and a scenario that basically gathers all information linking algorithm and architecture. The target architectures are embedded heterogeneous architectures, with possibly several processors and many cores. The current version of PREESM focuses on static mapping/scheduling of algorithms. The Model of Computation (MoC) is used to describe algorithms based on Synchronous Dataflow Graphs (SDF) extended with the hierarchy feature. The PREESM tool is not suitable for adaptable applications (DPN MoC of Orcc) in which the mapping/scheduling should be done partially or completely at runtime.

2.4.3 HMPP, a Hybrid Multicore Parallel Programming model

HMPP developed by CAPS, is a directive-based compiler to build parallel GPU applications, and support for the OpenACC standard [4]. HMPP offers a high level abstraction for hybrid programming that fully leverages the computing power of stream processors with a simple programming model. HMPP compiler integrates powerful data-parallel backends for NVIDIA CUDA and OpenCL that drastically reduce development time. The HMPP runtime ensures application deployment on multi-GPU systems. Software assets are kept independent from both hardware platforms and commercial software. While preserving portability and hardware interoperability, HMPP increases application performance and development productivity.

These are two basic paired directives of HMPP: *Codelet* and *Callsite*. *Codelet* is used before the definition of C functions. *Callsite* is the HMPP directive for invocation on such an accelerator device. Using only simple paired directives, HMPP can replace the complex procedure of manually writing the CUDA/OpenCL kernel code. For the basic HMPP's transformation, we just need to insert two lines of directives into the source C code as shown in Algorithm 1:

Algorithm 1: *HMPP_Transformation()*

```
1 #pragma hmpp motion_estimation codelet, target=CUDA;
2 Definition of function motion_estimation();
3 ...;
4 Main(argc, argv);
5 #pragma hmpp motion_estimation callsite;
6 Function call of motion_estimation();
```

For exploiting the better performance with HMPP, hmppcg directives can optimize the performance of generated HMPP kernel code:

1. The hmppcg gridify directives determine which loop should be parallelized. Because of the $gridify(i, j)$, the i and j for-loops in line 3 and 4 will be parallelized.

```
1 #pragma hmppcg gridify(j,i)
2 for(k = 0; k < N; k++)
3    for(j = 0; j < H; j++)
4       for (i = 0; i < W; i++){
5       ...
6       }
```

2. The hmppcg grid allocate the buffer (data storage region in device) to shared memory. The specified buffer will be loaded into shared memory before complex computations. Because accessing the shared memory is faster than the global memory (more details will be discussed in the 3.1), time efficiency will benefit from the hmppcg grid directive.

```
1 #pragma hmppcg grid shared buffer
```

3. Level 1 cache, is a static memory integrated with processor core that is used to store information recently accessed by a processor. Level 1 cache is often abbreviated as L1 cache. The purpose of L1 cache is to improve data access speed in cases when the processor accesses the same data multiple times. In NVIDIA's GPU devices of compute capability 2.0 and later, there is 64 KB of memory for each multiprocessor. This per-multiprocessor on-chip memory is split and used for both shared memory and L1 cache. By default, 48 KB is used as shared memory and 16 KB as L1 cache. The configuration of L1 cache preference: there is an obvious improvement of performance in our experiments.

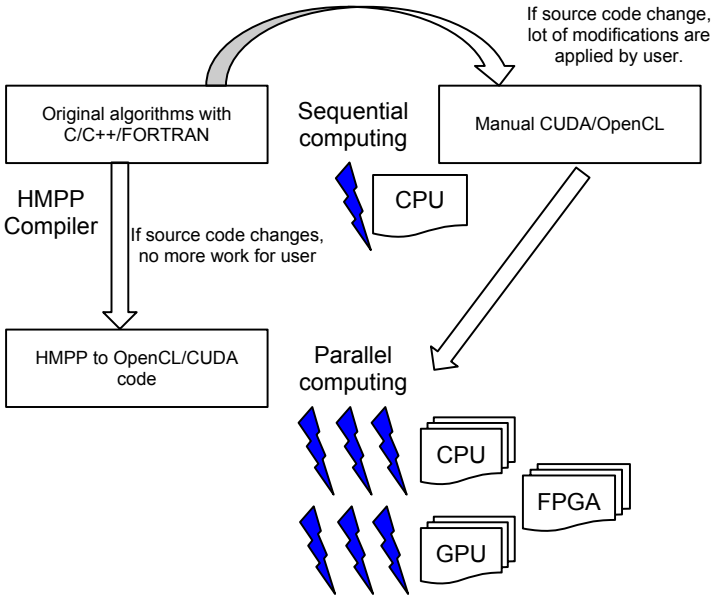


Figure 2.16: HMPP workflow in COMPA project

```
1 cudaDeviceSetCacheConfig(args);
2 args = cudaFuncCachePreferL1;
```

Based on hmppcg directives, we can obtain great performance enhancement comparing with default HMPP. The HMPP backend of Orcc will seek out the part of parallelism in original C code like multi-level *for loops*, and insert the HMPP directives into suitable position of C code. When we compile the C code with HMPP directives, we can obtain the .cu or .cl file for the execution of full search function which targets to GPU. Furthermore, as shown in Figure 2.16, we only need to maintain and modify the C source code, and HMPP will produce the CUDA/OpenCL kernels according to the source C code of applications.

2.5 Concerned Image and Video Processing Algorithms

Video decoders with Orcc and Preesm used to evaluate the prototyping framework in the previous research work. But video decoders are usually based on the dataflow and structure with a lot of data dependencies which are very hard to map

with parallelism and to implement on many core heterogeneous systems. Some parallel applications are proposed in this thesis to evaluate the rapid prototyping framework which can fully make use of the parallelism of target architectures. Thus they are suitable to evaluate the efficiency of the rapid prototyping framework.

2.5.1 Motion estimation

Motion estimation (ME) is the process of determining motion vectors that describe the transformation from one 2D image to another, usually from adjacent frames in a video sequence. It is an ill-posed problem as the motion is in three dimensions but the images are a projection of the 3D scene onto a 2D plane. ME is the main time-consuming task of a video encoder with more than half of the whole computation load. In order to eliminate temporal redundancy, ME could find relative motions between two images as shown in Figure 2.17. The motion vectors

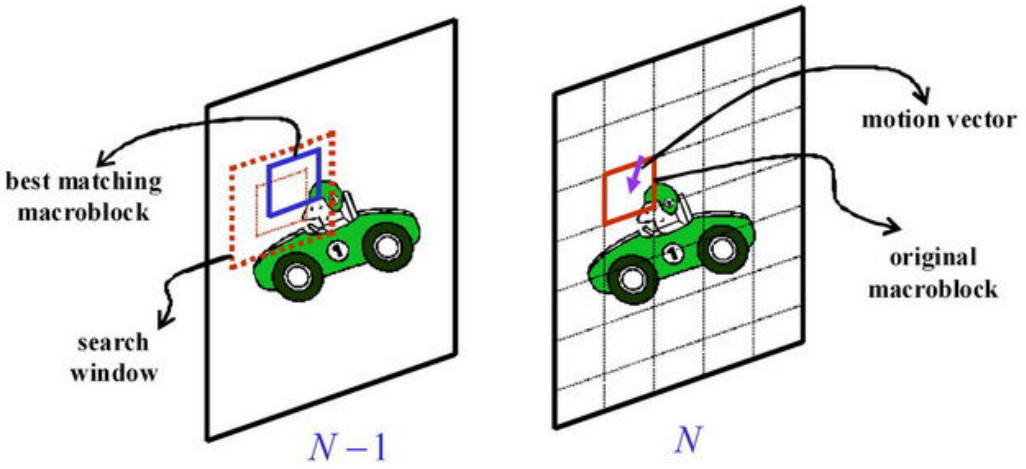


Figure 2.17: Motion estimation illustration

may relate to the whole image (global motion estimation) or specific parts, such as rectangular blocks, arbitrary shaped patches or even per pixel. The motion vectors may be represented by a translational model or many other models that can approximate the motion of a real video camera, such as rotation and translation in all three dimensions and zoom. Closely related to motion estimation is optical flow, where the vectors correspond to the perceived movement of pixels. Applying the motion vectors to an image to synthesize the transformation to the next image

is called motion compensation. The combination of motion estimation and motion compensation is a key part of video compression standard as used by H.264 and HEVC as well as many other video codecs.

The methods of motion estimation can be categorized into pixel based *direct methods* and feature based *indirect methods*.

Direct Methods

– Block-matching algorithm [32]

– Phase correlation and frequency domain algorithms [33]

– Pixel recursive algorithms [34]

– Optical flow algorithms [35]

Indirect Methods employs features, such as corner detection, and match corresponding features between frames, usually with a statistical function applied over a local or global area. The purpose of the statistical function is to remove matches that do not correspond to the actual motion.

Block matching ME algorithms are most widely used in video compression standard, also the most concerned in this thesis. There is a brief overview of available motion estimation systems shown in Table 2.2. The frames per second (fps) indicates the processing speed of the different systems.

Table 2.2: Overview of Motion Estimation implementations

References	Algorithms	Resolution and search region	fps	Platform
mehta *et al.* [36]	threshold in the SAD	640×480, 32×32	52.5	GPU
martin *et al.* [37]	samll diamond search	352×288, 16×16	20	GPU
chen *et al.* [32]	variable block size	352×288, 32×32	31.5	GPU
Luo *et al.* [38]	content adaptive search technique	352×288, 32×32	60.7	CPU
Lee *et al.* [39]	multi-pass full search	352×288, 32×32	8.89	GPU
Li *et al.* [40]	most significant bit	640×480, 32×32	129	FPGA
Urban *et al.* [41]	HDS for Hierarchical Diamond Search	720×576, 32×32	100	DSP

2.5.2 Stereo matching

As shown in Figure 2.18, stereo matching is a technique aimed at computing depth information from two cameras used in many applications such as 3D-TV, 3D reconstruction and 3D object tracking.

In Epipolar Geometry, some essential concepts in stereo matching are shown in

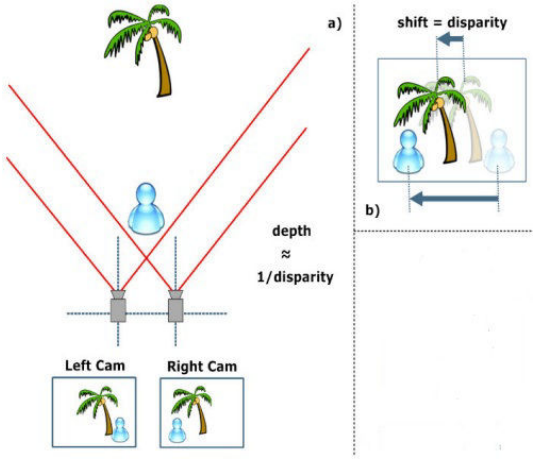


Figure 2.18: Overview of the view stereo matching process, where (a) the stereo vision is captured in a left and right image, (b) the disparities are searched through stereo matching.

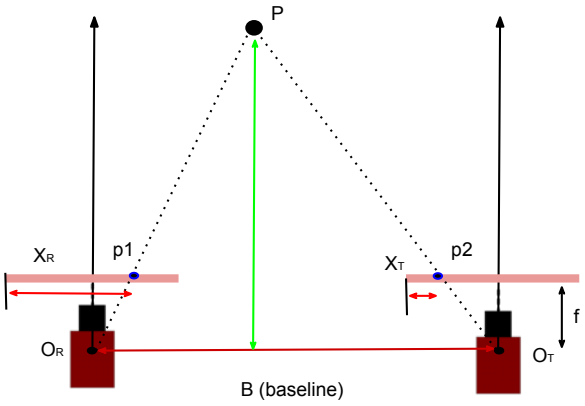


Figure 2.19: Basic concepts of stereo matching in epipolar geometry

Figure 2.19. The object point P corresponds to the pixel $p1$ from the left frame and the pixel $p2$ from the right frame. O_T is the target optical center (left camera) and O_R is the reference optical center (right camera). The distance between the two optical centers is defined as the Baseline represented by B, which is parallel with by epipolar lines, and the length f from frame plane to B is the focal distance. In the case shown in Figure 2.19, the distance from P to B is defined as the depth Z which is the actual distance between cameras and object point, and the distance from $p1$ to $p2$ is defined as the disparity $d = X_T - X_R$), which is the displacement of one stereo pair's locations in the stereo frames. Considering the similar triangles (PO_RO_T and $Pp1p2$), we obtain the relationship between depth Z and disparity d:

$$\frac{B}{Z} = \frac{(B + X_R) - X_T}{Z - f} \Rightarrow Z = \frac{B \cdot f}{X_T - X_R} = \frac{B \cdot f}{d} \tag{2.4}$$

The above equation basically demonstrates that the more deep the object lies, the less disparity it has between the stereo pair in the input videos, and this principle is also the basis of stereo matching. Figure 2.20 elaborates the stereo matching technique with two rectified images from middlebury stereo database [42]. Figure 2.21 lists the histogram of matching cost value, the max bin in this histogram means the best matching result.

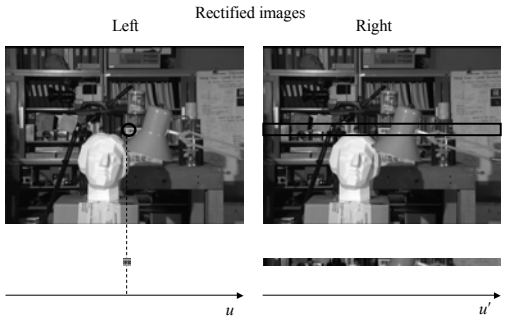


Figure 2.20: Stereo matching scanline

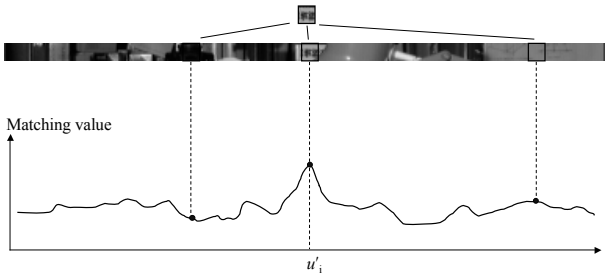


Figure 2.21: Estimates the real disparity

Table 2.3: Overview of stereo vision system

References	**Algorithms**	**Mde/s**	**fps**	**Platform**
Yang *et al.* [44]	constant space belief propagation	14.4	20	CPU
Xiang *et al.* [45]	hierarchical belief propagation	28.8	16.3	CPU
Hosni *et al.* [46]	guided filter	199.6	25	GPU
Zhang *et al.* [47]	Bitwise fast voting	100.9	57	GPU
Yang *et al.* [48]	SAD	283.2	11.5	GPU
Miyajima *et al.* [49]	SAD	491.5	20	FPGA
Chang *et al.* [50]	SAD	88.4	50	DSP

There are two main groups of stereo matching algorithms: feature-based and area-based algorithms [43]. The first category aims to employ suitable features, such as corners or edges of the images and match them afterward. The second category tries to match each pixel independently. Feature-based methods create sparse disparity map because they only obtain disparities for these extracted features. Area-based methods calculate the disparities for each pixel of image, resulting in a dense disparity map. This section introduces an overview of the basic stereo matching technique that contains raw matching cost, stereo vision applications and so on. The stereo techniques discussed in this thesis are restricted to area-based method. There is also a brief overview of available stereo vision systems shown in the Table 2.3. Mde/s is more meaningfully in million disparity evaluations per second ($Mde/s = width \times height \times disps \times fps$).

2.6 Conclusion

In this chapter, the background has been presented. It also gives the reader the necessary knowledge on the theoretical and practical aspects of parallel computing, rapid prototyping methodology and image of video processing algorithms (motion estimation and stereo matching). In section 2.1, we presented and compared different target architectures CPU, GPU, and MPPA Kalray. We have presented an intermediate level programming model: OpenCL in section 2.2. OpenCL programming model and development environments are described in this section. In section 2.3, we presented an high level programming model: the dataflow approach. This section introduce the different MoC, dataflow sigmaC for the platform MPPA Kalray, and the high level description language RVC-CAL used in our descriptions of application in the following sections. Some tools of rapid prototyping methodology is presented in section 2.4. Finally, we introduced the basic concept of motion estimation and stereo matching algorithms whose applications are widely used in industry and science in section 2.5. The next chapter will introduce the parallelized motion estimation based on heterogeneous computing system.

3

Parallelized Motion Estimation Based on Heterogeneous System

3.1 Introduction

Due to the growing demand of video quality and compression rate, the efficiency and complexity of video coding standards are facing with great challenges. Motion Estimation (ME) is always the main time-consuming task of a video encoder with more than half of the whole computation load. ME aims to find relative motions between two images in order to eliminate temporal redundancy. Block matching ME algorithms are most widely used in video compression. An example is the full search where each candidate within a search window of p pixels is considered. The consumed time depend on the size of search range. Because it tries the matching probability as much as possible, the full search ME is the most time consumed block matching ME method. But it can obtain the best result compared with other methods. In all, the full search ME is very hard to implement in real-time. So some fast algorithms [51] [52] [53] [54], [55] have been proposed to accelerate the ME. The another improvement is to accelerate full search ME based on hardware accelerators [56] [39] [57] [40] [58].

Fast ME algorithms

Tourapis *et al.* [51] proposed the Predictive Motion Vector Field Adaptive Search Technique (PMVFAST), which significantly outperforms most previously proposed

algorithms in terms of speed up performance. Their algorithm relies upon robust and reliable predictive techniques and early termination criterion, which make use of parameters adapted to the local characteristics of a frame. Luo *et al.* [38] proposed a multistage motion estimation algorithm that includes a pre-stage to analyze the motion characteristics of video sequence. This stage predicts the motion vector field (MVF) from the previous coded frame, and clusters macroblocks into background and foreground regions based on the predicted MVF. The information from the pre-stage are passed on to the next stage, which includes a mathematical model for block distortion surface to estimate the distance from the current search point to the global optimal position.

Cheung *et al.* [55] proposed two cross-diamond-hexagonal search (CDHS) algorithms, which differ from each other by their sizes of hexagonal search patterns. To further reduce the checking points, two pairs of hexagonal search patterns are proposed in conjunction with candidates found located at diamond corners. Experimental results show that their method perform faster than the diamond search (DS) by about 144% and the cross-diamond search (CDS) by about 73%, whereas similar prediction quality is still maintained. Moradi *et al.* [54] proposes two enhanced CDHS algorithms to solve the motion-estimation problem in video coding. These algorithms differ from each other by their second step search only, and both of them employ cross-shaped pattern in first step. Their method is an improvement over CDHS, which eliminates some checking points of CDHS algorithm. Experimental results show that their methods perform faster than the diamond search (DS) and CDHS, whereas similar quality is preserved.

Yao *et al.* [53] proposed a novel and simple fast block-matching algorithm, called adaptive rood pattern search (ARPS), which consists of two sequential search stages: initial search and refined local search. For each macroblock (MB), the initial search is performed only once at the beginning in order to find a good starting point for the follow-up refined local search. For the initial search stage, an adaptive rood pattern (ARP) is proposed, and the ARP's size is dynamically determined for each MB, based on the available motion vectors (MVs) of the neighboring MBs. In the refined local search stage, a unit-size rood pattern (URP) is exploited repeatedly,

and unrestrictedly, until the final MV is found. In these fast ME algorithms, the number of evaluated candidates is decreased, but the reduction comes with a loss in terms of quality which means lower Peak Signal to Noise Ratio (PSNR) of the compressed video. In the other hand, many researchers proposed their solutions to accelerate the ME with hardware accelerators like GPU, FPGA and DSP.

Hardware-based FSME

Motion estimation becomes a very time consuming task even for today's CPU. On the other hand, modern graphics hardware includes a powerful GPU whose computing power remains idle most of the time. Urban *et al.* [41] proposed one real-time ME for H.264 high definition video encoding on multi-core DSP, which is well-suited for embedded parallel systems. Li *et al.* [40] proposed a novel most significant bit (MSB) first bit-serial architecture for full-search block matching ME based on FPGA platform. Since the nature of MSB-first processing enables early termination of the sum of absolute difference (SAD) calculation, the average hardware performance can be enhanced.

Some first results about GPU-based ME have been proposed in [32, 39, 57, 59–61] with CUDA approach. Chen *et al.* [32], Lee *et al.* [39] and Lin *et al.* [59] proposed multi-pass GPU-based ME algorithms. They store each matching blocks into shared memory, and they ignore pixels of search range in reference frame. Cheng *et al.* [57] implemented the full search algorithm, the diamond search algorithm, and the four step algorithms in both the CPU and the CUDA platforms. All the computation about SAD is executed in the global memory. Experiment results show that the CUDA-based implementations of these ME algorithms can be more than 8 times faster than the CPU-based ME implementations. Schwalb *et al.* and Ho *et al.* implement the motion estimation for H.264 using programmable GPU. To overcome the dependency problem, they introduce a new implementation which performs ME on block-by-block basis. And their implementation is 10 times faster than SIMD optimized CPU implementation. Ko *et al.* [56] proposed a novel motion estimation algorithm for GPU implementation which is accompanied by a novel pipelining technique, called subframe ME processing, to effectively hide the communication

overhead between the host CPU and the GPU. Lee *et al.* [39] proposed one multi-pass and frame parallel algorithms to accelerate various motion estimation methods with the GPU. By the multi-pass method to unroll and rearrange the multiple nested loops, the ME can be implemented with two-pass process on GPU. Moreover, fractional ME needs six passes for frame interpolation with six-tap filter and motion vector refinement. Motion estimation with multiple reference frames can be implemented with two-pass process with framelevel parallel scheme by use of SIMD vector operations of GPU. In this context, the parallel structure of the full search ME is easy to accelerate the computation without decreasing the PSNR. We propose our parallelized motion estimation method which divides the whole workload of full search ME into two kernels respectively.

In order to obtain better performance with hardware accelerators like GPU, we port our source C code to the kernel code through our rapid prototyping methodology. In general, there are two ways to perform the code porting. One is to manually write the kernel of CUDA or OpenCL which redesign the code and compile the kernel code with the specified compiler. The other way utilize rapid prototyping framework like the one we proposed in the last chapter to automatically generate code executable on hardware accelerators from the high-level description of applications.

1. First we generate the HMPP code (C code with HMPP directives) from the high-level description of RVC-CAL.
2. Second we generate the OpenCL and CUDA kernel through HMPP compiler.

In this chapter, we elaborate the two ways of porting code to GPUs and describe how our rapid prototyping framework works.

This chapter is organized as follows: Section 3.2 illustrates our approach of motion estimation with OpenCL. Section 3.3 presents the optimization strategies of our ME implementation and the experimental results. Section 3.4 applies our proposed algorithm on the heterogeneous computing system with OpenCL and presents the method to distribute workload in the heterogeneous system. Section 3.5 introduces the high level CAL description of ME and how our rapid prototyping framework works, also gives the analysis of experiments result. A brief conclusion is given in Section 3.6.

3.2 Parallelized Motion Estimation Algorithm

Two aspects should be considered for distributing workloads to GPUs or CPUs, namely *massively parallelized motion search* and *calculation of the matching criterion* in motion estimation algorithms. *Full Search* ME process is to search the best candidate of macroblock (MB) in reference frame (the frame before current frame in video sequence, N-1 frame in Figure 2.17) for the original macroblock in current frame (N frame in Figure 2.17).

There is one question: how we determine the best candidate macroblock in reference frame? Here, we chose SAD as the matching criterion for selecting the best Motion Vectors (MV) in (3.1). M is the specified block size.

$$SAD = \sum_{i=1}^{M}\sum_{j=1}^{M} |block\_current(i,j) - block\_ref(i,j)| \tag{3.1}$$

The proposed *Full search* ME approach is divided into two OpenCL kernels. One is SADs computation; the other is SADs comparison for the best SAD candidate (final MV). We define the two OpenCL kernels as *kernel_compute* (see in Algorithm 2) and *kernel_compare* (see in Algorithm 3).

Algorithm 2: *kernel_compute()*

```
input  : Current and reference frames
output: SAD costs candidates with offset in the x and y axis
Initialize the local memory space for macroblocks and search window;
for n = 0; n <number of macroblocks/number of CUs; n++ do
    for m = 0; m < 4; m++ do
        256 work-items in one work-group calculate 256 SAD candidates
        simultaneously;
    end
end
return (SAD, MV)
```

In the host, we pad the reference frame when we load video frames. The definition of frame is $width \times height$, and the search range is defined as $[-w, w]$. The padded image size should be $(width + 2 \times M) \times (height + 2 \times M)$ as shown in Figure 3.1.

Algorithm 3: *kernel_compare()*

input : SAD costs candidates with offset in the x and y axis
output: motion vector with the minimum SAD cost

1 Initialize the local memory for 1024 costs candidates of each blocks;
2 Each work-items compares 4 SAD candidates and return the minimum;
3 **for** $n = 0; n < \log_2 256; n++$ **do**
4 | Parallel reduction for the minimum SAD, half number of work-items compare the adjacent data and return the minimum data to next iteration;
5 **end**
6 **return** MV with the minimum SAD

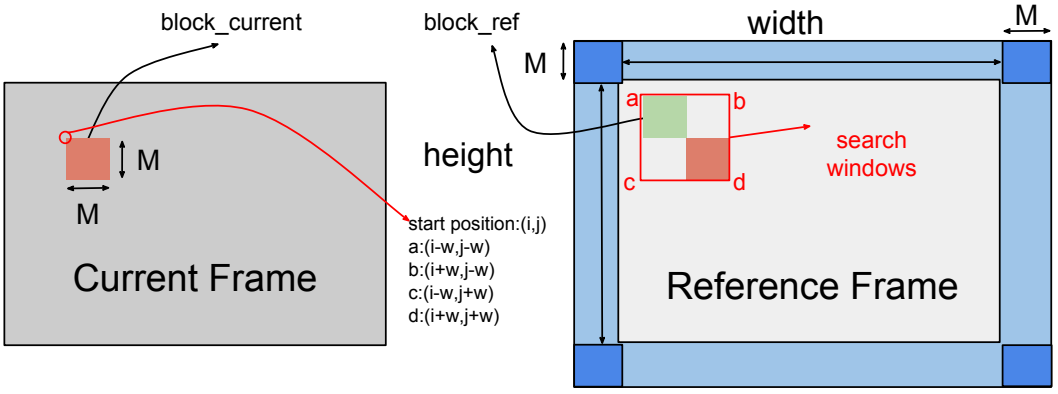


Figure 3.1: Padding image illustration

Padding reference frame guarantees no memory accessing violation when the search window moves on boundary of frame and avoid performance decreasing caused by if-else statement in OpenCL kernels.

We define the first pixel (i,j) of the block_current as the start point. Because of the search range $[-w, w)$, the search windows is represented by 4 vertex a,b,c and d as shown in Figure 3.1. The block_ref moves in the search windows with the step 1 pixel in two directions (different position (x,y)) from upper left corner a(i-w,j-w) to bottom right corner d(i+w,j+w), and calculates the SAD in each position with equation (3.1). The candidate of block_ref will be chosen when it possesses the minimum SAD value. The position (x,y) of the best candidate of block_ref relative to block_current is the motion vector.

Because the execution time for *Full Search* ME method is proportional with the size of search region and blocks, most previous research work employ the [-16,16) (32×32 pixels) search region considering a 16×16 pixels block as shown

in Table 2.2: Overview of ME implementations. The proposed *Full search* ME approach is divided into two OpenCL kernels. One is SADs computation; the other is SADs comparison for the best SAD candidate (final MV). We define the two OpenCL kernels as *kernel_compute* (see in Algorithm 2) and *kernel_compare* (see in Algorithm 3).

3.2.1 SAD computation

Based on block matching method, each frame is divided into macroblocks (MBs, $MB\_size = MB\_width \times MB\_height = 16 \times 16 = 256$) which have a search region in the reference frame. We suppose the $Search\_Range = 32$. So every MB has $32 \times 32 = 1024$ candidates which concern $(48) \times (48) = 2304$ pixels in the reference frames as shown in Figure 3.2.

In our design, we set the $localsize = MB\_size$ and there are 256 work-items executed in one work-group ($work\_group\_size = 256$). Because each work-item processes four candidates SAD, 1024 candidates SAD of one MB are distributed to one work-group perfectly. For every frame, the total number of work-groups is N:

$$N = \frac{width}{MB\_width} \times \frac{height}{MB\_height} \times \frac{Search\_Range^2}{work\_group\_size} \tag{3.2}$$

When an OpenCL program invokes a kernel, N work-groups are enumerated and distributed as thread blocks to the multiprocessors with available Compute Units of CPU or GPU. In *kernel_compute*, all pixels of MB are transferred into local memory ($local[256]$) by the 256 work-items in one work-group. Until all these work-items in the same work-group reach the synchronous point using *barrier()* function, all the 256 work-items continue transferring the 1024 candidates (2304 pixels of search region concerned) of reference image into local memory ($local\_ref[2304]$). This differentiates our approach from approaches of [32] and [39] which accelerate the full search motion estimation in H.264 with CUDA. In their work, current MB is stored in local memory, but the search region of the reference frame is still in the global memory which results in inevitable re-fetching from global memory.

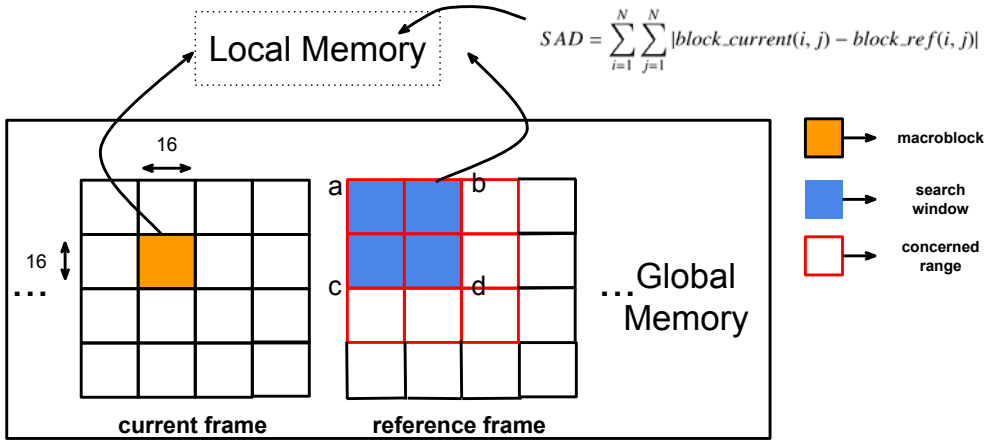


Figure 3.2: Proposed SAD computation in local memory

At the end, we adopt full search strategies to calculate the 1024 candidate SADs in local memory without re-fetching from the global memory. All the 1024 candidate SADs are stored back to global memory ($cost[1024]$).

3.2.2 SAD comparison

In *kernel_compare*, we search the best candidate with the minimum SAD from $cost[1024]$ using 256 work-items as presented in Algorithm 3. First, the $cost[1024]$ is transferred into local memory. Then, each work-item compares 4 candidates with stride to find the minimum as equation (3.3).

$$\begin{aligned} cost[i] = Min(&Min(cost[i], cost[i + stride]), \\ &Min(cost[i + 2 \times stride], cost[i + 3 \times stride])) \end{aligned} \tag{3.3}$$

where i is the index of work-items, $i \subseteq [0, 255]$, stride is equal to the number of work-times ($stride = 256$). At last, there are 256 candidates remained from $cost[0]$ to $cost[255]$. We employ the parallel reduction method [62] which adopts x times iterations ($2^x = 256, x = 8$) to find the candidate with the minimum SAD value from the remaining 256 candidates, also to obtain the final MV. Parallel reduction is usually used to sum up the data in large dataset. Here we use it as one sorting approach. Figure 3.3 illustrates one simple example to search the minimum form

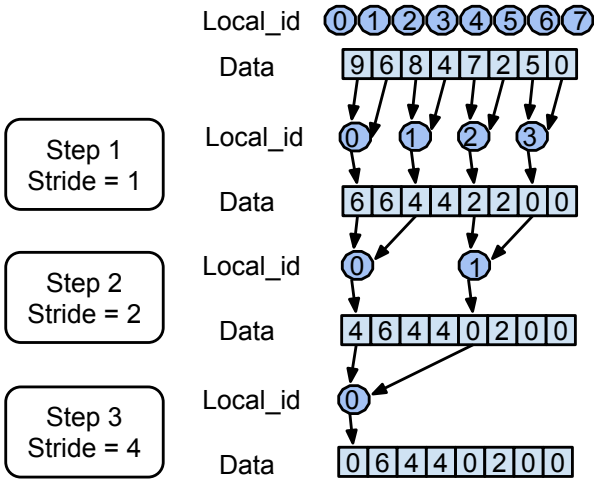


Figure 3.3: SADs comparison in parallel reduction

the 8 numbers ($2^x = 8, x = 3$). It needs 3 times iterations to find the minimum. In the first iteration, half numbers of threads (index from 0 to 3) compare the adjacent data and return the minimum data to next iteration. The next iteration will be the same; half numbers of threads compare and return data. At the end, the minimum SAD is selected.

3.3 Optimization Strategies of Parallel Computing

3.3.1 Using shared memory

Accessing shared memory it is where OpenCL local memory reside) is as fast as accessing a register, and the shared memory latency is 100x lower than the device memory latency. To achieve high memory bandwidth for concurrent accesses, the shared memory is divided into equally sized memory banks that can be accessed simultaneously. However, if multiple addresses of a memory request mapping to the same memory bank, these accesses are serialized and caused banks conflict. To gain maximum performance, it is therefore important to understand how memory addresses map to memory banks in order to schedule the memory requests so as to minimize the bank conflicts. The most efficient way to avoid bank conflict is the

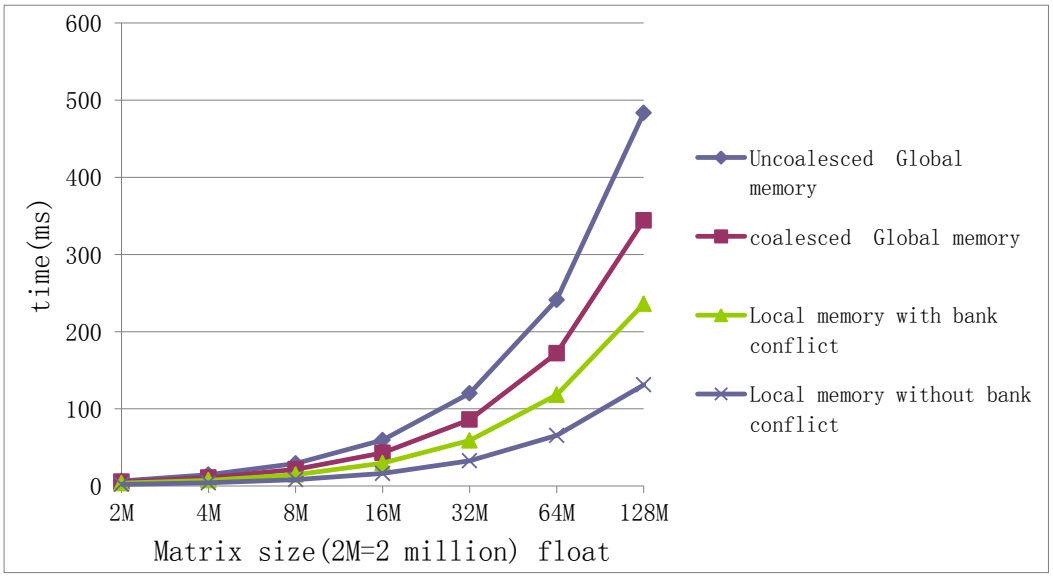


Figure 3.4: Performance comparison between global and local memory in Matrix and Vector Multiplication application

alignment between the memory bank and thread.

As shown in Figure 3.4, it can be observes that the performance comparison between global and local memories with Matrix Vector Multiplication applications. Coalesced global memory access is that all data is fetched in one memory transaction. Uncoalesced global memory access is that data is fetched one-by-one, always 16 memory transactions. Comparing the computation in global memory, we should load as much data as possible into the shared memory ahead of complex and repeated computations. The data density is the ratio between time consuming of computations and data transferring. The more data density we use, the better speed-up ratio we can obtain with the local memory.

3.3.2 Using vector data

There are special vector instruction sets for vector data of Intel's CPU and NVIDIA's GPU, named vectorization. Each work-item can process several elements which benefit from vectorization of kernel code. Algorithm 4 and Algorithm 5 present the pseudo code with vectorization and without vectorization respectively. In our implementation as shown in Algorithm 5, ***uchar16***(one data type defined in OpenCL standard with the combination of 16 uchar data) is employed instead of

Algorithm 4: *SAD computing without vectorization()*

```
1 for m = 0; m < 16; m++ do
2 |   for n = 0; n < 16; n++ do
3 |   |   cost+ = abslute_difference(current_mb[m + n], ref_mb[m + n]);
4 |   end
5 end
```

Algorithm 5: *SAD computing with vectorization()*

```
1 uchar16 cur_16=(cur_mb[m],cur_mb[1+m],...,cur_mb[15+m]);
2 uchar16 ref_16=(ref_mb[m],ref_mb[1+m],...,ref_mb[15+m]);
3 for m = 0; m < 16; m++ do
4 |   cost+ = abslute_difference(cur_16.s0, ref_16.s0);
5 |   cost+ = abslute_difference(cur_16.s1, ref_16.s1);
6 |   ...;
7 |   cost+ = abslute_difference(cur_16.se, ref_16.se);
8 |   cost+ = abslute_difference(cur_16.sf, ref_16.sf);
9 end
```

uchar to unroll the ***inner for loop*** in the procedure of SAD computing.

3.3.3 Compute Unit Occupancy

Compute Unit Occupancy is the ratio of the number of active warps to the maximum number of warps supported on a Compute Unit of the GPU. With OpenCL, programmers should consider the practical infrastructure of different platforms. Although OpenCL has a unified programming model, it cannot ensure that the same OpenCL kernel has the same performance on different platforms. Therefore programmers should organize as much as possible work-items into work-groups to fully saturate Compute Units of device.

3.3.4 Experimental Result

We first briefly introduce the different GPU architectures used in this chapter. From the Table 3.1, we can find out the different features of selected GPUs which are factors to impact the performance of general purpose computing, and also are foundations that we choose to normalize the performance of different GPUs.

For instance, our GPU NVIDIA GT540m has 2 Compute Units (CUs), and

Table 3.1: Summary of different NVIDIA GPUs in our comparison

Devices	**GeForce GT540m**	**GeForce 7800GT**	**GeForce 8800GTX**
Architecture	Fermi	G71	G80
Compute Unit (CU)	2	6	16
Processing Element (PE)	96	48	128
Core Speed/Mhz	672	440	575
Memory Speed/Mhz	900	900	900
Memory Bus Width/bits	128	256	384

each CU has 48 Process Elements (PEs), resulting in a total of 96 processor cores (PE). NVIDIA 8800GTX has 16 Compute Units (CUs), and each CU has 8 Process Elements (PEs), resulting in a total of 128 processor cores. We evaluate the performance of our proposed ME algorithm with manual OpenCL kernels on such hardware HASEE environment: Intel I7 2630qm (2.8GHz), NVIDIA GT540m. We compare the performance of GPU-based FSME implementation and selected state-of-the-art fast ME algorithms. To the best of our knowledge, there are several criterions of evaluation for final motion vectors of our implementations: *time efficiency*, *matching accuracy*. So our experimental results mainly focus on two aspects: time consumed and PSNR.

Time Efficiency

First, we execute the sequential C implementation of full search ME with single CPU core. Then, we run the proposed ME algorithm on the OpenCL CPU platform and two different GPUs platforms. We chose three test sequences with different resolutions, Foreman (CIF, 352×288), City (4CIF, 704×576), Mobal_ter (720p, 1280×720) and the same 32x32 search range. As shown in Figure 3.5 from CIF to 720P, our proposed method with OpenCL CPU achieves (107,19,7.6) fps, with OpenCL GT540m achieves (355,88,38) fps compared with C code that only produces (7.5,1.7,0.6) fps. It means that our proposed method achieves 45x to 65x speed-up for different resolutions comparing with implementation with single CPU core. The CPU I7 2630qm with 4 physical cores and 8 threads achieves more than 10 times speed-up due to the utilization of vectorization and the unroll procedure.

We also describe the time consuming of selected GPU-based full search ME methods as shown in Table 4.2. Our method significantly outperforms other GPU-

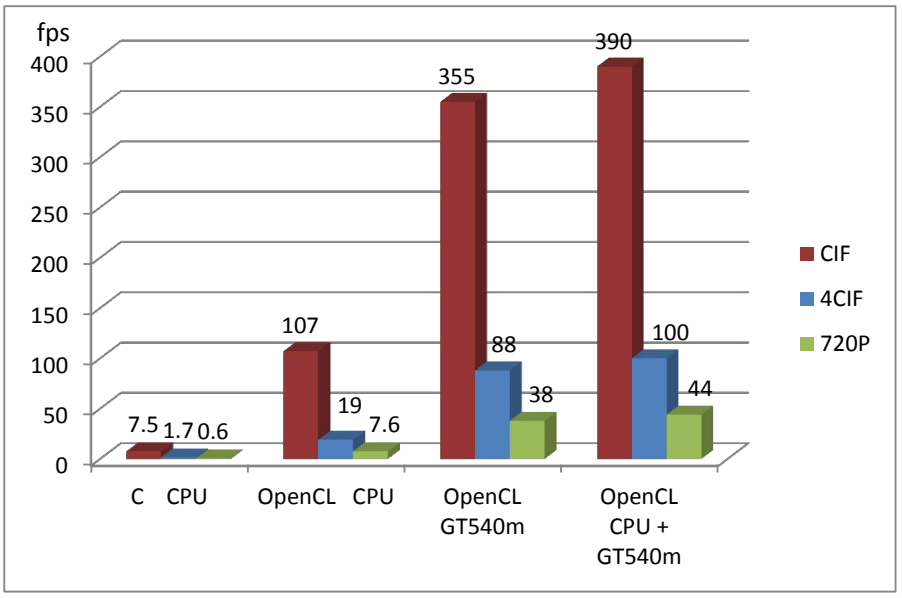


Figure 3.5: Performance comparison with heterogeneous parallel computing

based methods in time efficiency, even though we employ the low-end laptop's GPU platform: NVIDIA GT540m. As shown in Figure 3.6, our method has the best performance comparing with other's implementations due to the fully utilization of shared memory and vectorization as discussed in Section 3.3. Figure 3.7 shows the proposed method performance under different resolutions.

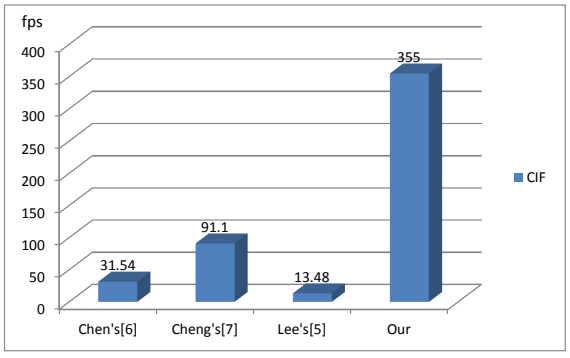


Figure 3.6: Time consuming comparison of selected GPU-based ME methods.

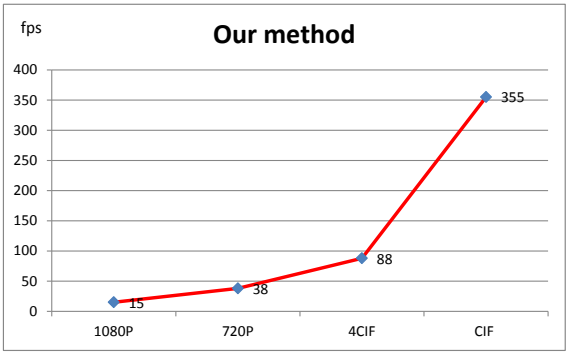


Figure 3.7: Proposed method performance under different resolutions

Matching Accuracy

Secondly, we compare the time consuming and PSNR with selected state-of-the-art fast ME algorithms: Four-step search (4SS) [52], Adaptive Rood Pattern Search

Table 3.2: Integrated Analysis for selected methods of GPU-based ME, unit:fps

Method	Foreman (CIF)	City (4CIF)	Mobcal_ter (720P)	Device	Rank
Chen's [32]	31.54	9.19	4.10	NVIDIA Geforce 8800 GTX	3
Cheng's [57]	91.10	24.7	10.6	NVIDIA Geforce 8800 GTX	2
Lee's [39]	13.48	3.50	1.17	NVIDIA Geforce 7800 GT	4
Our method	**355**	**88**	**38**	**NVIDIA GT540M**	1

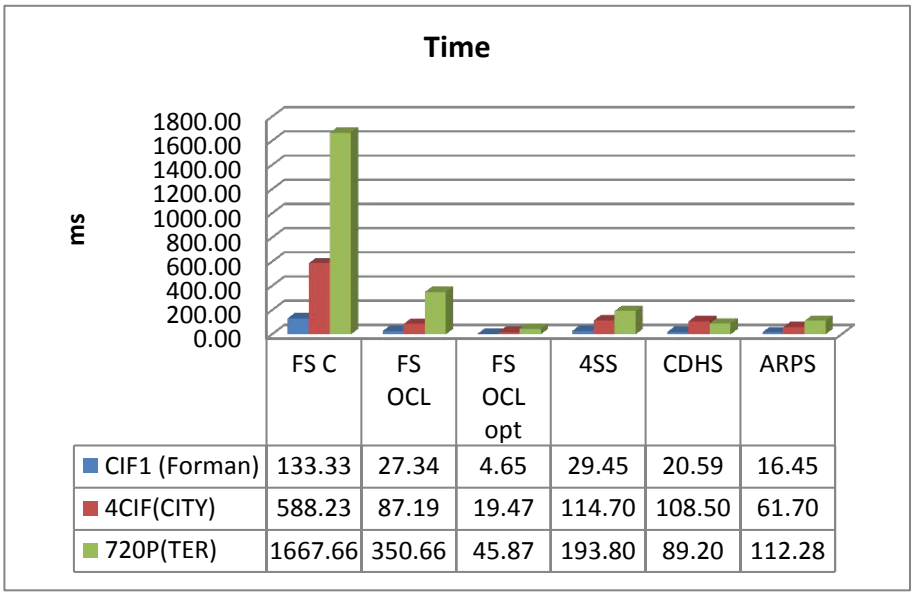


Figure 3.8: Time comparison of specified state-of-the-art fast ME algorithms

(ARPS) [53], Cross-diamond-hexagonal search (CDHS) [54, 55] .

As shown in Figure 3.8 and Figure 3.9, our full search OpenCL opt (opt:with shared memory) implementation outperforms others fast ME methods in time consuming and PSNR with three different video resolution.

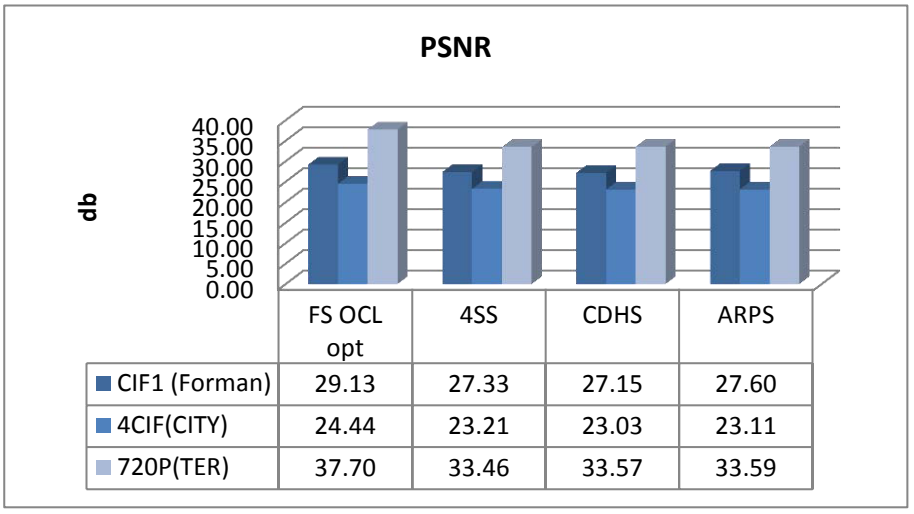
The PSNR is defined as:

$$psnr = 10 * log10(w * h * 255 * 255/sse) \tag{3.4}$$

where $w = width/blocksize, h = height/blocksize,$

$$sse = \sum_{i=0}^{width} \sum_{j=0}^{height} (current_{frame}(i,j) - ref_{frame}(i+dx, j+dy))^2 \tag{3.5}$$

where (dx, dy) is the calculated motion vector. Our time consuming and PSNR are the average value based on total 300 frames of test video.



	FS OCL opt	4SS	CDHS	ARPS
CIF1 (Forman)	29.13	27.33	27.15	27.60
4CIF(CITY)	24.44	23.21	23.03	23.11
720P(TER)	37.70	33.46	33.57	33.59

Figure 3.9: PSNR comparison of specified state-of-the-art fast ME algorithms

3.4 Heterogeneous Parallel Computing with OpenCL

In this section, we employ the special feature of OpenCL - Heterogeneous Parallel Computing [63] to dig for the better performance of our proposed algorithm. First we build one heterogeneous parallel computing environment with one CPU (I7 2630qm) and one GPU (GT540m) as coprocessor for our arithmetic data-parallel computing with OpenCL. Then we distribute the workload to CPU and GPU respectively. In such a heterogeneous computing system, we should determine the usage model of multi-devices and workload distribution. In order to minimize the cost of memory transferring between different devices, we prefer to use ***multi-input*** and ***multi-output*** for multi devices (assigning workload for different devices respectively). Under the perfect condition, all devices complete their workload at the same time, and the total time should be the minimum.

3.4.1 Cooperative multi-devices usage model

For heterogeneous computing environment with different devices, we build one cooperative multi-devices model. In OpenCL, there are two kinds of models for multi-devices:

1. Separate contexts for different devices.
2. One combined context for multi-devices.

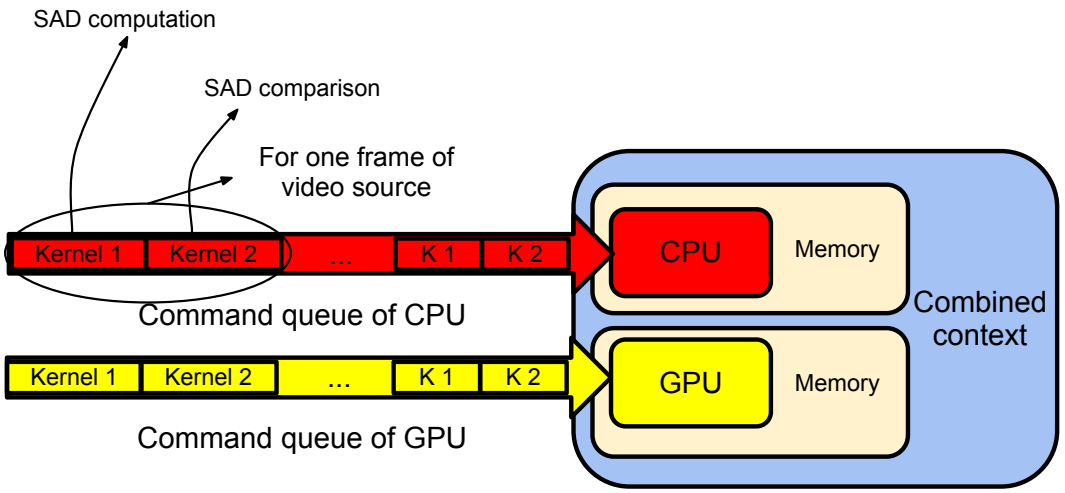


Figure 3.10: Combined context for our heterogeneous computing system

Algorithm 6: *context_building()*

```
1 Initialize the spaces for devices;
2 Get the device ID for CPU;
3 Get the device ID for GPU;
4 Create the combined context with CPU and GPU both;
5 Create and initialize the command_queues for CPU and GPU respectively;
```

Because it is difficult to share one memory object and synchronize command queues in different contexts, we prefer to build one combined context for CPU and GPU as shown in Figure 3.10. In this combined context, CPU and GPU possess their own global memory respectively and don't need transfer data between CPU and GPU which causes huge time delay. The example code of building the combined context is shown in the Algorithm 6.

3.4.2 Workload distribution

With the fixed problem size, how to distribute the workload to different devices is the key point to gain the enhancement of performance in heterogeneous computing systems. In the video applications, the basic problem unit is the frame or image.

We make one supposition that one test video sequence has N frames. We transfer M frames to CPU device and $(N - M)$ frames to GPU device to execute our proposed ME algorithm (*kernel_compute* and *kernel_compare*). The suitable number of M

does guarantee the best performance. As discussed in Section 3.2, the time of transferring the needed frames is $T_{memtrans}$; The time of kernel computing is T_{kernel}; the time of copying buffer for kernel executed is $T_{memcopy}$. The total nonlinear cost time is

$$T_{device} = T_{memtrans} + T_{kernel} + T_{memcopy}. \tag{3.6}$$

$T_{cpu}(M)$ is the time of CPU processes M frames, and $T_{gpu}(N - M)$ is the time of GPU processes *(N - M)* frames. Under the perfect situation, the CPU and GPU will finish their work separately at the same time. When $T_{cpu}(M) = T_{gpu}(N - M)$, we can obtain the best performance of heterogeneous computing system.

3.4.3 Find the balance of performance

As some results we have presented in [64], we should repeat enough experiments to generate the experimental curves of performance as shown in Figure 3.11 and Figure 3.12. So we can get the equations of $T_{cpu}(M)$ and $T_{gpu}(N - M)$ from our experimental results with the technology of curve fitting [65–67] as shown in equation (3.7) and (3.8).

$$T_{cpu}(x) = 29x^2 - 52x + 6 \tag{3.7}$$

$$T_{gpu}(x) = 8x^2 - 11x - 2.7 \tag{3.8}$$

Then we calculate the suitable M value according to our hypothesis:

$$\begin{aligned} T_{cpu}(M) &= T_{gpu}(N - M) \\ 29M^2 - 52M + 6 &= 8(N - M)^2 - 11(N - M) - 2.7 \end{aligned} \tag{3.9}$$

As our test video sequence has 300 frames ($N = 300$), the final suitable M is calculated as 105 according to equation (3.9). It means that $M = 105$ is the best tradeoff of proposed ME method based on our heterogeneous computing system. The practical experiment with CPU and GPU has verified the feasibility of our hypothesis. When the number of frames for CPU is around 100, the heterogeneous system consumes the least time as shown in Figure 3.13.

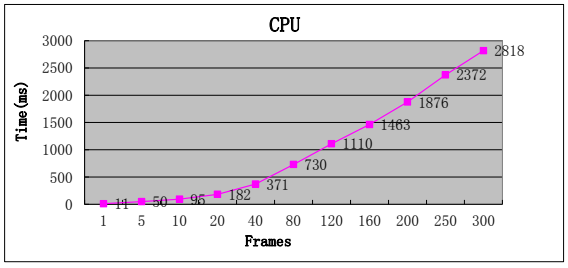


Figure 3.11: Experimental curves of CPU

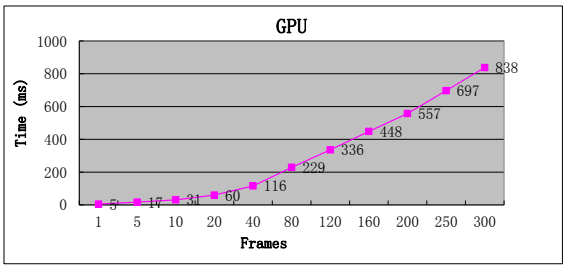


Figure 3.12: Experimental curves of GPU

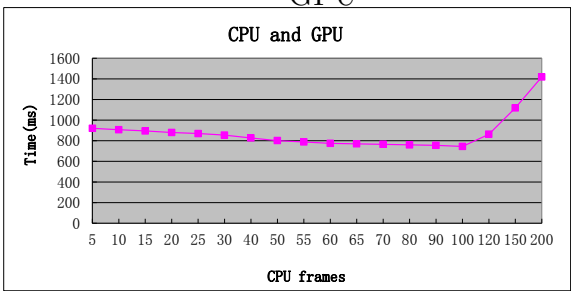


Figure 3.13: Experimental results of GPU and CPU both

3.5 Rapid Prototyping for Parallel Computing

As we have mentioned in Section 3.1, besides of manually writing the kernel code of OpenCL, we can also use the high level compiler which targets to parallel computing. Based on our previous research on motion estimation algorithm, firstly we describe our motion estimation method with RVC-CAL language. Obviously there are two branches in our rapid prototyping framework, one branch is $CAL \rightarrow Orcc \rightarrow HMPP \rightarrow GPU(OpenCL/CUDA)$, the another one is $CAL \rightarrow Orcc \rightarrow Preesm \rightarrow DSP/ARM(EmbeddedC)$ as shown in Figure 1.2 of rapid prototyping framework. In this paper, we choose the first branch to generate and verify C/OpenCL/CUDA code on heterogeneous platforms like multi-core CPU and GPU platforms respectively. Using the HMPP-backend of Orcc, we obtain the C code with HMPP directives from the High level description that will be introduced in next section. Then the HMPP compiler will automatically generate the OpenCL/CUDA kernel code of motion estimation. So our rapid prototyping methodology with HMPP compiler can greatly simplify the procedure of manually writing the OpenCL code.

3.5.1 CAL Description of Motion Estimation

RVC-CAL is one high level language of description which is designed to describe the reconfigure video coding standard. Now, based on our rapid prototyping framework, RVC-CAL is not only used in video coding, but also in some image and video processing algorithm, like motion estimation and stereo matching. As described in chapter 2, the Model of Computation defines the behavior of a program. Applications are described as a block diagram with RVC-CAL, also known as network or configuration, where blocks are the aforementioned FUs. RVC defines a XML-based format called FU Network Language (FNL) that is used for the description of networks, also named the XML Dataflow Format (XDF). A FNL network consist of parameters and variables, ports, and a directed graph whose vertices may be instances of FUs. So our ME application is described as a dataflow graph shown in Figure 3.14.

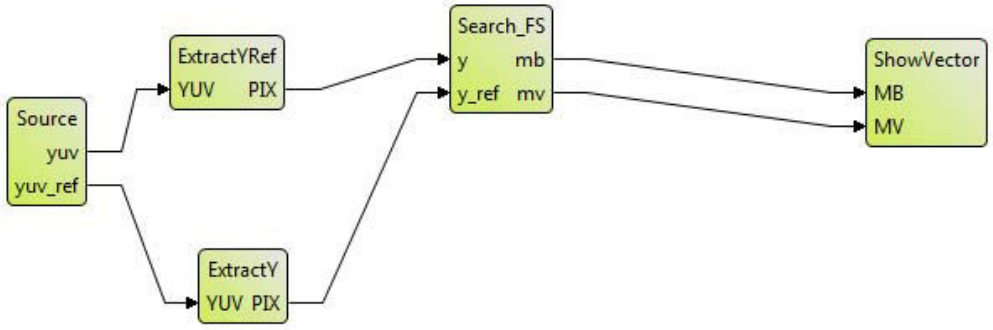


Figure 3.14: Motion Estimation of SDF diagram

In the block diagram of the motion estimation, the block "source" indicates the function of *load_file*, a FU that loads video sequence; the block "ExtractYRef" and "ExtactY" indicate the function of *extractY*, a FU that extracts Y channel from the current and reference frames of YUV video sequence; the block "Search_FS" indicates the function of *fullSearchME*, a FU that does the full search motion estimation; the block "ShowVector" indicates the function of *display*, a FU that shows the calculated motion vectors.

As shown in Figure 3.15(A), there is not efficient way to manage the multi-

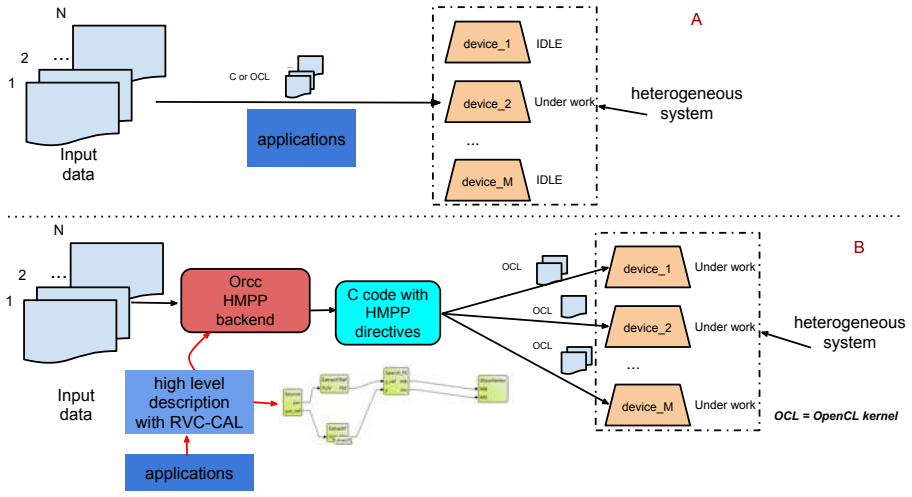


Figure 3.15: Our Prototyping methodology

devices and to distribute workload, few even only one device can participate in the applications. We have built the prototyping framework on heterogeneous parallel system for different image and video applications. The function of our prototyping methodology are: first, HMPP backend of Orcc translates the CAL description into C code with HMPP directives; second, with help of high level compiler HMPP, our prototyping methodology manages these multi devices like CPU, GPU, and generates multiple target kernel codes like OpenCL kernels and distributes suitable workload for different devices as shown in Figure 3.15(B).

3.5.2 Analysis of experiments result

To the best of our knowledge, there are several criterions of evaluation for final results of our implementations: *algorithm accuracy*, *time efficiency*, *percentage of shared memory* and *compute units occupancy*. For the *algorithm accuracy*, we compare the output (motion vector) of our C, manually writing OpenCL and HMPP OpenCL implementations, all the results are consistent. Figure 3.16 illustrates that HMPP OpenCL kernel cannot reach the same performance comparing with manual writing OpenCL kernel. As described in Section 3.2, our manual OpenCL implementation decompose all the workload into two OpenCL kernels *kernel_compute* and *kernel_compare* and employ well the shared memory. But in the HMPP OpenCL implementation, the compiler generates only one top kernel which contains functions

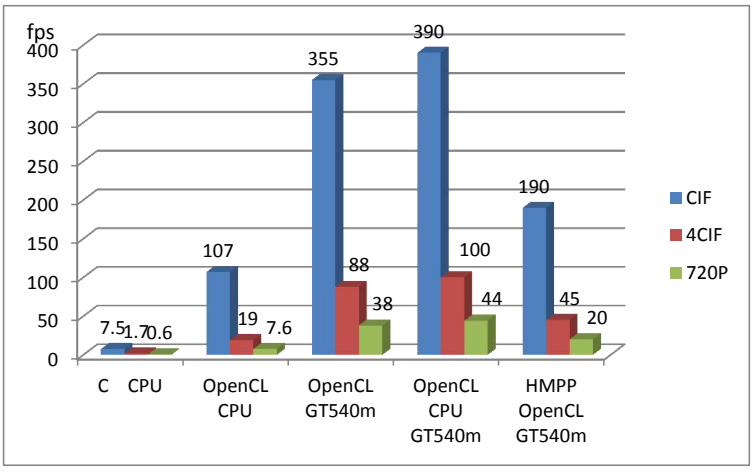


Figure 3.16: OpenCL kernel time efficiency analysis

of SAD computation and comparison.

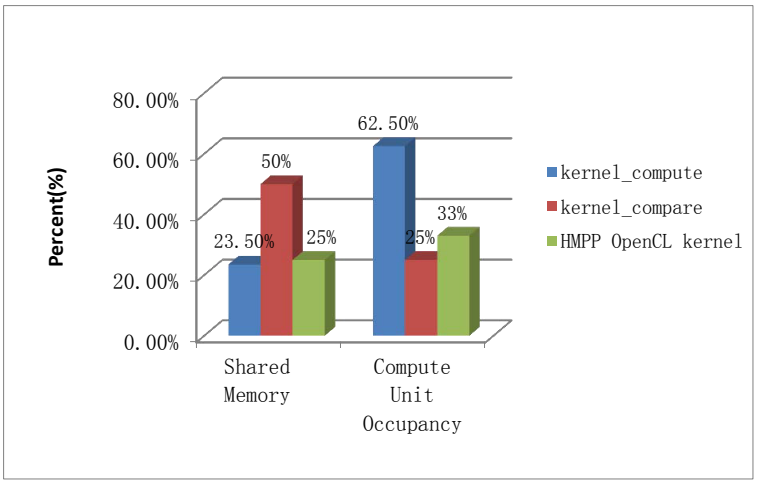


Figure 3.17: Performance difference analysis

So we employ the NVIDIA profiler to inquire some details about execution of these kernels, and Figure 3.17 shows these reasons of performance difference between manual and HMPP kernel. Because manual writing kernels: *kernel_compute* and *kernel_compare* possess 23.5% and 50% in usage of shared memory, 62.5% and 25% in the compute unit occupancy. But the HMPP OpenCL kernel only possesses 25% and 33% in usage of shared memory and compute unit occupancy respectively. We have discussed in Section 3.3, the shared memory and compute unit occupancy are the key factors which impact the performance of our algorithms executed on specified GPU devices. The HMPP OpenCL kernel employs less shared memory and compute

unit than manually writing kernel. This is the reason why it cannot rival manually writing kernel.

3.6 Conclusion

This chapter has presented one parallelized ME algorithm with OpenCL, and the proposed algorithm executed on heterogeneous parallel system which contains CPU and GPU. The experimental results show that, our implementation has the better performance than other GPU-based FSME implementations and also obtains better balance between time efficiency and PSNR than the state-of-the-art fast ME algorithms. And our method has the better speed-up ratio comparing with other GPU-based implementations of ME due to our optimizations like shared memory and vectorization.

We have also developed one basic method to find the balance of workload on heterogeneous parallel computing system with OpenCL. Additionally, experimental results show that we have found the accurate method to distribute the workload in video applications based on heterogeneous computing system which achieves obviously enhancement of performance. With the proposed rapid prototyping framework, we can directly and rapidly generate the target code like OpenCL/CUDA for heterogeneous system. In the meantime, we analyze the performance of the two kinds of methods porting source C code to the code executed on GPU.

4 Real Time Local Stereo Matching Method

4.1 Introduction

Due to the high computational complexity, stereo matching is still a great challenge to obtain high quality disparity map in real-time applications. In the case of stereo matching algorithm, the two major concerns are its matching accuracy and time efficiency. To improve them, many algorithms are proposed and splitted into two categories: *global* methods and *local* methods. *Global* methods rely on global energy minimization function along a single scanline or on the entire disparity map. These methods like Belief Propagation [45] [58] [68] [69], or Graph-Cuts [70] [71], or dynamic programming [72] [73] [74], provide good quality but involve a huge computation. Consequently few global methods are suitable in real-time applications (more than 15 frames per second (fps)).

Comparing with global methods, *local* methods provide faster processing speed, but less accurate disparity maps. They are area-based algorithms which compute the disparity for each pixel with the help of the neighboring pixels. Because of this constraint of area, local algorithms can fail probably in occluded regions and less textured regions. To continue our discussion in section 2.5.2, the area-based stereo matching algorithm is divided into four stages:

1. Cost construction
2. Cost aggregation
3. Disparity determination

4. Disparity refinement

The popular costs calculations methods for pixel $p(x, y)$ in the left image I_l and the right image I_r are listed as followed. The disparity is defined as d, and the $n \times m$ aggregation region is built in the below equation 4.1.

$$\sum_{i=n}\sum_{j=m} = \sum_{i=-|\frac{n}{2}|}^{|\frac{n}{2}|}\sum_{j=-|\frac{m}{2}|}^{|\frac{m}{2}|} \tag{4.1}$$

The raw matching costs for all disparity $d \in [d_{min}, d_{max}]$ are aggregated within a certain neighborhood window named block. There are some popular cost methods: sum of absolute differences(SAD)(4.2), Truncated Absolute Difference(TAD)(4.3), sum of squared differences(SSD)(4.4), normalized cross correlation(NCC)(4.5), zero mean sum of absolute differences(ZASD)(4.6).

$$SAD = \sum_{i=n}\sum_{j=m} |I_l(x+i, y+j) - I_r(x+d+i, y+j)| \tag{4.2}$$

$$TAD = \sum_{i=n}\sum_{j=m} min(|I_l(x+i, y+j) - I_r(x+d+i, y+j)|, \sigma) \tag{4.3}$$

$$SSD = \sum_{i=n}\sum_{j=m} (I_l(x+i, y+j) - I_r(x+d+i, y+j))^2 \tag{4.4}$$

$$NCC = \frac{\sum_{i=n}\sum_{j=m} |I_l(x+i, y+j) - I_r(x+d+i, y+j)|}{\sqrt{\sum_{i=n}\sum_{j=m} I_l(x+i, y+j)^2 \sum_{i=n}\sum_{j=m} I_r(x+d+i, y+j)^2}} \tag{4.5}$$

$$ZASD = \sum_{i=n}\sum_{j=m} |(I_l(x+i, y+j) - \bar{I}_l - (I_r(x+d+i, y+j) - \bar{I}_r| \tag{4.6}$$

where $\bar{I} = \frac{1}{nm}\sum_{i=n}\sum_{j=m}(I(x+i, y+j))$.

Different stereo matching cost

Stereo matching technique has many kind of matching criterions, named raw matching costs as shown in equations (4.2, 4.3, 4.4, 4.5, 4.6). Different with above cost computations, there is another strategy to employ a local transform of paired images. Some transforms are the Census and the Rank transforms proposed in [75] [76] [43] [77] [78] [79] [80] [81] [82]. All these transforms are based on intensity relations of between the actual pixel and neighbors pixels in one certain window. The more details information about Census transform will introduced in section 4.2. And the time-consumed of Census based matching mainly depends on Census window size. Due to the similarities between neighboring pixels as well as the intensity-value differences between corresponding pixels, classical matching measures based on intensity similarity produce slightly imprecise results. Hermann *et al.* [83] provided experimental evidence to show that the gradient as a data descriptor outperforms other pixel-based functions such as absolute differences and the Birchfield and Tomasi (BT) cost functions. Furthermore, analysing the effect of the cost functions when exposure and illumination settings are different between the left and right camera is analysed. Our experimental results in section 4.3 also illustrates the matching results under different condition of illumination. Mei *et al.* [76] combines the Census cost with the Absolute Difference (AD) cost in RGB channels, and consequently it is in the top 5 of Middlebury benchmarks [42] with complex costs aggregation and post processing of disparity. Our method initializes the Truncated Absolute Difference (TAD) and the Census cost only on gray channel transformed from the RGB channels. It avoids repeatable computation in cost construction. Our method also employ the sparse cost aggregation to improve the time efficiency with sacrifice in matching accuracy as shown in section 4.3.

Zhou *et al.* [80] and Pinggera *et al.* [84] also proposed a relative gradient stereo matching algorithm. Boundary and low texture problems are resolved by using a Gaussian weighting function and by limiting the search range. Experimental results show their local method is effective and robust, and even outperforms some of the well-known global methods. Kaaniche *et al.* [81] proposed a novel approach, based

on vector lifting schemes (VLS), which offers the advantage of generating two compact multi-resolution representations of the left and the right views. Trinh *et al.* [82] demonstrated unsupervised learning of a 62 parameter slanted plane stereo vision model involving shape from texture cues. Their approach to unsupervised learning is based on maximizing conditional likelihood. The shift from joint likelihood to conditional likelihood in unsupervised learning is analogous to the shift from Markov random fields (MRFs) to conditional random fields (CRFs). The performance achieved with unsupervised learning is close to that achieved with supervised learning for this model.

We mainly focus on paired image methods as the essential components stereo matching. However, it should be relatively straightforward to transform these methods to include multi-images application like multiple-baseline stereo matching [85] and its plane-sweep generalizations [86, 87]. Speers *et al.* [88] explored the use of feedback from the environmental model being constructed to the static stereopsis task. A prior estimate of the disparity field is used to seed the stereo matching process within a probabilistic framework. Wang *et al.* [89] presented a novel global stereo model that makes use of constraints from points with known depths, i.e., the Ground Control Points (GCPs) as referred to in stereo literature. Their formulation explicitly modeled the influences of GCPs in a Markov Random Field. A novel GCPs-based regularization term is naturally integrated into our global optimization framework in a principled way using the Bayes rule. The optimal solution of the inference problem can be approximated via existing energy minimization techniques such as graph cuts used in this paper. Their generic probabilistic framework allows GCPs to be obtained from various modalities and provides a natural way to integrate the information from multiple sensors.

GPU-based stereo matching

Facing with dense and complexity computation, high-accuracy stereo matching algorithms are very hard to perform with traditional CPU platform in real-time. In the meantime, the nature of stereo matching algorithms has a great advantage of employing the parallelism of GPU devices. Hosni *et al.* [46] suggested to use

the recently proposed guided filter to overcome this limitation. Analogously to the bilateral filter, this filter has edge preserving properties, but can be implemented in a very fast way, which makes their stereo algorithm independent of the size of the match window. The GPU implementation of stereo algorithm can process stereo images with a resolution of 640×480 pixels and a disparity range of 26 pixels at 25 fps.

Zhu *et al.* [90] proposed a dense stereo matching method implemented using CUDA. It presented the general strategy of the parallelization of matching methods on GPUs, the tradeoff between accuracy and time-consumed on current GPU hardware. Two representative and modern widely-used methods, the Sum of Absolute Differences (SAD) method and the Semi-Global Matching (SGM) method, are used and their results are compared using the Middlebury test sets. Our work of stereo matching also focus on the tradeoff between accuracy and time-comsuming on our heterogeneous platforms which will be discussed in section 4.3.

Stereo applications

Stereo matching technique has a widely applications in computer vision and pattern recognition as follows [91] [92] [93] [94] [95]: Nefian *et al.* [91] proposed one stereo matching technique used in NASA plans manned missions to generating accurate three dimensional planetary models. Their methods described a stereo correspondence system for orbital images and focuses on a novel approach for the sub-pixel refinement of the disparity maps which uses a Bayesian formulation that generalizes the Lucas-Kanade method for optimal matching between stereo pair images. Their method was demonstrated on a set of high resolution scanned images from the Apollo era missions.

Kuhl *et al.* [92] employed the stereo matching technique in navigation and obstacle avoidance of mobile robots. Depth maps provide an essential description of the world seen through cameras to satisfy the perception and identification of the surrounding environment. They also investigated with different types of stereo matching algorithms to build and to implement a system for generating depth maps on a mobile robot. Bleyer *et al.* [93] addressed the problem of computing a sequence

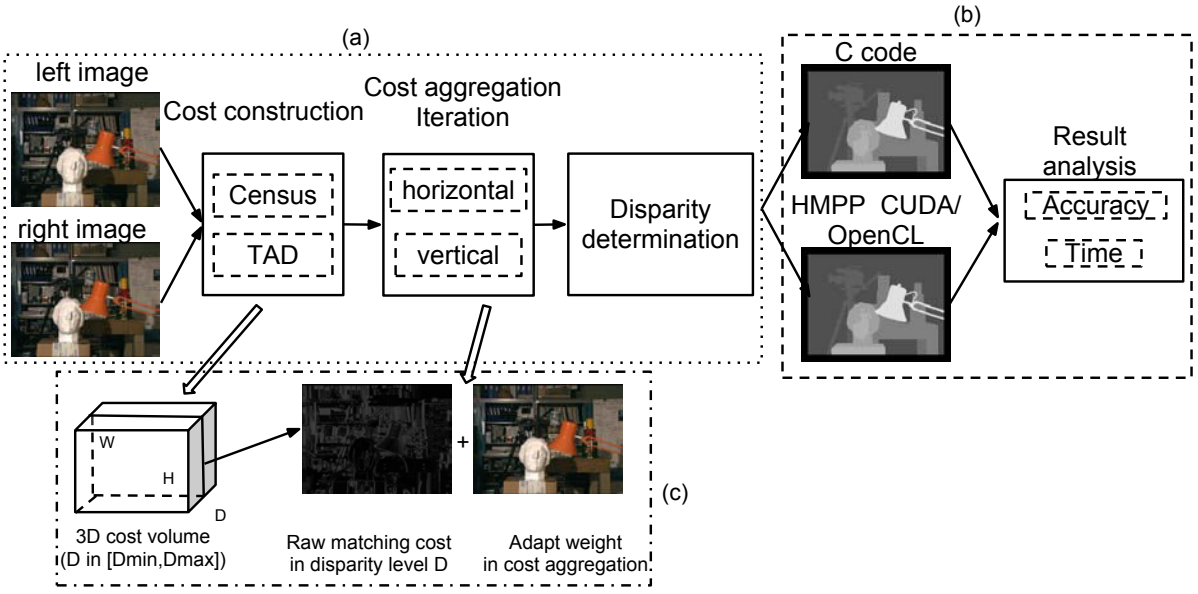


Figure 4.1: Proposed stereo matching system

of dense disparity maps from two synchronized video streams recorded by slightly displaced cameras. They also demonstrated the good quality results using various challenging real-world stereo streams. Heath *et al.* [96] described and evaluate a vision-based technique for tracking many people with a network of stereo camera sensors. Sensors communicate a subset of their sparse 3D measurements to other sensors with overlapping views. Their method achieved a tracking precision of 10-30 cm in all cases and good tracking accuracy even in crowded scenes of 15 people.

In all, one of our two main aims in this chapter is to propose a local stereo matching algorithm that greatly balances the matching accuracy and time efficiency. Our method is shown in the first part of Figure 4.1(a), mainly contains the first two stages: In the first stage, the cost volume is constructed by combining the TAD cost with the Census cost; In the second stage, the matching cost is aggregated by applying a weighted and iterative method in both directions to produce the final disparities. The second part of Figure 4.1(b) also illustrates our method used to implement our stereo matching algorithm on a GPU platform. This is our second main aim in this chapter.

This chapter is organized as follows: Section 4.2 describes our real time stereo matching method; Section 4.3 presents experimental results about matching accu-

racy and time efficiency from the Middlebury Stereo Database [42] and CAL description for different architectures; Section 4.4 gives a brief conclusion.

4.2 Proposed Stereo Matching Algorithm

Extending from Figure 4.1(a), Figure 4.2 elaborates proposed stereo matching method which consists of color-space transforming, cost construction, cost aggregation, disparity selection and involved parameters like iteration times, maximum-minimum disparity and so on. In the beginning, the left and right RGB images are transformed into gray images due to simplify the computation.

$$Gray = 0.299 * R + 0.587 * G + 0.144 * B; \tag{4.7}$$

Our cost construction combines the Truncated Absolute Difference (TAD) cost with the Census cost. The cost aggregation computes the matching cost by a weighted method, which is iterated on both horizontal and vertical directions. In the third stage, the Winner-Takes-All (WTA) method is applied to determine the final disparities.

4.2.1 Combined cost construction

There are various methods of cost measure such as TAD or intensive-based or gradient-based measure, but few methods combine cost measure. In fact a suitable combination can greatly improve the accuracy of the raw matching. To this end, we define the measure method that combines the TAD cost with the Census cost. In addition, our local method distinguishes each pixel at each disparity as shown in Figure 4.1(c). The TAD cost $C_{TAD}(pl, d)$ computes the intensity absolute difference between the left image and the d-shifted right image, truncated by the σ threshold:

$$\begin{aligned} C_{TAD}(pl, d) &= min(|i_{left}(pl) - i_{right}(pr)|, \sigma) \\ &= min(|i_{left}(x, y) - i_{right}(x - d, y)|, \sigma), \end{aligned} \tag{4.8}$$

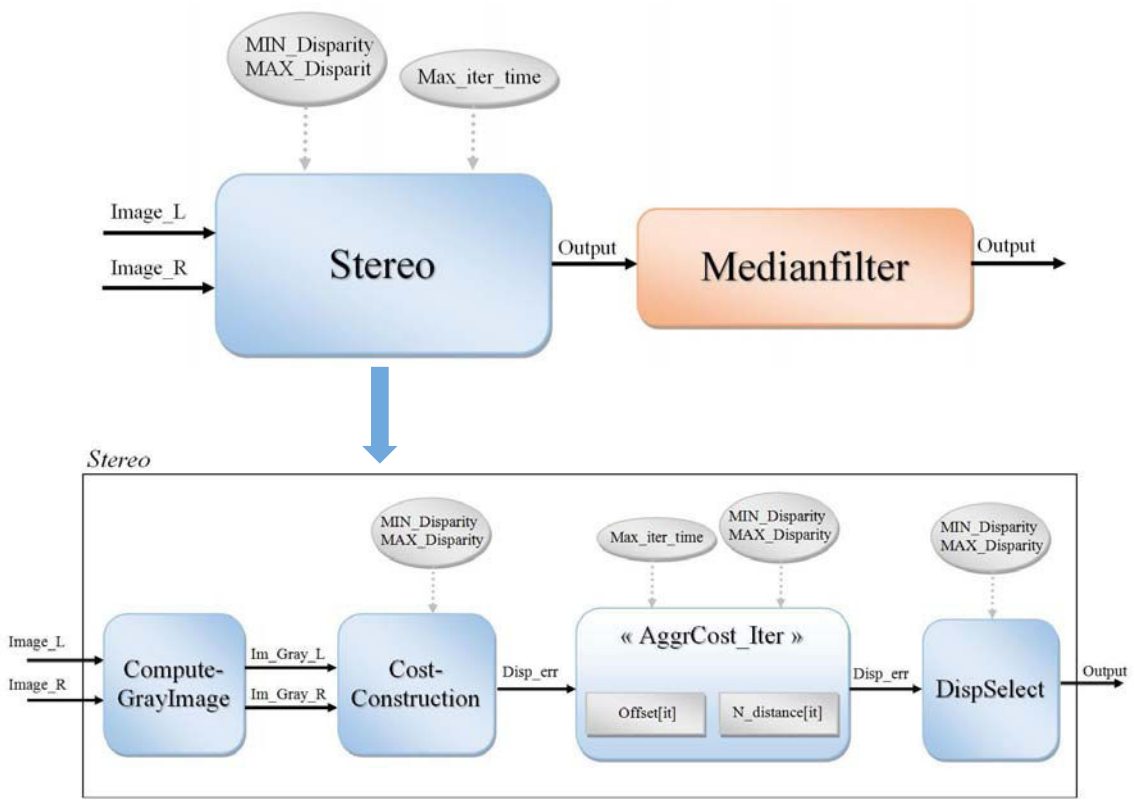


Figure 4.2: Detailed description of our proposed method

where $pl = (x, y)$ is the coordinates of the pixel in the left image, and given a disparity d, $pr = (x - d, y)$ is the coordinates of the corresponding pixel in the right image. $i_{left}(pl)$ and $i_{right}(pr)$ are the gray-level intensities of the pl and pr pixels respectively.

The census cost $C_{census}(pl, d)$ computes the Hamming distance by applying bit-wise XOR operations between both the 8-bit strings of the pl and pr pixels. As shown in Figure 4.3, a 3×3 census window permits to transform the neighborhood structure of a central pixel into a 8-bit string: every bit is set to 0 if the neighboring pixel in the census window has a lower intensity than the intensity of the central pixel, and 1 otherwise [43]. Figure 4.4 illustrates census transforming of teddy paired images.

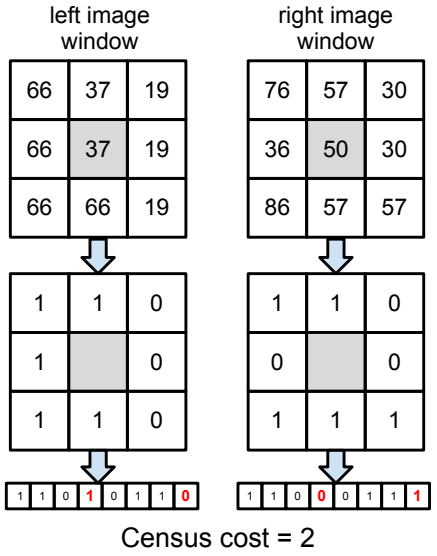


Figure 4.3: Census transforming example

The construction cost $C(pl, d)$ is then calculated as:

$$\begin{aligned} C(pl,d) =& \rho(C_{census}(pl,d), \lambda_{census}) \\ &+ \rho(C_{TAD}(pl,d), \lambda_{TAD}), \end{aligned} \tag{4.9}$$

$\rho(c, \lambda)$ is one robust function based on variable c:

$$\rho(c,\lambda) = 1 - exp(-c/\lambda). \tag{4.10}$$

Equation (4.9) combines the two costs to the range from 0 to 1 and controls the outliers by the λ parameters.

4.2.2 Proposed cost aggregation method

Cost aggregation consumes the most time of a stereo matching algorithm. Thus our method limits the aggregation of neighboring pixels in order to reduce the spatial complexity, while maintaining data parallelism required for an efficient implementation on GPU.

To this end, we employ adaptive weights in our implementation: this approach is to adjust weight of each pixel based on the intensity of neighboring pixels and on

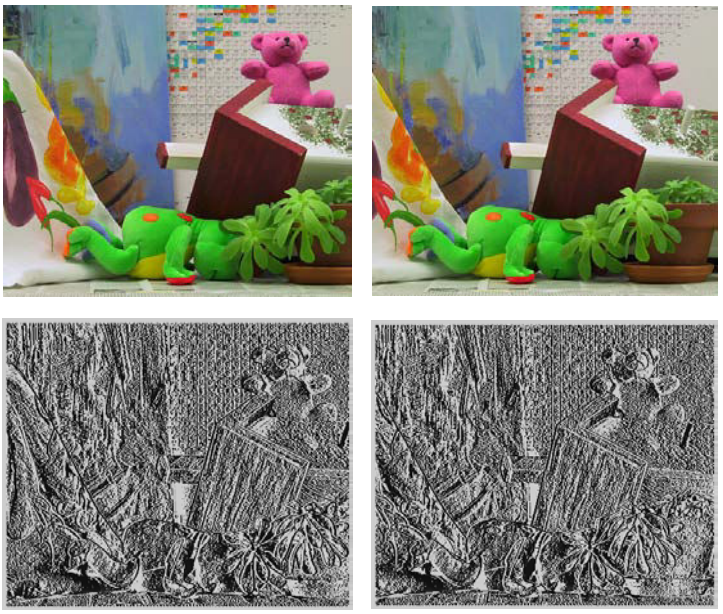


Figure 4.4: Census transforming of Teddy images

a geometric relationship with the central pixel [97]. Basically, a higher weight will be assigned to a pixel if its intensity and its distance are closer to the central pixel. We define our adaptive weight $w(pl, q)$ as follows:

$$w(pl, q) = exp(-|I_{pl} - I_q|/\lambda_I - ||pl - q||/\lambda_d) \tag{4.11}$$

where pl is the central pixel of the left image and q is a pixel in the neighborhood of pl. $|I_{pl} - I_q|$ is the intensity absolute difference between the pl and q pixels, while $||pl - q||$ is the Euclidean distance between them. λ_I and λ_d are constant parameters that adjust the balance of both kinds of weight.

Instead of processing with a $N \times N$ fixed window, we perform a cross approach on the image that costs are aggregated in a first horizontal pass followed by a second vertical pass. To cover the specified range of influence, several iterations are performed in both directions to get the final aggregated cost. Thus, at each iteration i, each pixel is the aggregation of the costs of three pixels, i.e. the target pixel and

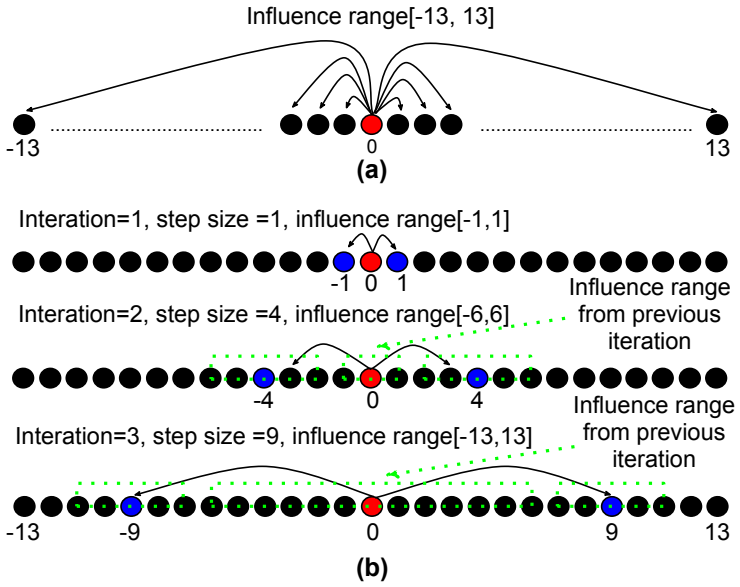


Figure 4.5: 1D Cost aggregation example

two pixels with offset $-s$ and $+s$:

$$\begin{aligned} C(pl,d) += & C(pl+s,d) * w(pl,pl+s) \\ & + C(pl-s,d) * w(pl,pl-s) \end{aligned} \tag{4.12}$$

where s is the size of the iteration step: a squared step ($s = i^2$) proposed in our implementation in order to reduce the complexity.

Figure 4.5 illustrates our aggregation in the horizontal direction (the same process is repeated in the vertical direction). Figure 4.5(a) shows the conventional method that performs the processing on 27 pixels [98] for the influence range $[-13, 13]$. Figure 4.5(b) describes our iterative method to achieve the same influence range: at least three iterations are needed with the respective values of s equal to 1, then 4, and 9. Consequently the range varies from $[-1, 1]$, then $[-6, 6]$, up to $[-13, 13]$. Thus, our iterative method performs the processing on only 9 (3×3) pixels, which provides a significant reduction compared to the conventional method.

4.2.3 Disparity selection

Among all the disparity assumptions, the Winner-Takes-All (WTA) method is performed to determine the disparities:

$$D(pl) = \underset{d \in [d_{min}, d_{max}]}{\arg\min} \; C(pl, d) \tag{4.13}$$

4.2.4 Post processing of disparity maps

To improve the disparity maps and eliminate the bad matching pixel, we calculate the disparity maps both the left and right image respectively for post processing of disparity maps. Normally the disparity maps contain outliers in the occlusion and depth discontinuities regions [99] [100] [101]. The first step is detecting these outliers as much as possible with left-right consistency check; the second step is adopting refinement methods to fill them with nearest reliable disparities.

left-right consistency check

Left-Right Consistency (LRC) check is performed to get rid of the occlusion (objects scene in one image, and not in other) pixels in the final disparity map. This is computed by taking the computed disparity value in one image, and re-projecting it in the other image. If the difference in the values is less than a given threshold, then the pixels are half-occluded. In the occlusion maps, the light areas are occlusion parts and are not reliable, thus the disparities in these areas are not reliable either. The function of consistency check is to detect these occlusion areas and modify them with the closest reliable disparities. The adjustment of reliable disparity is given in Equation 4.14.

$$\begin{cases} |D_L(x,y) - D_R(x - D(x,y), y)| \leq threshold, left \\ |D_R(x,y) - D_L(x + D(x,y), y)| \leq threshold, right \end{cases} \tag{4.14}$$

where $D_L(x, y)$ is the disparity of left image, and the $D_R(x, y)$ is the disparity of right image.

In addition, the consistency check also makes up the corresponding unreliable

boundaries (left boundary of left depth map and right boundary of right depth map) by replacing them with the nearest reliable disparities in the same scan-line.

Region Voting

The detected outliers should be filled with reliable neighboring disparities value. Most accurate stereo algorithm employ segmented regions for outlier refinement [100, 101]. Region voting is used to select the most frequently emerged disparity in the support region. For one outlier pixel p, the voting firstly build a histogram $H(p)$ over all the disparities from 0 to D_{max} in a region $U(p)$; then adopting cross voting approach to select the disparity with the highest bin value (defined as D_{maxp}) in disparity histogram $H(p)$, as shown in Equation 4.15 and Figure 4.6.

$$D_{maxp} = argmax H(d), d \in [0, D_{max} - 1] \tag{4.15}$$

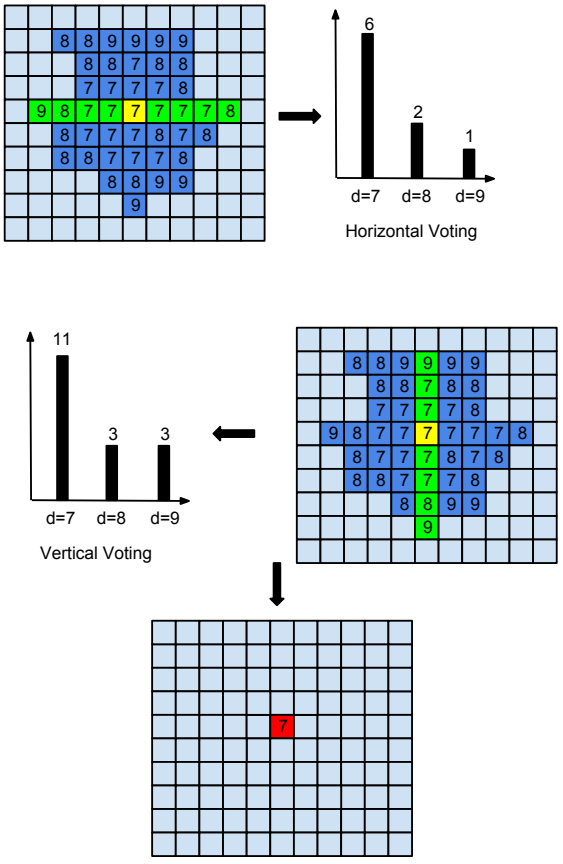


Figure 4.6: $2 \times 1D$ horizontal and vertical voting

4.2.5 CAL description of stereo matching

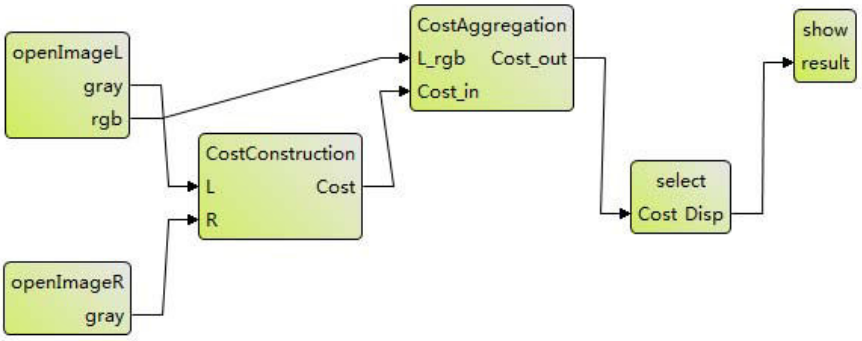


Figure 4.7: Proposed stereo matching of SDF diagram

As shown in Figure 4.7, our real time stereo matching method is described as a dataflow graph. In the block diagram of the stereo matching, the block "openImageL" and "openImagR" indicate the function of *load_file*, a FU that loads left and right paired images; the block "CostConstruction" indicates the function of *costConstruction*, a FU that calculates row matching costs; the block "CostAggregation" indicates the function of *AggrCost_H* and *AggrCost_V*, FUs that aggregate the row matching costs along the horizontal and vertical directions; the block "select" indicates the function of *dispSelect*, a FU that determines the final disparity; the block "show" indicates the function of *display*, a FU that shows the final disparity.

In the meantime, we also proposed another description of stereo matching. It is designed to distributes the specified workload on multi-DSP platforms as shown in Figure 4.8. The main difference between Figure 4.8 and Figure 4.7 is the part of cost aggregation. As we previously described, we assume that the disparity range is from D_{min} to D_{max} ($D \in [D_{min}, D_{max}]$). We already know that the cost aggregation is the most time consumed part in the stereo matching algorithm, above 90% of total computation. And all these computation of cost aggregation is based on each level of disparity assumption. So we can optimize cost aggregation by averaging all computation of aggregation into the channels with fixed number (the number is 5 in Figure 4.8) to obtain the better performance. For instance, the disparity range is 60 ($D \in [0, 59]$). Every channel is in charge of 12 level

disparity($[0, 11], [12, 23], [24, 35], [36, 47], [48, 60]$ respectively). The block "rebuild" indicates a FU that reorganize these costs from different channels into final costs.

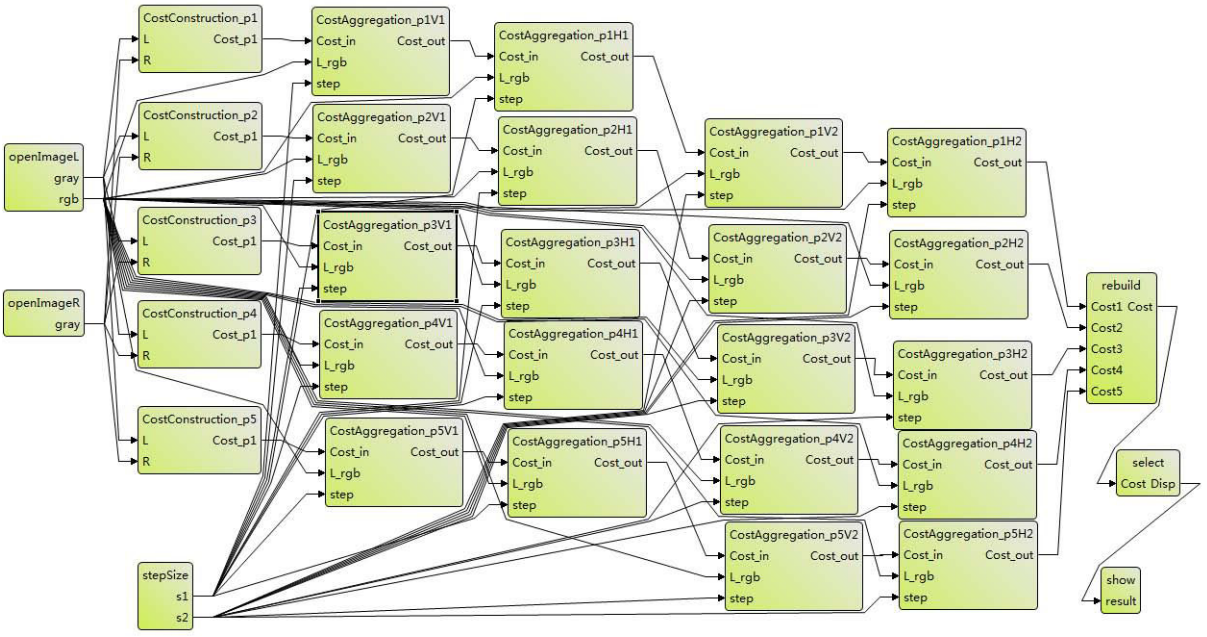


Figure 4.8: Proposed multi-pass stereo matching method XDF graph

4.3 Experimental Result and Discussions

For stereo matching algorithm, many test source and test scene are proposed. Andorko *et al.* [102] proposed a new disparity map testing scene for consumer electronic devices. The Middlebury stereo database [42] is the most famous and widely used test benchmark. It has many standard paired images with ground truth. Stereo matching methods calculate the disparity from these paired images, submit to the online benchmark system and obtain the ranking in whole already submitted methods according to the percentage of bad matching pixels. We evaluated the performance of our proposed stereo matching method with the benchmark Middlebury stereo database [42]. These parameters are constant in our experiments on several stereo datasets, i.e, $\sigma = 12$, $\lambda_{census} = 6$, $\lambda_{TAD} = 20$, $\lambda_I = 15$, $\lambda_d = 15$. And we analyse our matching algorithm from its matching accuracy and time efficiency.

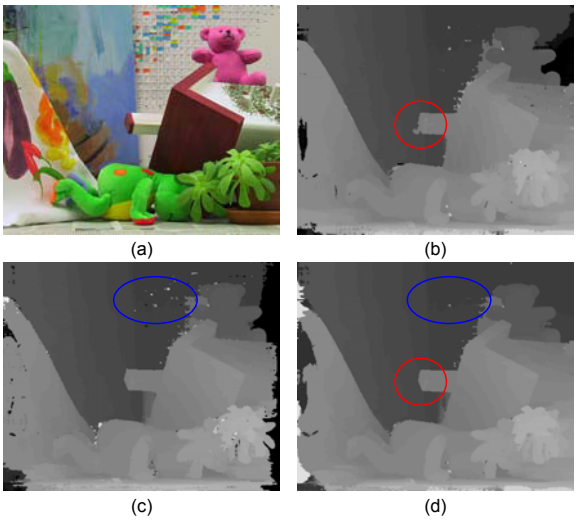


Figure 4.9: Disparity results comparison,(a) Teddy image reference, (b) ESAW method [103], (c) RTcensus method [43], (d) our method

4.3.1 Matching accuracy

Figure 4.9 presents the visual and quantitative result of the dataset Teddy comparing with the two methods: ESAW [103] and RTcensus [43]. Figure 4.13 shows our disparity results and bad pixels maps of Middlebury Stereo Evaluation. Our disparity map outperforms the other two methods in the discontinued region (red circle mark) and flat region (blue circle mark). Table 4.1 lists the error rate of our experimental algorithm according to the evaluation of Middlebury. The error rate in non-occluded (nonocc) regions, all regions (all), and depth-discontinuity (disc) regions are illustrated respectively. As we discussed in the last section, the AD-census method proposed by Mei *et al.* [76] is in the top 5 of Middlebury benchmarks. Different with our method, it employees the different way of combination of AD and Census costs and aggregate all the possibility in the support region for better matching accuracy. But better accuracy brings huge time consuming of 15 seconds on teddy paired images on Intel Core2Duo 2.2Hz CPU platform who is very hard to be optimized to implement on embedded system in real-time.

In the meantime, we consider and verify the "real-world" problems of illumination condition. We compare the intensity-based method like ESAW with our method in Figure 4.10. The experiment results indicate that our method outperforms the

Table 4.1: Quantitative Evaluation Results of the Middlebury Stereo Database [42]

algorithm	Tsukuba			Venus			Teddy			Cones			average
	nonocc	all	disc	nonocc	all	disc	nonocc	all	disc	nonocc	all	disc	
AdaptingBP	1.11	1.37	5.79	0.10	0.21	1.44	4.22	7.06	11.8	2.48	7.92	7.32	4.23
PMBP	1.96	2.21	9.22	0.30	0.49	3.57	2.88	8.57	8.99	2.22	6.64	6.48	4.46
Our method	**1.03**	**2.01**	**5.06**	**0.21**	**0.46**	**1.95**	**6.67**	**12.0**	**13.8**	**3.51**	**10.4**	**9.9**	**5.58**
ESAW [103]	1.92	2.45	9.66	1.03	1.65	6.89	8.48	14.2	18.7	6.56	12.7	14.4	8.21
RTCensus [43]	5.08	6.25	19.2	1.58	2.42	14.2	7.96	13.8	20.3	4.10	9.45	12.2	9.73

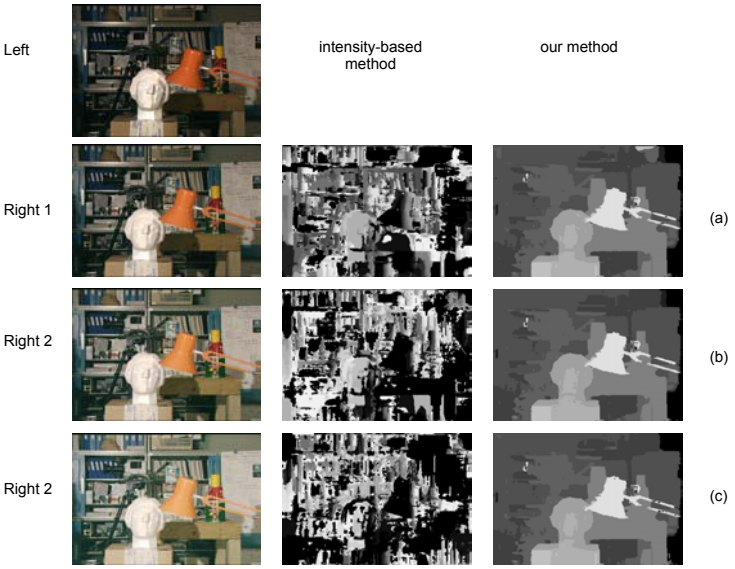


Figure 4.10: Matching results under differen condition of illumination, the left image hold the same illumination and the right image has three conditions of illumination as shown in (a,b,c)

intensity-based method in different illumination condition. Figure 4.11 describes the 3D reconstruction model with our calculated disparity which could reflect the accuracy of matching results.

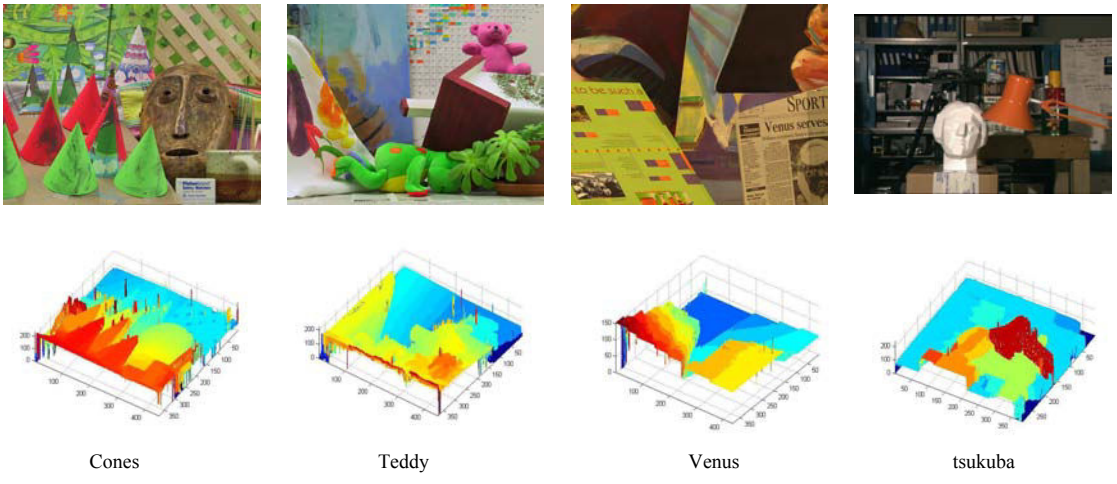
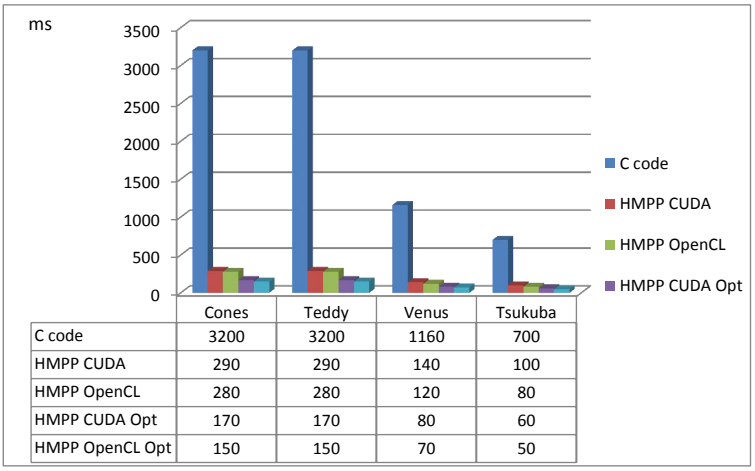


Figure 4.11: 3D reconstruction with depth information



	Cones	Teddy	Venus	Tsukuba
C code	3200	3200	1160	700
HMPP CUDA	290	290	140	100
HMPP OpenCL	280	280	120	80
HMPP CUDA Opt	170	170	80	60
HMPP OpenCL Opt	150	150	70	50

Figure 4.12: Time consuming of Middlebury Stereo Database, HMPP CUDA and OpenCL (Opt is optimized version)

4.3.2 Time-efficiency results

As shown in Figure 4.12, our GPU-based implementations are over 10 times fast than the C implementation. Our approach obtains 700ms, 1200ms, 3200ms, 3200ms based on the CPU platform and 50ms, 70ms, 150ms, 150ms based on the GPU platform for Tsukuba, Venus, Teddy, Cones of Middlebury stereo benchmark respectively.

Table 4.2: Integrated Analysis of selected real-time methods from The Middlebury Stereo Database [42]

Method	Rank	Avg.error %	Avg.runtime (ms)	Device	Estimated time(ms)	Estimation Rank(th)
costFilter	24	5.55	65	NVIDIA GTX480	325	5
Plane-fitBP	30	5.78	650	NVIDIA Geforce 8800 GTX	about 650	7
Proposed method	**43**	**5.58**	**50**	**Nvidia GT540M**	**50**	**2**
RealtimeHD	59	6.66	15	NVIDIA GTX580	75	3
Adaptweight	70	6.67	8500	AMD2700+	about 500	6
RealTimeABW	87	7.90	2807	2.80GHz Core 2 duo	about 280	4
Real-time GPU	100	9.82	142	ATI Radeon 1800	below 50	1

Table 4.2 provides one brief introduction and benchmark comparison of these announced real-time methods from Middlebury Stereo Evaluation. Our experimental environments: GPU: NVIDIA GT 540M (96 CUDA cores), CPU: I7 2630qm, image definition: 352×288, disparity range: [0,15]. Compared with these announced real-time methods in Table 4.2, our method is not the best one about matching accuracy, but we can get the top 10 rank in 140 submissions about the time-efficiency. As described in Table 4.2, costFilter algorithm is implemented on the Nvidia GTX480

(GF100 architecture) with 480 CUDA cores while RealtimeHD algorithm is implemented on Nvidia GTX580 (GF110 architecture) with 512 CUDA cores. Considering the number of CUDA cores, GTX480 and GTX580 almost have the 5 times compute capability than our GPU device GT540m.

In order to normalize the result of time-efficiency, we estimate time-consuming of all these methods running on our GPU device GT540m in the column ***Estimation*** of Table 4.2. From the ***Estimated Rank*** about the time-efficiency of Table 4.2, our proposed method can get the second position. It means that our implementation has a great potential in terms of time-efficiency and matching accuracy.

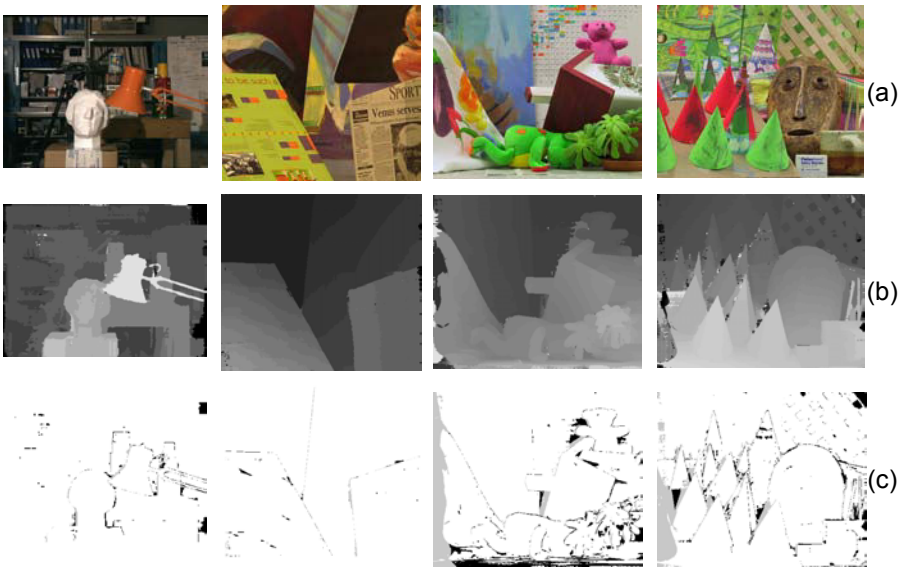


Figure 4.13: Results of Middlebury datasets: Tsukuba, Venus, Teddy and Cones. (a) datasets, (b) disparity of our method, (c) bad pixel (absolute disparity error >1.0) maps where mismatches in non-occluded areas are labeled with black, in occluded areas with gray color.

4.3.3 Proposed stereo matching method with SigmaC dataflow

In the section 2.1.2, the MPPA of KALRAY is introduced with its own dataflow approach SigmC. Because of its many cores strategy (256 cores total), the MPPA has great potential in parallel computing. Alexandre MERCAT described the proposed stereo matching method with SigmaC dataflow. The implementation of proposed method is verified on the platform of MPPA. In this SigmaC dataflow description, there are six *agents* assigned for function *ComputeGrayImage*, nine *agents* assigned

for function *CostConstruction* and *CostAggregation*, one *agents* assigned for function *DispSelect* as shown in Figure 4.14. The time consuming of Tsukuba test images is 160 ms which is about 3 times more than the GPU-based implementation. The matching accuracy has little difference between the MPPA-based and GPU-based implementation.

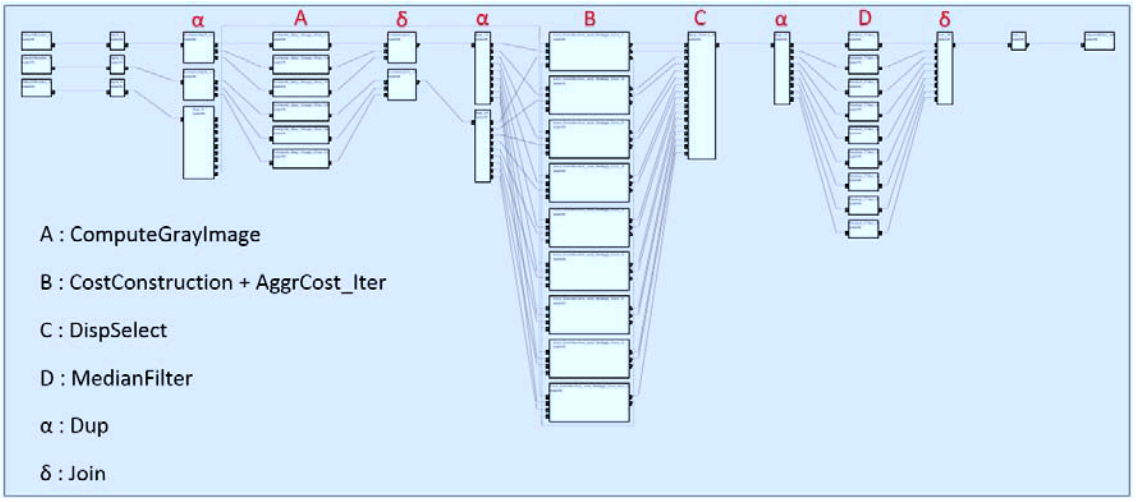


Figure 4.14: SigmaC dataflow of proposed stereo matching method

4.4 Conclusions

This chapter has presented our real-time stereo matching algorithm. Our method integrates the cost construction by combining Census with TAD measure cost, aggregates the raw matching cost by a adapt weighted method, which is iterated on both horizontal and vertical directions. The Winner-Takes-All (WTA) method is applied to determine the final disparities in the end.

Two high-level descriptions of proposed stereo matching method are introduced. One is for Orcc HMPP branch, another is designed for multi-core parallel architecture. Based on our rapid prototyping framework, OpenCL/CUDA kernel code is automatically generated for GPU Platform. Our GPU-based implementations are over 10 times faster than C implementation and the Optimized HMPP gains 25%-45% improvement about processing speed compared with default HMPP. Experimental results show that our method outperforms other real-time stereo matching methods about tradeoff between matching accuracy and time-efficiency.

5 Joint Motion-Based Video Stereo Matching

5.1 Introduction

For static paired image, estimating disparity has been well developed in two last several decades. For video sequences, most of stereo matching methods only take care of the "spatial intra-frame" (i.e,paired images with the same order) and a few methods not involve the "temporal inter-frame" (i.e, consequent paired images in video sequence). The use these of spatial intra-frame methods leads to flicker-frames and worse bad matching results.

To deal with this problem, Lee *et al.* [77] proposed one local method based on the optical flow in a video for the disparity estimation. But their method cost 19 seconds for paired frames at 400×300 resolution which is not fast enough in real applications. Most of the time is spent for the motion estimation in video sequence. Khoshabeh *et al.* [104] proposed one method based on the spatio-temporal consistency in the video to estimate the disparity. This method applies an l_1-normed minimization on a novel three dimensional Total Variation (TV) regularization (Chan *et al.* [105]). Due to their unique formulation, other static image stereo matching algorithms could use the method as a post-processing step to refine noisy. The method emphasizes how the motion cue impacts the adaptive weight, and does not consider the way to build the support region with the help of the motion cue. Support region is built by these neighbors' pixels when the costs aggregation happened. Bleyer *et al.* [93] address this problem of computing a sequence of dense disparity maps from two synchronized video streams recorded by slightly displaced cameras.

In this chapter we propose one joint motion-based local stereo matching method for video sequence. For each frame of the stereo video, our method first builds the support region for the selected central pixel with the help of motion vectors. Then it aggregates raw matching cost with adaptive motion weight and iterates with square step size in the support region along the horizontal and vertical passes.

This chapter is organized as follow: Section 5.2 introduces our stereo video matching algorithm with several procedures. Section 5.3 presents our implementation of the video stereo matching on an heterogeneous system. Section 5.4 describes our experimental results and compare them with other existing stereo video matching methods. Section 5.5 introduces our video stereo matching GUI interface which integrates our proposed method. Section 5.6 gives a brief conclusion.

5.2 Joint Motion-Based Stereo Matching

All the local stereo matching methods employ cost aggregation in support region for each pixel. In this sense, there is one common assumption that all the neighboring pixels of the support region have the same disparity. However, without any prior knowledge about disparity, all these pixels should have the similar intensity. Yoon *et al.* [106] proposed one adaptive support-weight approach. Their method aggregates cost over large support windows size (35×35) according to color similarity and geometric proximity. Zhang *et al.* [107] proposed one upright cross local method and dynamically construct a shape-adaptive support region. Their method reduced the complexity of computation due to the shape-adaptive support region instead of fixed support region. In short, the more accuracy of building support region we get, the more accuracy support weights and row matching costs we could use in the cost aggregation of video stereo matching.

5.2.1 Local support region building

Instead of a collection of values associated with individual photo receptors, human perceive a number of visual groups, usually associated with objects or well-defined parts of objects. How the eye tends to group visual information depending

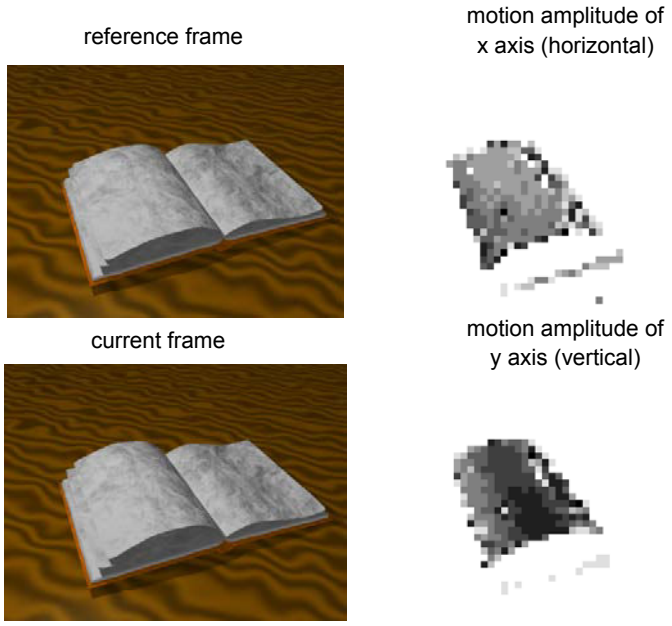


Figure 5.1: Motion amplitudes of stereo video test sequence: book

on color similarity, geometric proximity, connection and enclosure of images in relation to each other is equally important for machine vision. As described in [106], visual grouping play the significant role of building a support region and relative support-weights. There are so many visual cue used in visual grouping. Beside color similarity and geometric proximity, motion cue is also a key factor in the stereo video matching. The main idea of our proposed method is to build a support region with motion cue from motion estimation. Block-based motion estimation is the process of determining motion vectors between adjacent frames in a video sequence. We obtain accurate motion vectors for each block (these block size equal to $2^n \times 2^n$) by directly employing our parallelized GPU-based motion estimator introduced in chapter 3. In Figure 5.1, the resulting motion vectors of each block ($blocksize = 8 \times 8$) are drawn with gray-scale values for x and y axis respectively.

White color means there is no motion happened, and the darker color means the more explosive motion. We can make such a supposition: ***These motion blocks with the similar motion amplitude should have the same disparity.***

As inspired by the work of Zhang [107] and Mer [76], we build one pixelwise adaptive cross which consists of vertical and horizontal segments, crossing at pixel p. To model a suited cross for pixel p, we should determine the length of left,

right, up, bottom limit as shown in Figure 5.2. Zhang *et al.* [107] employs the color similarity under the connectivity constraint. The length of limits is only decided upon color similarity. For the selected pixel p, the largest span is defined as R, where all these pixels covered in this span have the similar color with pixel p.

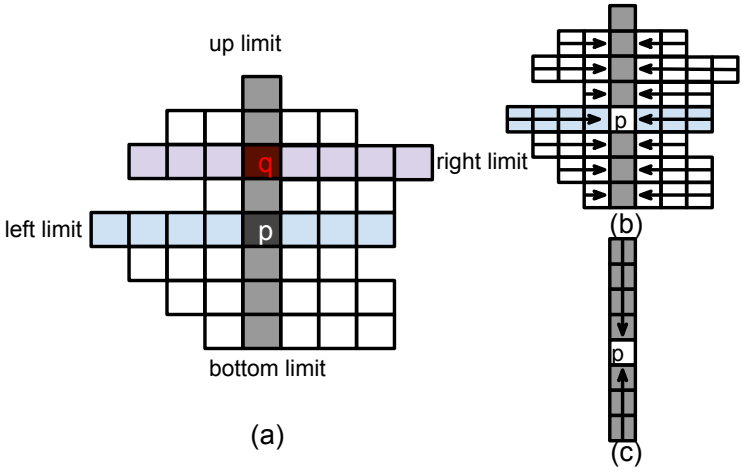


Figure 5.2: Support region of selected pixel p

$$R = \underset{r\in[1,L]}{max}(r\ \Pi_{i\in[1,r]}\delta(p,p_i)), \tag{5.1}$$

where $p = (x, y), p_i = (x - i, y)$ and L is the preset largest length of limits (the default value of L is 13). $\delta(p_1, p_2)$ is one judgement function based on the color similarity between the pixels p_1 and p_2.

$$\delta(p_1,p_2) = \begin{cases} 1,\ \max_{c\in[R,G,B]}(|I_c(p_1) - I_c(p_2)|) \le \tau \\ 0,\ otherwise \end{cases}. \tag{5.2}$$

where i_c is the intensity of color band c ($c \in R, G, B$). τ is the termination threshold in color similarity.

For stereo video matching, we propose such improvements as follows. Extending our above suppose, we also confirm that each pixel in blocks with the similar motion amplitude also have the same disparity. Firstly, each pixel of input RGB image I (current frame of video) can be described as $p = (i, (x, y))$. i is the gray value transformed form the RGB channels. (x, y) is the 2D coordinates in the input image. After the motion estimation, every pixel is defined as $p = (i, (x, y), (dx, dy))$. So the

judgment function $\delta(p_1, p_2)$ is redefined by:

$$\delta(p_1, p_2) = \begin{cases} 1, \ (|I_{gray}(p_1) - I_{gray}(p_2)|) \leq \tau_{gray} \\ 0, \ otherwise \end{cases} . \tag{5.3}$$

where avoid repetitive comparison in R,G and B channels. In the dataflow of our method, every pixel is one vector set which will benefit of following GPU computing.

Second, we make use of the combination of motion amplitudes (dx, dy) and the color similarity to determine the length of each limit as described in the new definition of largest span:

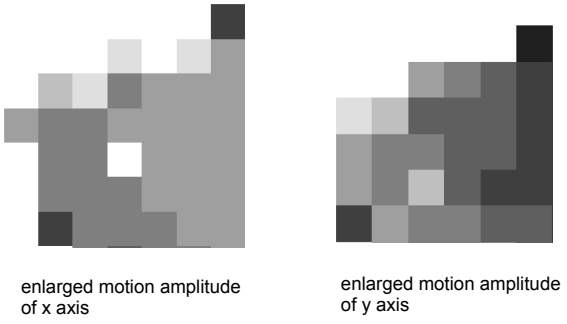


Figure 5.3: Enlarged motion amplitudes of moving page (left up corner) from Figure 5.1(sequence: book)

$$R = \max_{r \in [1,L]} (r \ \Pi_{i \in [1,r]} \delta(p, p_i) \varphi(p, p_i)), \tag{5.4}$$

where $\varphi(p_1, p_2)$ is the judgment function based on the motion amplitude between the pixel p_1 and p_2.

$$\varphi(p_1, p_2) = \begin{cases} 1, \ (|dx(p_1) - dx(p_2)| + |dy(p_1) - dy(p_2)|) \leq \tau_{motion} \\ 0, \ otherwise \end{cases} . \tag{5.5}$$

where $dx(p)$ is the motion amplitude of pixels p in x axis, $dy(p)$ is the motion amplitude in y axis.

As shown in Figure 5.1 and 5.3, the motion amplitude accurately distinguishes the edge of object (segmentation between different objects). Neighbor pixels which have similar value of intensity are only considered in color similarity judgment. It

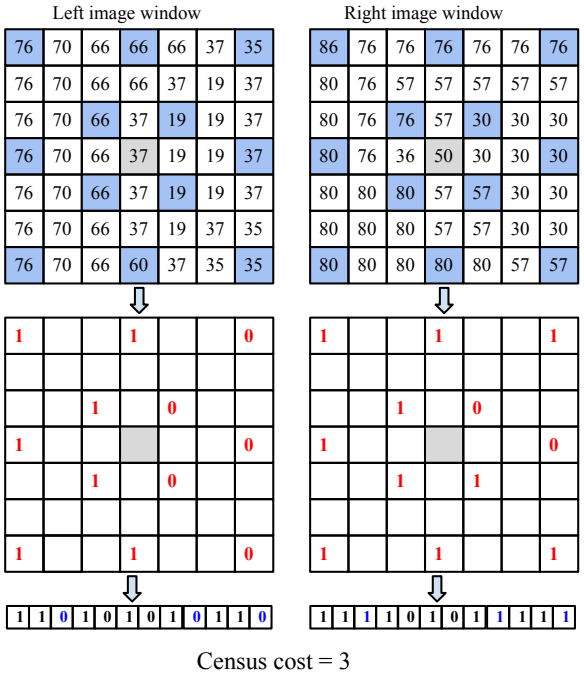


Figure 5.4: Optimized sparse census transforming example

means that our method can remove undesired noise. With Equation (5.4), we can calculate the length of left, right, up and bottom limits and obtain the support region S_p for pixel p as shown in the Figure 5.2(a).

5.2.2 Combined raw cost construction

We use the combined raw cost construction defined in section 4.2.1. The cost TAD $C_{TAD}(pl, d)$ is calculated using equation (4.8). Different with the usual census transforming described in section 4.2.1, we proposed our sparse census transforming as shown in Figure 5.4. A 7×7 census windows is used to represent the neighbor structure. But we adopt sparse sampling of 12 pixels instead of total 49 pixels to reach the same influence region as normal census transforming. The census cost $C_{census}(pl, d)$ is defined as the Hamming distance (bitwise XOR operations) of the two bit strings about pixels pl and pr along scanline in left and right image respectively. Every bit is set to 0 if the neighboring pixel in the census window has a lower intensity than the center pixel, and 1 otherwise [43]. Our sparse sampling of census not only saves 75% of the time for computing census transforming, but also brings

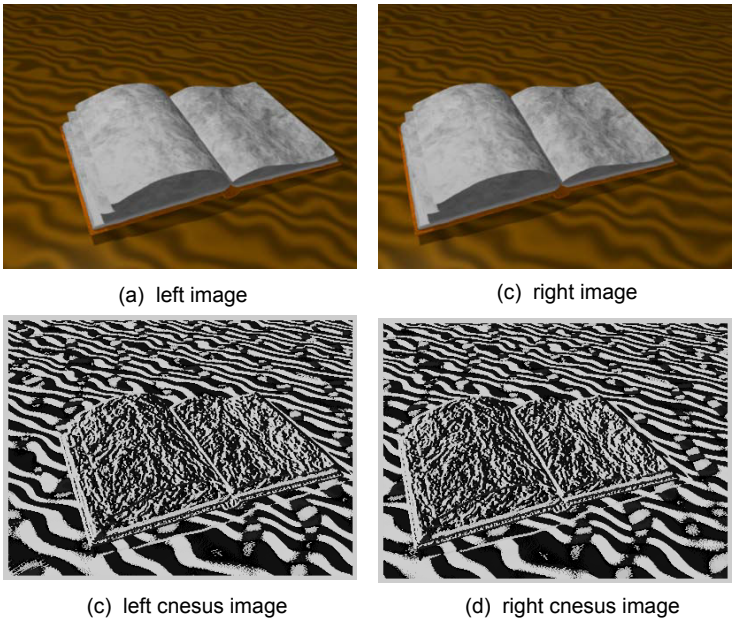


Figure 5.5: Census transformed image (books)

the similar result about the accuracy. Figure 5.5 shows the transformed census image of book sequence. Final cost construction $C(pl, d)$ performed with C_{TAD} and $C_{census}(pl, d)$ as the shown in equation (4.9):

5.2.3 Gaussian adaptive support weights

We have decided the support region and constructed the initial cost $C(pl, d)$. The support region has been defined and initial cost has been computed. For the accurate matching result, it should aggregate the cost of these support pixels to the central pixel pl with adaptive weights. Motivated by Gestalt theory of perceptual grouping, the weight between two pixels is defined as:

$$w(pl, q) = exp(-|I_{pl} - I_q|/\lambda_i - ||pl - q||/\lambda_d) \tag{5.6}$$

where pl is one of central pixels in left image, q is one of neighbor pixels around pl, $|I_{pl} - I_q|$ and $||pl - q||$ is the intensity difference and the Euclidean Distance between pixel pl and pixel q respectively. λ_i and λ_d are the constant parameter that adjust the balance of two kind of weights. We set the default value of $\lambda_I = 17.5$ and $\lambda_d = 5$ as proposed in Yoon's work [106].

Inspired by Richardt's work [108], we rebuild Yoon's method using Gaussian weight with the help of the motion amplitude.

$$w(pl,q) = G_{\sigma_i}(|I_{pl}-I_q|)\cdot G_{\sigma_d}(||pl-q||)\cdot G_{\sigma_m}(|dx(pl)-dx(q)|+|dy(pl)-dy(q)|). \quad (5.7)$$

where σ_i, σ_d and σ_m are constant parameters, and $G_\sigma = exp(\frac{-x^2}{\sigma^2})$. When one pixel located in background or moveless region ($(|dx(pl) - dx(q))| + |dy(pl) - dy(q)|) = 0$, $G_{\sigma_m} = 1$), the equation (5.7) becomes (5.8).

$$w(pl,q) = G_{\sigma_i}(|I_{pl} - I_q|) \cdot G_{\sigma_d}(||pl - q||). \quad (5.8)$$

The weight branch G_{σ_m} will not bring extra noise or unreasonable weight for moveless region in cost aggregation.

So our aggregated cost is now recalculated as:

$$C^{'}(pl,d) = \frac{\sum_{q\in S_p} w(pl,q)\cdot w(pr,q)\cdot C(pl,d)}{\sum_{q\in S_p} w(pl,q)\cdot w(pr,q)}, \quad (5.9)$$

5.2.4 Proposed cost aggregation

We adopt the same cost aggregation as described in section 4.2.2. The conventional adaptive weight method will aggregate cost along horizontal pass and will store the intermediate cost value. Then, the same job in vertical pass will be computed as shown in the Figure 4.5 (b) and 4.5 (c). Instead of aggregating all the pixels in the support region along the horizontal pass, we perform a square step approach. For instance, we use the largest length of support region bras $L = 13$ to illustrate how the aggregation approach works. To cover the specified bras's length $[-13, 13]$, several iterations are repeated in the horizontal pass. Thus, at each iteration i, every pixel aggregates the costs of three pixels: target pixel and pixels with $-s$ and $+s$ offsets ($s = i^2$ is the square step size). The aggregated cost is stored as an intermediate cost value in the location of pixel q.

5.2.5 Disparity determination

After matching cost aggregation, the Winner-Takes-ALL (WTA) is performed for disparities determination among all the disparity assumptions as:

$$D(pl) = \underset{d \in [d_{min}, d_{max}]}{\arg\min} \; C'(pl, d) \tag{5.10}$$

Where $d \in [d_{min}, ..., d_{max}]$ is the set of all possible disparities.

5.3 Video Stereo matching Implementation with Heterogeneous System

The proposed joint-motion based stereo video matching method is very friendly for parallel computing system. The heterogeneous parallel computing with OpenCL depicted in chapter 3 can be presented for the parallelization of the Joint Motion-Based Video Stereo Matching method. This method consists of two parts: Motion Estimation and Stereo Matching. Likewise, we build one heterogeneous parallel computing environment with one CPU and one GPU as coprocessors for our arithmetic data-parallel computing with OpenCL.

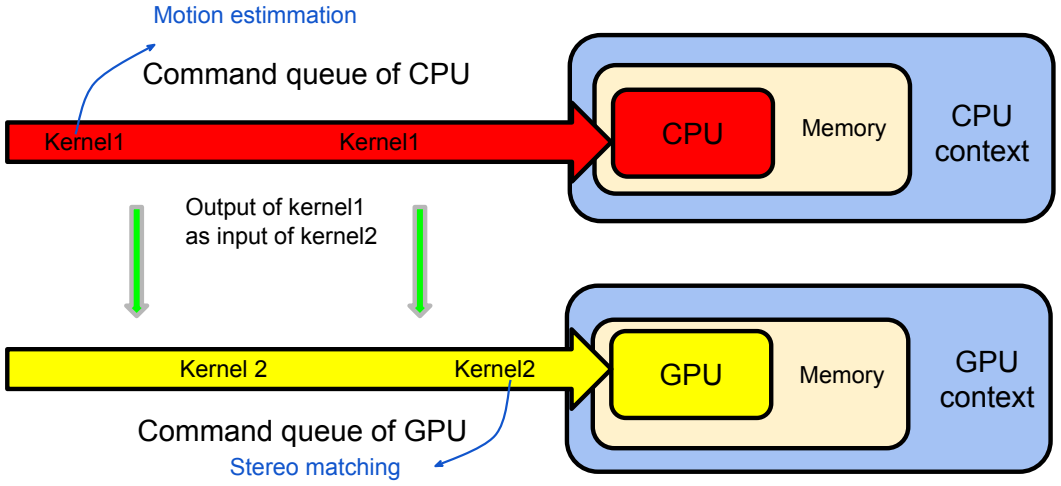


Figure 5.6: Multi-contexts for our heterogeneous computing system

Different with the aforementioned OpenCL context about parallelized motion

estimation, we define the Motion Estimation and Stereo Matching as two separate kernels. Then we assign them on CPU and GPU respectively according to the different workload. As shown in Figure 5.6, *kernel*1 represents the function of Motion Estimation and *kernel*2 represents the function of stereo matching. The time-efficiency results will be introduced in section 5.4.2.

5.4 Evaluation and Discussions

To evaluate the proposed method, we test it on 5 synthetic stereo videos from [108]. The definition of each frame is 400×300 pixel with 64 disparity ranges. These parameters are preset to constant value: $\tau_{gray} = 15$, $\tau_{motion} = 8$, $\gamma = 10$, $\sigma_i = 10$, $\sigma_d = 10$ and $\sigma_m = 5$, $L = 13$ in our implementation.

5.4.1 Matching accuracy

Figure 5.7 shows some frames from the 5 synthetic stereo videos, ground truths and the disparity maps obtained by our method. Our experimental environment is CPU: I7 2630qm and GPU: NVIDIA GT 540m.

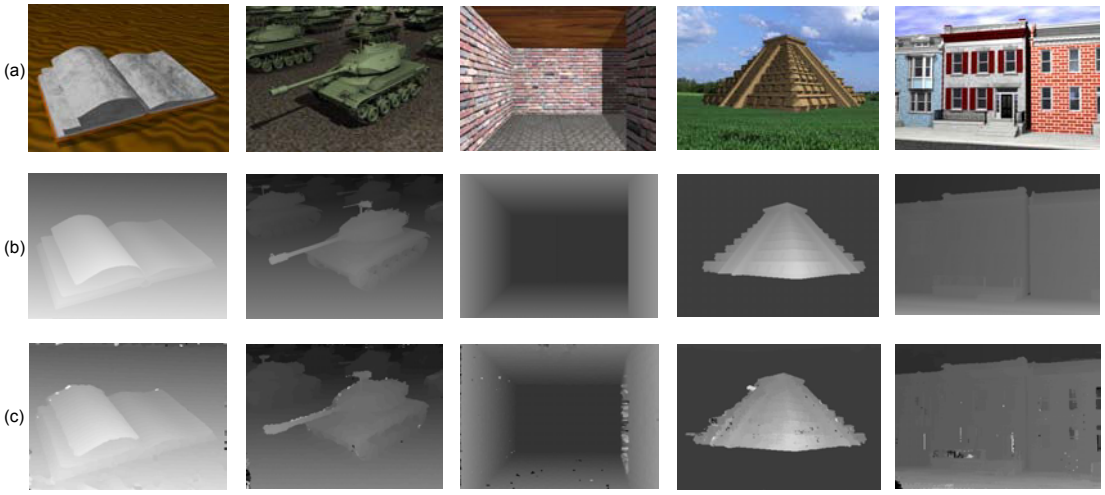


Figure 5.7: Results of stereo video datasets: Book, Tank, Tunnel, Temple and Street [108]. (a) left frame of stereo video, (b) ground truth, (c) disparity result from our method

Likewise, we run our method against LASW [106], HBP-TV [104], TDCB [108] on

all the 5 synthetic video with synthetic ground truth. LASW is the classic adaptive support-weight approach proposed by Yoon [106]. HBP-TV is the Hierarchical Belief Propagation (HBP) based method by setting up anl_1-normed minimization problem with three-dimensional total variation regularization. TDCB is one extension of LASW, which introduce the bilateral filter technique–bilateral grid based on GPU.

The ground truths of stereo videos do not contain any noise, but real videos do. Zero-central Gaussian noise simulated as the thermal imaging noise is added into all color channels of each frame in these 5 synthetic videos. Figure 5.8 gives the visual description of book sequence to illustrate the impact of disparity accuracy under different levels of noise. With the increase of noise intensity, the calculated disparity map is gradually polluted with irregular bad matching pixels. As shown in

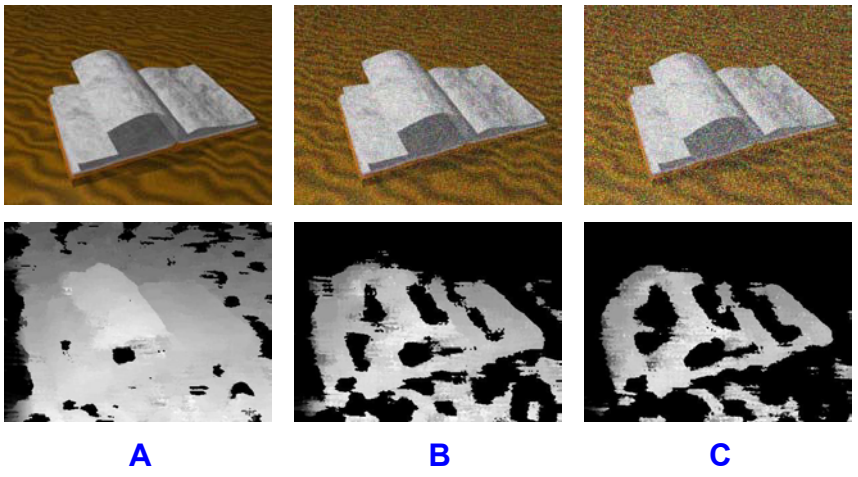


Figure 5.8: The book sequence stereo matching result, A with additive Gaussian noise($\sigma = 20$), B with additive Gaussian noise($\sigma = 60$), C with additive Gaussian noise($\sigma = 100$)

Table 5.1, all these methods are implemented under Gaussian noise with parameter. We can see that the JMSS method performs well at average error rate on the Book, Tanks, and Tunnel test sequences in comparison with the three methods. We can explain that because some explosive motion happened on those three sequences. In contrast, JMSS performs worse than HBP-TV on temple and street because there is less important motion vectors on these two sequences. A confirmation is thus made that the integration of motion vectors (temporal evidence) between adjacent frames brings improvement of accuracy.

Table 5.1: Results comparison of proposed method on synthetic stereo videos with additive Gaussian noise($\sigma = 20$). Shown values are the average and standard deviation of the percentage of bad pixels (threshold is 1).

Method	Book		Tanks		Tunnel		Temple		Street		Time(ms)
	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	
LASW	84.2	**1.24**	56.1	2.67	87.7	2.01	72.8	1.80	58.4	11.7	9770
HBP-TV	26.9	2.82	26.5	**0.16**	29.5	**1.76**	**18.0**	**1.03**	**17.7**	7.60	300
TDCB	44.0	2.02	25.9	2.00	31.4	6.06	31.7	1.82	36.4	7.88	90
Ours	**23.0**	1.37	**18.7**	0.18	**24.6**	8.62	46.4	1.38	21.4	**4.00**	**50**

To verify the robustness of our method, we apply the additive noise σ (range from 0 to 100) into 5 synthetic videos as described in [108]. Plots of the error are shown in Figure 5.9. Our method is superior except "Temple" and "Street" sequence.

5.4.2 Time-efficiency results

Table 5.2: GPUs comparison in our experiments

Devices	**GeForce GT540m**	**Quadro FX 5800**
Architecture	Fermi	GT200GL
Compute Unit (CU)	2	30
Processing Element (PE)	96	240
Core Speed/Mhz	672	648
Memory Speed/Mhz	900	1600
Memory Bus Width/bits	128	512

Our proposed method cost average 1300ms on our CPU platform and 50 ms (CUDA or OpenCL) on GPU platform. As shown in the Table 5.1, our proposed method is the fastest one and reach 20fps. Although the TDCB adopt the more powerful GPU NVIDIA Quadro FX 5800 compared from the Table 5.2, a frame is computed in 90 ms instead of 50 ms forour method. HBP-TV and LASW compute a frame in 300ms and 9770ms respectively.

5.5 Stereo Matching Graphical User Interface (GUI)

Based on our research on stereo matching algorithms, we developed one GUI interface which integrates cameras calibration and stereo matching together. Our GUI is based on Zou's camera calibration framework [109]. Based on their work, we reorganize the GUI into three function areas: *(a) display region*, *(b) camera calibration*,

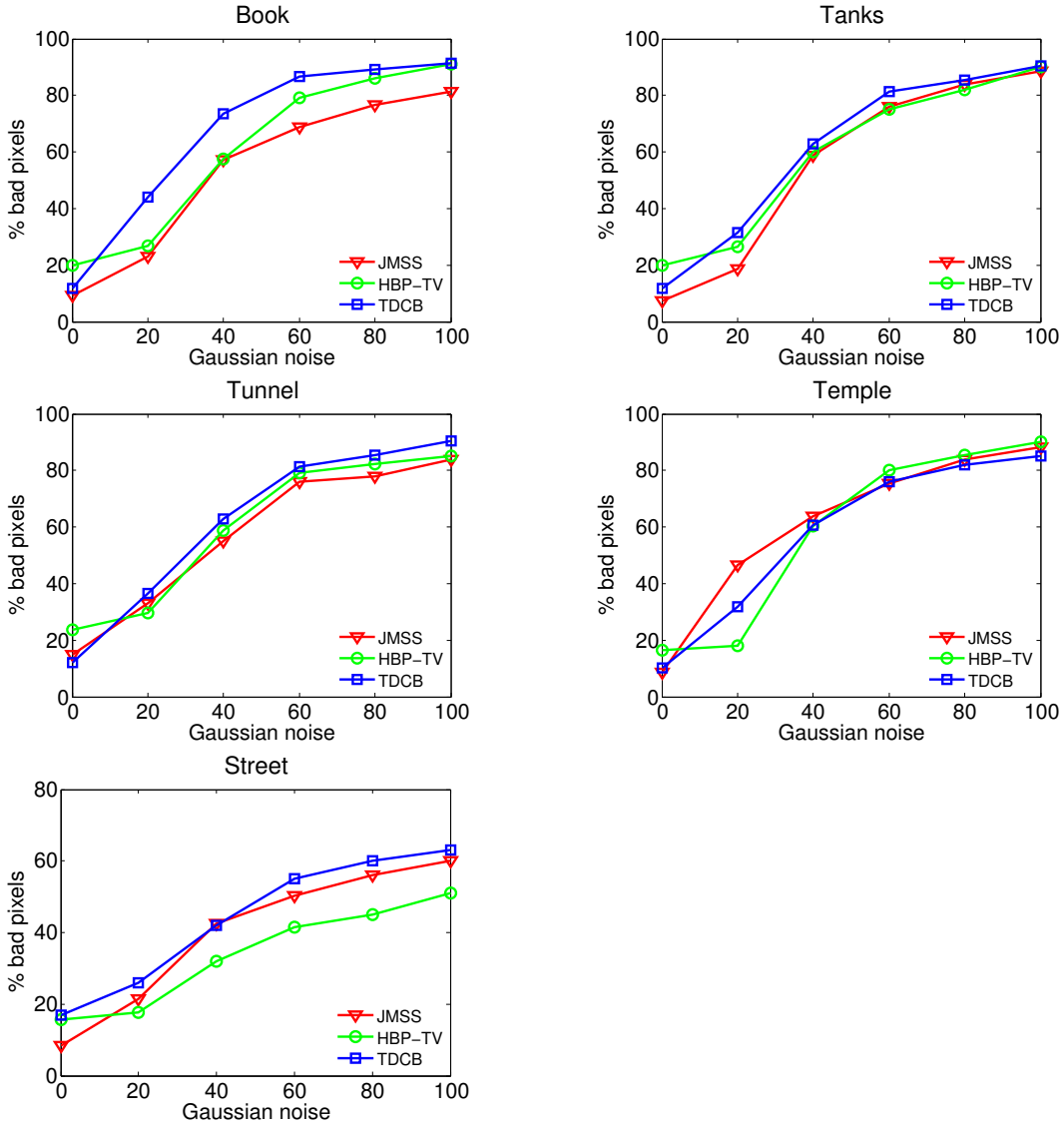


Figure 5.9: Error versus curves for increasing Gaussian noise.

and *(c) stereo matching* as shown in Figure 5.11. In the *display area*, there are three windows to show the left view, right view and disparity respectively. In the *camera calibration area*, there are some basic configuration for camera calibration which are over the scope of this thesis. In the *stereo matching area*, we can choose the four stereo matching algorithms: 1) Jonit Motion Square Step method (JMSS) proposed in this chapter, 2) Square Step Local method (SSL) proposed in the chapter 4, 3) LASW [106] and 4) SGBM method [101]. For the specified method, we can choose whether we use the GPU computing or not, also the way to show the disparity with color or gray.

The Stereo Vision GUI can also capture some real world examples of disparity

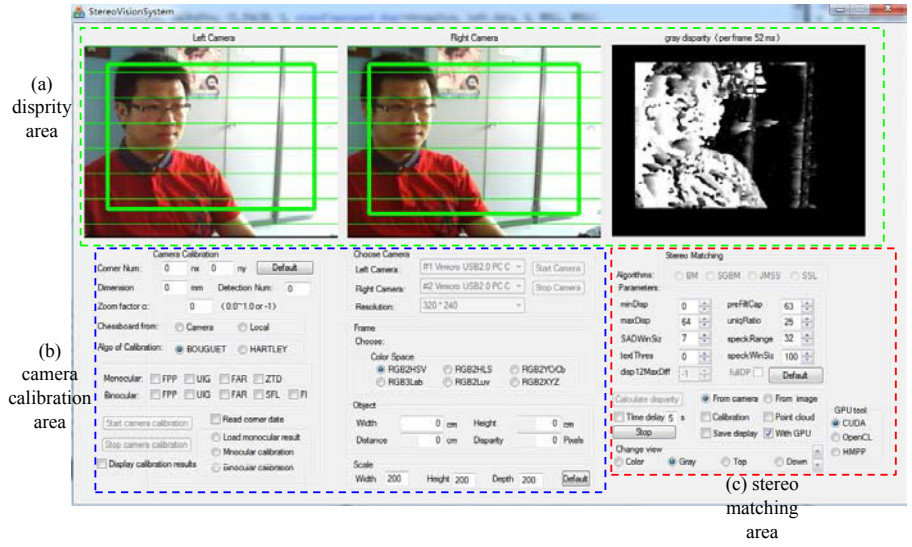


Figure 5.10: Our stereo matching GUI interface

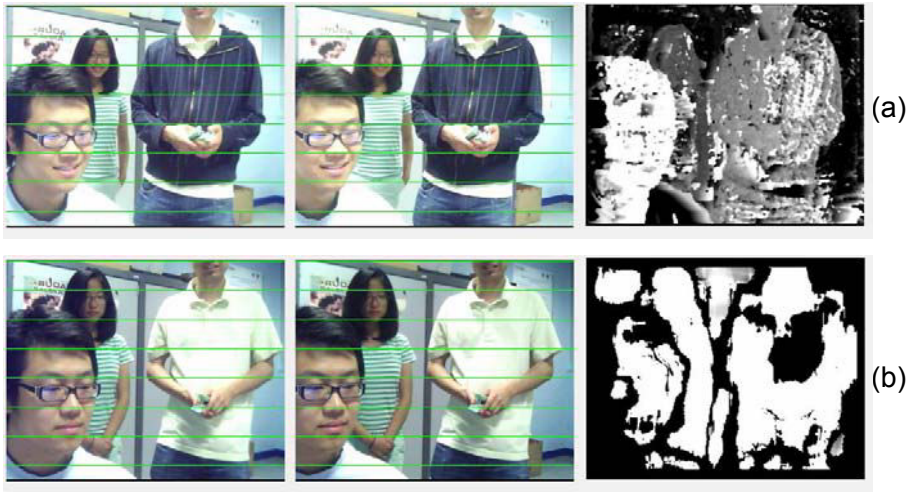


Figure 5.11: Real world disparity, (a) our proposed method and (b) LASW

as shown in Figure 5.11. It can be observed that there is a little difference on the illumination condition between the two views (the right view is brighter than the left view). Andorko et al [102] has discussed the impact of light conditions and light temperatures [110] in building their stereo test scene. According to their research and experiments, the extra noise caused by light conditions will impact the final computed disparity in real world scenes.

Due to the hardware limitation, the illumination condition significantly impacts the accuracy of intensity-based methods, and brings extra noise in the disparity map. As we have discussed in the section 4.3.1, illumination condition significantly

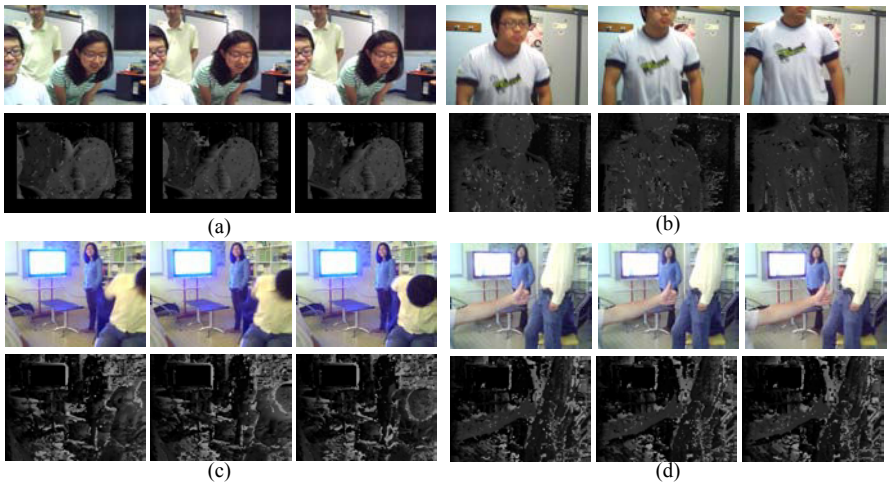


Figure 5.12: Real world disparity examples with fixed stereo camera

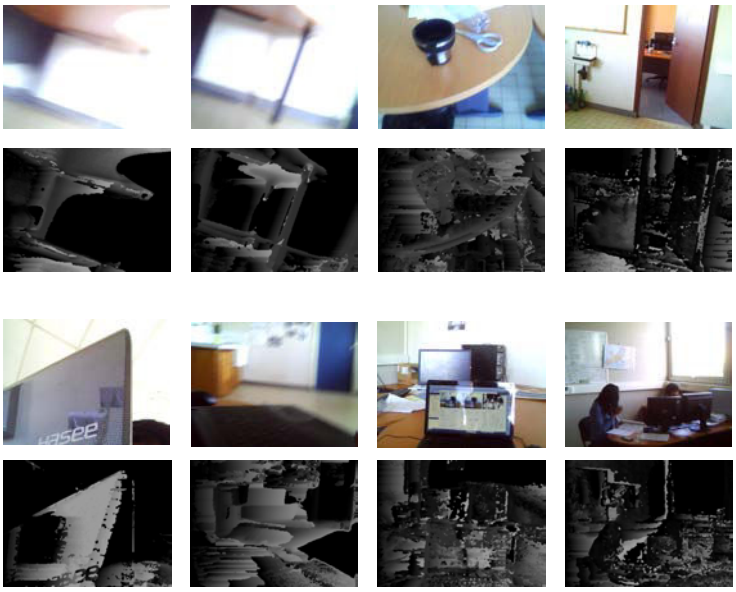


Figure 5.13: Real world disparity examples with moving stereo camera

impacts the accuracy of intensity-based methods, bringing extra noise in disparity map. We have also discussed in the above subsection about the impact of accuracy with Gaussian noise. Similarly with our experiments shown in Figure 5.8 and Figure 4.10, these frames captured from stereo camera are polluted by Gaussian noise and bad illumination condition. So the proposed JMSS method inevitably brings extra and irregular bad matching pixels in real stereo videos. The final results of the proposed method are much better than the result of LASW as shown in Figure 5.11. The proposed method outperforms other method in both synthetic videos and real world videos.

The following additional experiments focus on two conditions: real world disparity with fixed camera and moving camera. Figure 5.12 shows the visual result of the JMSS method in different scenes with fixed stereo camera. Several persons are moving in the scene and there is a strong light source. Even with those difficult real world scenes, the proposed JMSS method provides accurate disparities for the different objects.

In particular, Figure 5.13 illustrates some extrem conditions of real stereo visions such as out of focus (the 1st and 2nd examples in the top part), light reflection (the 1st, 3rd and 4th examples in the bottom part), object distortion and occlusion (camera is too close to the objects, the 1st example in the bottom part) with moving camera like the actual conditions of consumer electronics in daily use. JMSS method can efficiently and accurately obtain the real disparity in spite of the high level of noise and the bad matching points of those extreme conditions.

5.6 Conclusions

In this chapter, we propose a joint motion-based local stereo video matching algorithm named JMSS. Our method first build the support region for selected central pixel with the help of motion vectors, then aggregates raw matching costs with adaptive motion weights, and finally iterates with square step size in support regions along horizontal and vertical pass. We apply the motion vector component into support regions building and adaptive weights to elaborate our approach for the stereo video matching.

Experimental results show that our proposed method outperforms the state-of-the-art stereo video matching methods in test sequences with large amount of noise and high motion vectors. We introduce our graphic user interface of stereo matching which consists of three parts: display region, camera calibration, stereo matching. Some real world examples of disparity are verified by our proposed method which focuses on two conditions: real world disparity with fixed camera and moving camera. Experimental results based on real world video sequences indicate that our method performs better than LASW method. For timing efficiency, our method obtains almost 20 fps in video stereo matching applications with help of GPU computing.

6 Conclusion and Perspective

6.1 Conclusion

Due to both higher accuracy and higher user experience, the most recent image and video processing algorithms become more and more complex. In the meantime, variety of architectures leads to uncertain and unportable designs from a system view of implementation. A system-level view requires algorithm and architecture models. Building a high-level view of a heterogeneous embedded system brings new challenges compared with usual sequential software development chains. This thesis has introduced, analyzed, and studied motion estimation and stereo matching algorithms for heterogeneous systems with a rapid prototyping methodology.

With tools such as Orcc, Preesm, HMPP, our proposed rapid prototyping framework has two branches for many core CPU/GPU and embedded DSP respectively to replace the certain tedious manual development steps. In the two branches of our framework, we make use of the high level description with RCV-CAL to automatically generate the target code for specified architectures.

The background and some fundamental concepts related to rapid prototyping framework and our proposals have been firstly introduced. It also gives the reader the necessary knowledge on the theoretical and practical aspects of parallel embedded computing, rapid prototyping methodology and image and video processing algorithms. We have presented the parallel embedded computing environment which contains parallel embedded systems, parallel programming and rapid prototyping framework. Based on different target architectures, we introduced the basic concept

of motion estimation and stereo matching algorithms which are two applications widely used in the industry and research work.

The parallelized motion estimation targets heterogeneous computing systems which contain CPU and GPU. Our method has the better speed-up ratio comparing with other GPU-based implementations of ME due to our optimizations like shared memory and vectorization. A new way to rebuild the full search ME with GPU can be applied easily in next video encoder like HEVC. The experimental results show that, our implementation has better performance than selected GPU-based FSME implementations. It also obtains a better balance between time efficiency and PSNR than the state-of-the-art fast ME algorithms. Additionally, experimental results show that we found the accurate method to distribute the workload in video applications based on heterogeneous computing system. The full search ME is firstly described as a block diagram with RVC-CAL. Then our prototyping framework generates automatically the OpenCL/CUDA code for our target. In the meantime, we proposed a stereo matching algorithm which adopts combined costs and costs aggregation with square size step to reach real-time performance on laptop's GPUs. Experimental results show that our method outperforms other the state-of-the-art methods providing the best tradeoff between matching accuracy and time-efficiency. Two high-level descriptions of our stereo matching method are introduced based on our rapid prototyping framework.

Based on our research of motion estimation and stereo matching, we have proposed the joint motion-based square step (JMSS) video stereo matching method. Our method first builds the support region for selected central pixel with the help of motion vectors, then aggregates raw matching cost with adaptive motion weights. It then iterates with square step size in support region along the horizontal and vertical pass. We applied adaptive weights to elaborate our approach for the stereo video matching. Our experimental results show that our method outperforms usual stereo video matching methods in video sequences with abundant movement even in noise context. In the end, one stereo matching GUI was proposed, to compare our method with LASW, SGBM and SSL methods. From the system level design view, our proposed image and video applications are rebuilt from the high level descrip-

tions to executable code on heterogeneous systems through the rapid prototyping framework. The design of an applications design the rapid prototyping framework is really faster and easier, reducing repetitive works on different platforms. All these results confirmed our contributions of several proposals.

6.2 Perspective

Based on our proposed motion estimation and video stereo matching method, future works focus on following aspects:

1. Based on the research work of chapter 3, we plan to integrate our parallelized motion estimation into video encoder like x.264 or HEVC, and to evaulate the time efficiency of video encoder with several video resolutions.
2. Another area of future work will concerns our research work of video stereo matching in chapter 4 and chapter 5. Based on the stereo cameras of smart phone and tablet, we plan to calculate the depth map in real-world scenes for developing some electronic consumer applications such as blind-aide, route-planning, 3D reconstruction and so on.
3. Although improving the time efficiency of stereo matching applications is important, it is equally eventful to improve the matching accuracy. We plan to optimize our stereo matching method with several post processing methods like scanline optimization, interpolation for depth discontinuity, sub-pixel enhancement and so on. With a better matching accuracy, we can compute 3D reconstruction for 3D printers and visualization with obtained depth information.
4. With our rapid prototyping methodology, we aim to develop one new backend of Orcc. It generates the executable code of image and video applications to fully exploit the parallel computing of SoC chips. It will be also designed to balance the workload on embedded system.

A

Résumé étendu en français

Chapitre 1 : Introduction

1.1 Contexte

La précision et les performances sont deux critères majeurs pour les applications de traitement d'image et de vidéo. La qualité peut notamment être améliorée en appliquant des algorithmes plus sophistiqués, mais qui s'avèrent aussi plus gourmands en puissance de calcul. Un compromis entre les deux critères peut être obtenu à partir de révolutions technologiques dans les domaines des systèmes embarqués hétérogènes et des langages de programmation parallèle. Les entreprises s'orientent ainsi vers des solutions hétérogènes, combinant des sous-systèmes optimisés pour exécuter des charges de calcul différentes : processeurs généraux (CPU), composants matériels programmables (FPGA), processeurs graphiques (GPU),...

De fait, le GPU ordinairement utilisé pour les calculs graphiques peut être employé pour réaliser des calculs généraux. Quelques tentatives ont ainsi réussi pour des systèmes hétérogènes, même si elles s'avèrent moins matures et bien plus délicates à développer et à utiliser. La puissance de traitement d'un GPU peut donc impacter l'exécution des algorithmes conçus pour exploiter efficacement le parallélisme disponible : c'est typiquement le champ de recherche GPGPU (General-purpose Processing on GPU), appliqué à des domaines aussi divers que le traitement d'images médicales, l'intelligence artificielle, ou la modélisation financière.

Développer efficacement des solutions et des applications sur de telles architec-

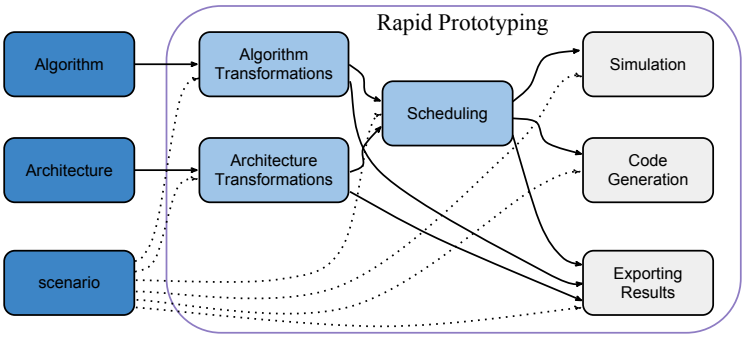


FIGURE A.1 – Environnement générique de prototypage rapide.

tures est un véritable challenge. A cet effet, les fabricants de systèmes hétérogènes proposent leurs propres langages et environnement de développement : AMD et Brook (Brook GPU), NVIDIA et CUDA (Compute Unified Device Architecture), IBM et UPC (Unified Parallel C),... Mais si l'environnement et le langage proposé par un fabricant sont adaptés aux systèmes dudit fabricant, ils ne s'avèrent pas ou peu compatibles avec d'autres systèmes. Pour implémenter plus globalement sur ces systèmes et pour différents niveaux de parallélisme, Apple, NVIDIA, AMD et IBM proposent depuis 2009 le langage OpenCL (Open Compute Language) : ce standard présente notamment une même interface de programmation (API), pour développer un système ouvert combinant ainsi divers composants et pour répartir la charge globale de travail sur ces composants.

La mise en oeuvre d'applications complexes sur ces plateformes complexes est une tâche complexe. Une réponse possible est une approche basée modèles : les caractéristiques intrinsèques de la plateforme matérielle et de l'application doivent alors être décrites par des représentations formelles telles que ressources de traitement et de communications pour la plateforme et modèle(s) de calcul pour le comportement de l'application (Kahn Process Network, Synchronous Dataflow, etc.). Mais actuellement, et en dépit de travaux préliminaires réalisés dans ce sens, il n'y a pas de réelle approche basée modèles qui prend en considération l'adaptation temps réel, tant du point de vue matériel que logiciel...

Une méthodologie de prototypage est un processus de développement logiciel qui permet aux développeurs de créer tout ou partie de solution, afin d'en démontrer la fonctionnalité et d'y apporter les améliorations nécessaires avant d'élaborer la

solution finale. La figure A.1 est une vue générale d'un environnement de prototypage rapide. L'objectif d'un tel environnement est de proposer des modèles génériques de systèmes embarqués multiprocesseurs adaptables. De nombreuses tentatives et travaux de recherche sont notamment orientés vers cet objectif. Succinctement, notre environnement de prototypage rapide s'appuie sur des modèles flot de données, sur un modèle de transformations de modèle, sur une interface utilisateur graphique (GUI) conviviale, et sur un back-end dédié à la génération de code.

1.2 Contributions

Ainsi, les objectifs de cette thèse sont d'évaluer et d'améliorer les méthodologies de prototypage sur systèmes embarqués, notamment celles fondées sur une modélisation flot de données (description haut-niveau des algorithmes), et sur le langage OpenCL (modélisation de niveau intermédiaire). La première contribution de cette thèse est la participation au développement de l'environnement de travail (framework), issu du projet COMPA (Conception Orientée Modèle de calcul pour multi-Processeurs Adaptables). Illustrée par la figure A.2, la méthodologie de prototypage rapide proposée est principalement constituée des outils Orcc [1] [2], Preesm [3] et HMPP [4]. A partir d'une description haut-niveau d'une application, ces trois outils permettent de générer et vérifier du code C ou CUDA ou OpenCL [5] [6] [7], pour des plateformes hétérogènes multi-CPU et GPU. Notre contribution au projet COMPA fut de modéliser dans le langage haut-niveau RVC-CAL [1], des algorithmes d'estimation de mouvement et d'appariement stéréo, afin de valider et vérifier la génération de code issue des outils Orcc et HMPP.

La deuxième contribution de cette thèse est le développement des trois algorithmes de traitement d'image et de vidéo suivants :

- L'estimation de mouvement (cf. chapitre 3) : cet algorithme a été parallélisé, puis implémenté pour un système hétérogène constitué d'un CPU et d'un GPU. De cette étude a été extraite une méthode permettant d'équilibrer la répartition des charges de travail sur un système hétérogène.
- L'appariement stéréo (cf. chapitre 4) : cet algorithme temps réel a été développé, puis étudié pour cibler les plateformes portables GPU d'entrée de gamme.

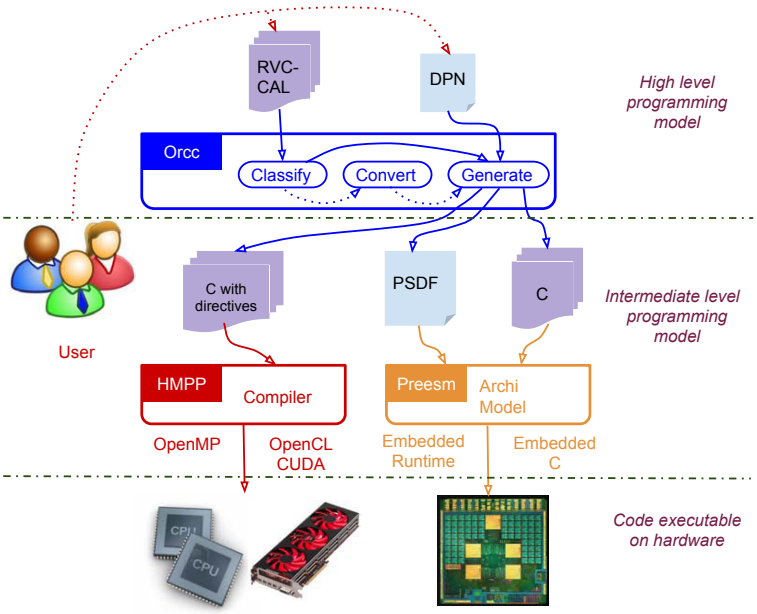


FIGURE A.2 – Proposé méthodologie de prototypage dans le projet COMPA

L'algorithme combine deux techniques de coût et agrège les coûts par un pas d'itération carré. Les résultats expérimentaux montrent que la méthode présente de meilleures performances que les méthodes de base.

– L'appariement stéréo basé mouvement (cf. chapitre 5) : cet algorithme utilise les vecteurs de mouvement obtenus par le premier algorithme pour élaborer la région d'étude de l'image sur laquelle est appliqué le second algorithme. Ce dernier traite alors les images vidéo sous la forme d'images statiques appariées. Les résultats expérimentaux montrent que cette méthode présente de bonnes performances pour les séquences riches en mouvement, même fortement bruitées.

1.3 Structuration du mémoire

Ce mémoire de thèse est structuré comme suit : Le chapitre 2 introduit globalement le contexte de travail, soient les architectures ciblées, les différents modèles de programmation, l'environnement de prototypage rapide et les bases sur les applications développées. Les chapitres suivants explicitent ensuite les trois algorithmes précédemment cités : le chapitre 3 pour l'estimation de mouvement, le chapitre 4 pour l'appariement stéréo et le chapitre 5 pour l'appariement stéréo basé mouvement. Finalement, le chapitre 6 présente la conclusion du mémoire.

Chapitre 2 : Contexte et environnement de travail

Ce chapitre donne un aperçu de notre méthodologie de prototypage rapide. L'approche est globalement ascendante, du matériel jusqu'au domaine applicatif. Elle décrit ainsi les systèmes embarqués parallèles, les modèles de programmation parallèle, les outils de la méthodologie et finalement les concepts de base des applications ciblées.

Les différents systèmes embarqués sont ainsi discutés dans la section 2.1. Le modèle de programmation OpenCL est présenté dans la section 2.2 : il est le modèle de niveau intermédiaire dans le cadre de notre méthodologie. L'approche flot de données est ensuite présentée dans la section 2.3 : elle est le modèle de haut-niveau de notre méthodologie. Trois outils de la méthodologie sont ensuite introduits dans la section 2.4. Finalement, dans la section 2.5, les bases des applications développées sont précisées : estimation de mouvement et appariement stéréo. Une brève conclusion est effectuée dans la section 2.6.

2.1 Architectures cibles

De nombreuses architectures telles que les CPU, GPU, FPGA et DSP posent des problèmes en termes de partitionnement de l'application, de transfert de données et de synchronisation des tâches. Ces problèmes deviennent de plus en plus complexes et entraînent une perte en temps de développement. La taxonomie de Flynn [11] est connue pour classer les architectures en quatre catégories, illustrées sur la figure A.3. Ces catégories sont répertoriées suivant les flots d'instruction et de données, comme suit : - SISD pour Single Instruction Single Data : un flot d'instruction pour un flot de données ; - SIMD pour Single Instruction Multiple Data : un flot d'instructions pour des flots multiples de données ; - MISD pour Multiple Instruction Single Data : des flots multiples d'instructions pour un flot de données ; - MIMD pour Multiple Instruction Multiple Data : des flots multiples d'instructions pour des flots multiples de données. Les architectures parallèles modernes sont principalement de deux types : les architectures SIMD telles que les GPU et les FPGA, et les architectures MIMD tels que les SoC (System on Chip). Les architectures MIMD sont

en outre, soit à mémoire partagée, soit à mémoire distribuée. Cela se répercute sur le réseau d'interconnexions des processeurs : typiquement basé bus ou hiérarchique pour une structure à mémoire partagée, classiquement hypercube ou maillé pour une structure à mémoire distribuée.

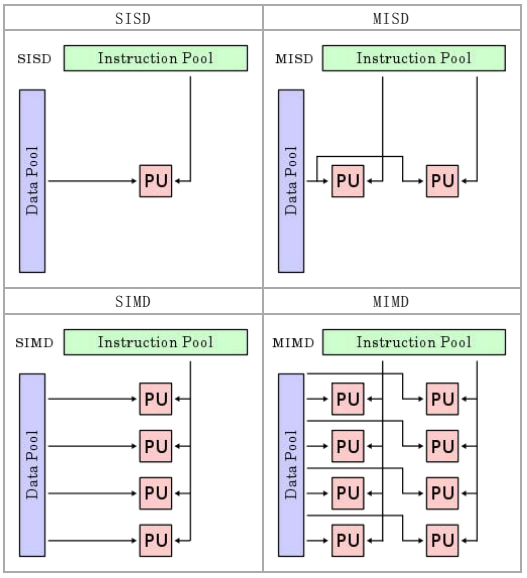


FIGURE A.3 – Taxonomie de Flynn des architectures.(PU - Processing Unit)

Les systèmes classiques mono-processeur appartiennent à la catégorie SISD. Un système SIMD traite plusieurs flots de données par le biais d'un unique flot d'instructions, ce qui décrit typiquement un réseau de processeurs ou une architecture vectorielle. Un système MISD exécute en parallèle plusieurs instructions sur le même flot de données. Les systèmes MIMD sont évidemment les plus complexes.

Les architectures parallèles modernes sont principalement de deux types : les architectures SIMD telles que les GPU et les FPGA, et les architectures MIMD telles que les SoC (System on Chip - système sur puce). Les architectures MIMD sont, en outre, soit à mémoire partagée, soit à mémoire distribuée, ce qui se répercute sur le réseau d'interconnexions des processeurs : typiquement basé bus ou hiérarchique pour une structure à mémoire partagée, hypercube ou maillé pour une structure à mémoire distribuée.

Dans le chapitre 3 de ce mémoire, nous présentons une architecture MIMD hétérogène CPU+GPU à mémoire partagée.

2.1.1 CPU versus GPU

Illustré par la figure A.4, un CPU est classiquement quadri-coeurs ou octo-coeurs, chaque coeur étant programmable, à jeu d'instructions multimédia (SSE par exemple), rapide, cadencé à 2 ou 3 GHz, et à multi-fils d'exécution (threads) : chaque thread peut ou doit supporter alors une forte charge de travail.

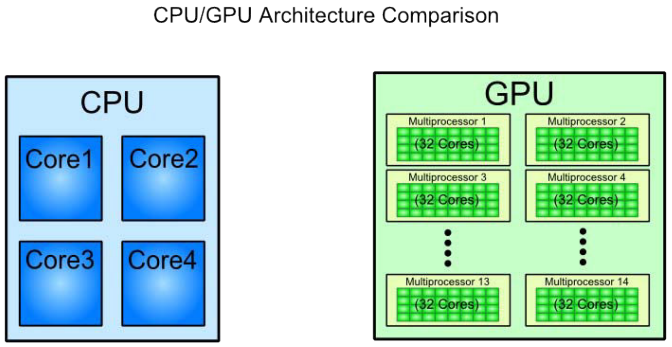


FIGURE A.4 – Comparaison CPU / GPU.

Pour comparaison, un GPU dispose de centaines de coeurs relativement simples, cadencés à 1 GHz typiquement : la charge de travail globale doit alors être partitionnée, parallélisée et répartie en de nombreuses mais légères charges de travail, ce qui peut s'avérer efficace pour des applications à parallélisme élevé. Le fort parallélisme inhérent à une architecture GPU explique en fait la rapidité avec laquelle un tel composant peut effectuer des traitements sur de grandes quantités de données.

2.1.2 Plateforme MPPA de Kalray

La plateforme MPPA (Multi-Purpose Processor Array) de Kalray est multi-coeurs (many-core), et optimisée pour les systèmes embarqués de haute performance et faible consommation d'énergie, soit une solution de traitement idéale pour les applications de faible et moyen volumes. Le composant de base de la famille intègre 256 coeurs, ainsi que des interfaces grande vitesse pour communiquer avec son environnement extérieur (PCIe, Ethernet, DDR). Ce MPPA offre une puissance de plus de 500 GOPS pour une faible consommation d'énergie, ce qui le positionne comme une des solutions les plus efficaces sur le marché de l'électronique professionnelle.

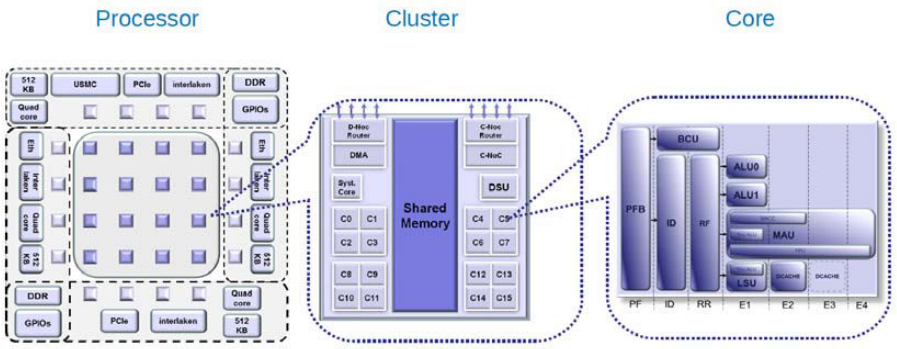


FIGURE A.5 – Structure de la plateforme MPPA.

En fait, comme le montre la figure A.5, le MPPA a par puce 256 coeurs VLIW (Very Long Instruction Word), organisés en 16 clusters de 16 coeurs, interconnectés via un NoC (Network on Chip - réseau sur puce), disposant d'une large bande passante. Chaque cluster dispose également d'une mémoire partagée, chaque coeur du cluster pouvant y accéder directement. En outre, chaque coeur a des entrées/sorties standards pour communiquer avec l'extérieur. Le parallélisme d'une telle plateforme est ainsi très conséquent.

2.2 Le modèle de programmation OpenCL

En fait, puisqu'un système hétérogène est constitué de divers composants tels que les CPU, GPU et SoC, il faut alors pouvoir partitionner l'application, générer de multiples codes cibles et distribuer une charge de travail appropriée par composant.

A cet effet, NVidia propose CUDA, une plateforme et un modèle de programmation pour calcul parallèle, implantable sur les GPU du fabricant. CUDA permet au développeur d'accéder au jeu d'instructions virtuel et à la mémoire des éléments de calcul qui composent un GPU NVidia. Pour comparaison, OpenCL (Open Computing Language) est un standard de langage de programmation parallèle pour les accélérateurs matériels hétérogènes.

OpenCL permet en fait au programmeur de décrire l'application sous la forme de noyau(x) (kernel). Le compilateur parallélise alors, à tous les niveaux possibles, l'exécution des instances du noyau. Comparé au classique langage C séquentiel, OpenCL permet une meilleure utilisation du parallélisme latent de l'architecture

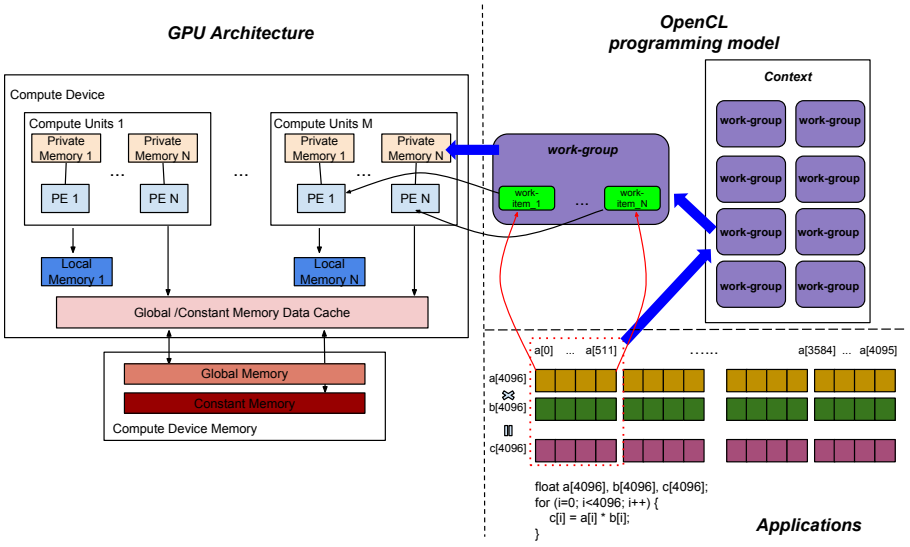


FIGURE A.6 – Architecture GPU et modèle OpenCL.

cible, tout en utilisant une grammaire similaire au langage C.

Dans le cadre de notre méthodologie de prototypage rapide, OpenCL est un modèle de programmation de niveau intermédiaire. Si OpenCL permet la portabilité des applications, il ne garantit pas la portabilité des performances, ce qui peut nécessiter un ajustement complémentaire dans la mise en oeuvre sur une plateforme spécifique ou engendrer des charges de travail dynamiques imprévisibles.

TABLE A.1 – Equivalences GPU - OpenCL.

Devices	**OpenCL**
thread	work-item
threads group	work-group
shared memory	local memory
device memory	global memory

Le tableau A.1 décrit l'équivalence directe établie entre la structure du langage OpenCL et la structure d'un GPU : thread et work-item, threads-group et work-group. Un work-item est l'unité atomique d'un noyau d'exécution OpenCL et tous les work-items d'un work-group exécutent la même instruction simultanément. De même, les mémoires locales et globales d'un contexte OpenCL correspondent respectivement à la mémoire partagée et à la mémoire du composant ciblé. Comme le montre la figure A.6 sur un GPU, chaque work-group s'exécute sur une unité de

calcul (CU - Computing Unit), et chaque work-item s'exécute sur un élément de traitement (PE - Processing Element). La mémoire privée est affectée à un work-item, tandis que la mémoire locale est partagée par tous les work-items d'un même work-group. Les mémoires globale et constante sont accessibles par tous les work-groups.

2.3 Le modèle de programmation flot de données

La complexité introduite par le large éventail des architectures matérielles rend difficile une mise en oeuvre optimale des applications. En outre, la stabilité d'une application doit être prouvé tôt dans le développement, afin d'assurer la fiabilité du produit final. Certains outils comme PeaCE [23] et SynDEx [24] visent à apporter des solutions à ces problèmes par des mesures automatiques menant à un prototype fiable dans un court laps de temps. Une méthodologie de prototypage rapide suit une approche descendante : son objectif est, en partant d'une description de haut niveau d'une application, d'arriver à sa mise en oeuvre en temps réel sur l'architecture cible, aussi automatiquement que possible.

2.3.1 Définition de reconfigurable codage vidéo

Le standard RVC (Reconfigurable Video Coding [25]), définit un ensemble de techniques de codage standard : les unités fonctionnelles (FUs - Functional Units). Les FUs forment la base des standards vidéo actuels et futurs, elles sont normalisées dans la bibliothèque d'outils vidéo VTL (Video Tool Library). Une FU est alors décrite en RVC-CAL, langage portable indépendant de la plateforme (cf. section suivante). Le processus de décodage vidéo est décrit en RVC comme un schéma fonctionnel, aussi nommé réseau de configuration. Ce schéma est constitué de blocs, où les blocs sont des FUs. Pour cela, RVC définit le format FNL (FU Network Language), pour décrire les réseaux. FNL correspond au format XDF, format flot de données XML.

2.3.2 Modèles de calcul flot de données

Un modèle de calcul MoC (Model of Computing), flot de données, définit le comportement d'un programme en le décrivant par un graphe flot de données. Un graphe flot de données est un graphe orienté : les sommets représentent les acteurs, tandis que les arcs représentent les canaux unidirectionnels de communication de type FIFO (First In First Out), à capacité illimitée, connectés aux ports des acteurs. Par exemple, les réseaux d'unités fonctionnelles décrits dans le standard RVC sont des graphes flot de données. Et un graphe flot de données respecte la sémantique d'un réseau DPN (Dataflow Process Network [26]), lié à un réseau KPN (Kahn Process Network [27]), de la manière suivante : 1. Ces modèles contiennent des blocs : processus dans un KPN , acteurs dans un DPN. Ces blocs communiquent par le biais de FIFO unidirectionnelles à capacité illimitée. 2. L'écriture dans une FIFO est non bloquante. 3. Les programmes qui respectent l'un ou l'autre modèle, doivent être programmés dynamiquement en général [28].

2.3.3 Flot de données SigmaC

Pour exploiter le parallélisme latent de la plateforme MPPA, Kalray propose son propre modèle flot de données, le SigmaC. Dans ce modèle, un bloc est un agent, comprenant le code C de la fonction exécutée, le nombre de données entrantes et sortantes via l'interface avec le jeu de production/consommation. Ces informations sont définies dans une syntaxe SigmaC spécifique, illustrée par la figure 2.8 (cf. chapitre 2 du mémoire) ; la figure 2.9 montre la syntaxe d'un sous-graphe interconnectant deux agents.

2.3.4 RVC-CAL Langue

RVC-CAL est un langage SDL (Domain-Specific Language), normalisé par RVC : c'est une version restreinte de CAL [1], inventé par Eker et Janneck.RVC-CAL est un langage SDL (Domain-Specific Language), normalisé par RVC : c'est une version restreinte de CAL [1], inventé par Eker et Janneck.

2.4 La méthodologie de prototypage rapide

La méthodologie de prototypage rapide présentée par la figure A.2 suit une approche descendante et part d'une description haut-niveau en RVC-CAL d'une application, pour se terminer par l'implantation des codes appropriés sur le système hétérogène cible. Cette méthodologie s'appuie principalement sur trois outils : Orcc, Preesm et HMPP.

2.4.1 L'outil de compilation Orcc

Orcc (Open RVC-CAL Compiler [2]) comprend un éditeur textuel RVC-CAL, une infrastructure de compilation, un simulateur et un débogueur. Son atout principal est une infrastructure de compilation permettant de générer plusieurs langages ou combinaison de langages (dans le cas du co-design), à partir des acteurs et réseaux RVC-CAL. Orcc ne génère, ni du code assembleur, ni du code directement exécutable. Il génère en fait un code source qui doit être compilé par un outil adapté. À partir d'une description en RVC-CAL, Orcc peut générer du code pour toute plateforme : matérielle (VHDL), logicielle (C/C++, Java, LLVM,...), ou hétérogène mixte (matérielle et logicielle).

2.4.2 L'outil de prototypage PREESM

Preesm (Parallel and Real-time Embedded Executives Scheduling Method [3]) est un outil de prototypage rapide pour implémentations parallèles. Illustrées par la figure A.1, les entrées de l'outil sont les graphes d'architecture et d'algorithme, ainsi qu'un scénario qui regroupe l'ensemble des informations liant l'algorithme et l'architecture. Les plateformes cibles sont les architectures hétérogènes embarquées, avec possibilité de plusieurs processeurs et de nombreux coeurs. La version actuelle de l'outil s'appuie sur un ordonnancement et un mapping statiques des algorithmes. Les algorithmes sont décrits via un modèle de calcul SDF (Synchronous Dataflow Graph), hiérarchique. Preesm n'est pas adapté pour les applications adaptables (modèle de calcul DPN d'Orcc), pour lesquelles l'ordonnancement et le mapping devraient se faire partiellement ou totalement à l'exécution...

2.4.3 L'outil de compilation HMPP

Développé par CAPS, HMPP (Hybrid Multicore Parallel Programming [4]) est un compilateur basé directives, supportant le standard OpenACC [4]. Il permet le développement d'applications parallèles sur GPU, assurant même le déploiement d'applications parallèles sur des systèmes multi-GPU. HMPP offre un niveau d'abstraction élevé pour une programmation hybride qui tire profit de la puissance de calcul des processeurs avec un modèle de programmation simple. Ce compilateur intègre des back-ends de données parallèles pour CUDA et OpenCL, réduisant ainsi efficacement le temps de développement. Tout en conservant la portabilité et l'interopérabilité du matériel, cet outil permet d'accroitre les performances des applications et la productivité de développement.

HMPP utilise deux directives appariées : *Codelet* et *Callsite*. emph Codelet est utilisée en amont des définitions des fonctions C ; *Callsite* est la directive pour instancier un accélérateur matériel. Par le biais de ces deux directives, HMPP peut remplacer la procédure complexe d'écrire manuellement le code du noyau CUDA ou OpenCL. Pour la transformation, il suffit d'insérer simplement deux lignes dans le code source C (cf. Algorithm 1 chapitre 2).

2.5 Les algorithmes d'image et vidéo utiisés

Certaines applications parallèles sont proposées dans cette thèse d'évaluer le cadre de prototypage rapide qui peut pleinement utilise du parallélisme des architectures cibles. Ils sont beaucoup plus appropriés pour évaluer l'efficacité du cadre de prototypage rapide.

Deux algorithmes de traitement d'image et vidéo sont principalement utilisés dans ce travail de thèse : le premier est l'estimation de mouvement (ME - Motion Estimation), le second est l'appariement stéréo (SM - Stereo Matching).

– L'estimation de mouvement : l'algorithme ME détermine les vecteurs de mouvement qui décrivent la transformation d'une image 2D en une autre, généralement à partir des trames adjacentes de la séquence vidéo. Dans un encodeur, l'estimation de mouvement est la tâche la plus gourmande en temps de calcul (plus de la moitié de la charge globale). Pour réduire la redondance temporelle,

l'algorithme ME pourrait déterminer les mouvements relatifs entre deux images, comme indiqué sur la figure A.7.

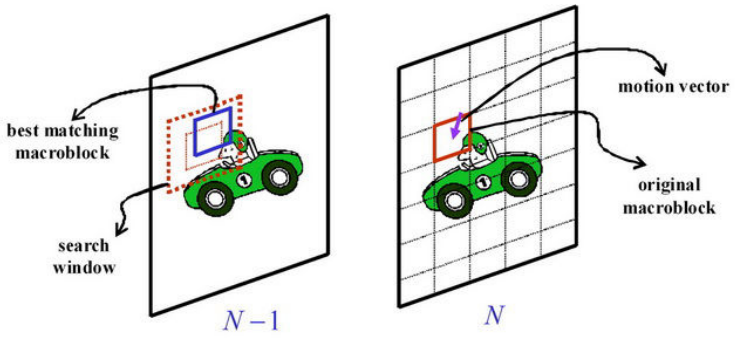


FIGURE A.7 – Illustration de l'estimation de mouvement

– L'appariement stéréo : l'algorithme SM calcule les informations de profondeur à partir de deux caméras, comme le montre la figure A.8. Les algorithmes SM sont typiquement classés en deux catégories : par une approche globale basée sur les caractéristiques (feature) ou par une approche locale basée surface (area). Un algorithme par approche globale s'appuie sur certaines caractéristiques des images telles que les coins ou les bords, pour les apparier ensuite : la carte de disparité alors obtenue est généralement peu fournie, car seules les disparités liées aux caractéristiques utilisées sont extraites. Un algorithme par approche locale essaie directement d'apparier chaque pixel des images, de manière indépendante : la carte de disparité obtenue peut alors se révéler dense. Dans le chapitre 4, nous présentons un algorithme SM de la seconde catégorie.

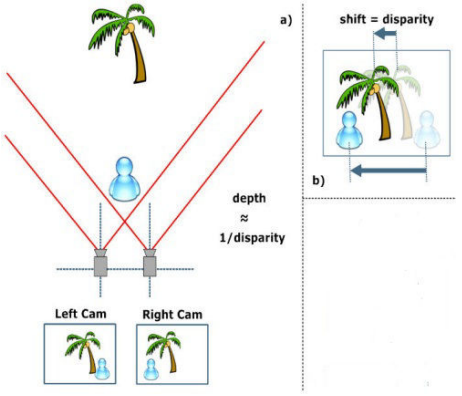


FIGURE A.8 – Aperçu du processus l'appariement stéréo

Chapitre 3 : Estimation de mouvement et système hétérogène

Nous présentons dans ce chapitre notre algorithme d'estimation de mouvement. Pour cibler les architectures hétérogènes, nous avons scindé l'algorithme en deux noyaux OpenCL (kernel) : *kernel_compute* et *kernel_compare*. L'exécution en parallèle des nombreuses instances de ces deux noyaux parallélise l'algorithme et accélère l'estimation de mouvement.

3.1 Le calcul SAD

SAD est la somme des valeurs absolues des différences entre les blocs d'images (cf. équation A.1). Le noyau *kernel_compute* utilise ce critère pour sélectionner les meilleurs vecteurs de mouvement (MV) : 1024 SAD sont ainsi calculés par macrobloc.

$$SAD = \sum_{i=1}^{M}\sum_{j=1}^{M} |block\_current(i,j) - block\_ref(i,j)| \tag{A.1}$$

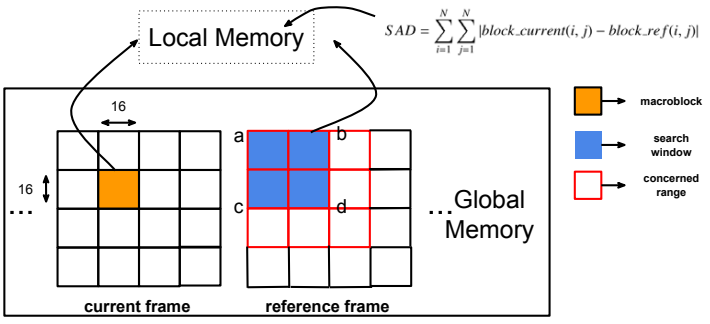


FIGURE A.9 – Proposé calcul de SAD dans la mémoire locale

3.2 Comparaison de SAD

Le noyau *kernel_compare* sélectionne ensuite le meilleur candidat parmi 1024, par le bais de 256 work-items : les 1024 candidats sont préalablement chargés en mémoire locale, puis, chaque work-item détermine le minimum parmi 4 (cf. équation (A.2), i est l'indice de work-item et $stride = 256$). En final, une méthode de

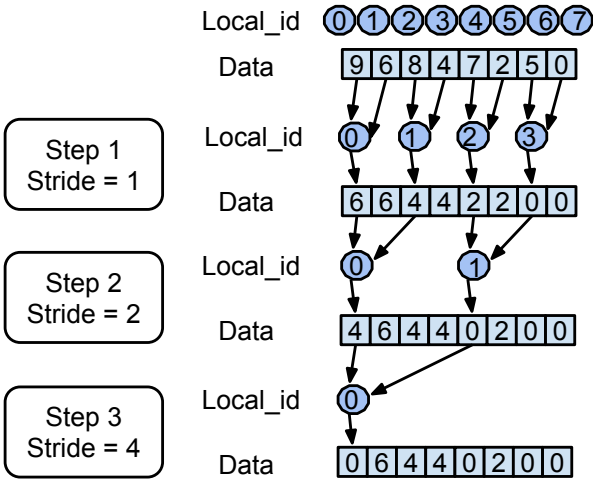


FIGURE A.10 – Proposé comparaison SAD en parallèle

réduction parallèle itérative [62] est appliquée pour déterminer le plus petit vecteur de mouvement parmi 256, comme le montre la figure A.10.

$$cost[i] = Min(Min(cost[i], cost[i+stride]), Min(cost[i+2\times stride], cost[i+3\times stride])) \tag{A.2}$$

3.3 Hétérogène Parallel Computing avec OpenCL

Comme il est difficile de partager un objet de mémoire ou de synchroniser les queues des commandes dans des contextes différents, nous construisons avec OpenCL, un contexte combiné des CPU et GPU, montré dans la figure A.11. Dans ce contexte combiné, le CPU et le GPU ont respectivement leur propre mémoire globale, ce qui évite les transferts de données entre ces deux composants, d'où un gain de temps.

3.4 Description en CAL de l'estimation de mouvement

La figure A.12 est une description flot de données de notre algorithme ME. Le bloc " source " est une FU qui charge la séquence vidéo. Les blocs " ExtractYRef " et " ExtractY " extraient la composante Y de la séquence vidéo codée en YUV, à

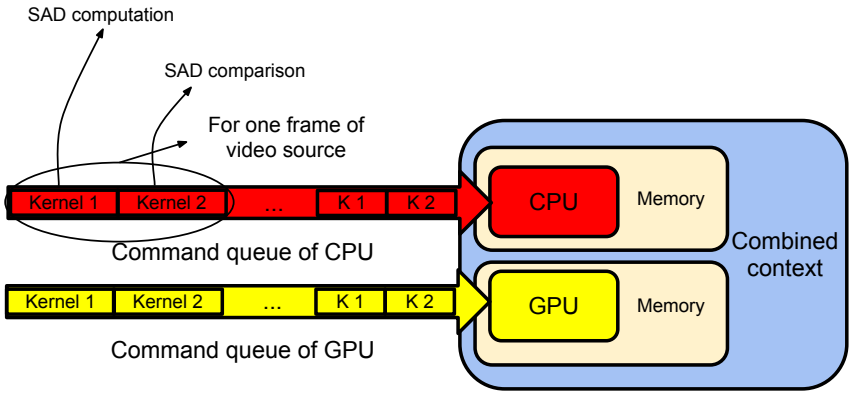


FIGURE A.11 – Combiné contexte de notre système informatique hétérogène

partir des trames de référence et courante respectivement. Le bloc " Recherche_FS " effectue la recherche exhaustive des vecteurs de mouvement et le bloc " ShowVector " affiche les vecteurs de mouvement calculés.

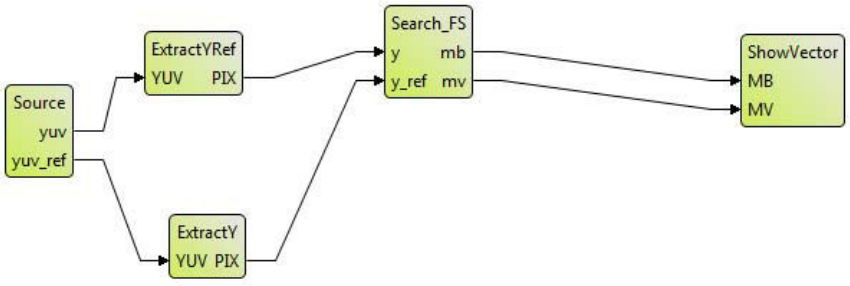


FIGURE A.12 – Estimation de mouvement de schéma SDF

3.5 Évalutation et Conclusion

Dans ce chapitre, nous avons présenté notre algorithme d'estimation de mouvement, décrit en OpenCL. L'algorithme est implémenté sur un système hétérogène combinant un CPU et un GPU. Les résultats expérimentaux montrent alors que notre implémentation a de meilleures performances que les algorithmes ME de recherche exhaustive. Vis-à-vis des algorithmes ME rapides, notre implémentation présente un meilleur équilibre entre l'efficacité temporelle et le rapport signal à bruit. De cette étude, nous avons extrait une méthode précise de répartition de la charge de calcul des applications vidéo sur un système hétérogène, ce qui améliore les performances. En outre, notre environnement de prototypage rapide nous permet de

générer rapidement du code OpenCL ou CUDA.

Chapitre 4 : Méthode locale d'appariement stéréo temps réel

La figure A.13 décrit globalement notre approche d'appariement stéréo :

- Les améliorations apportées impactent principalement les deux premières étapes de la figure A.13(a), soient la construction des coûts et l'agrégation des coûts. La méthode WTA est ensuite appliquée pour produire la disparité finale.
- La figure A.13(b) décrit la mise en oeuvre de notre algorithme sur plateforme GPU.
- La figure A.13(c) montre que notre approche locale identifie chaque pixel à chaque disparité.

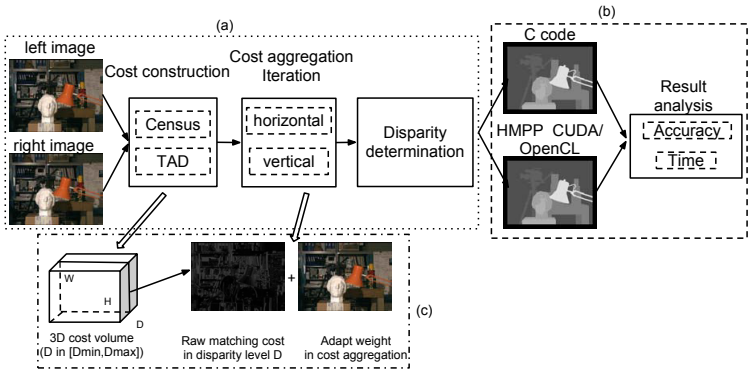


FIGURE A.13 – Proposé système d'appariement stéréo

4.1 Construction des coûts

Le volume des coûts est élaboré par une combinaison du coût TAD $C_{TAD}(pl, d)$ et du coût Census $C_{Census}(pl, d)$:

- Au seuillage σ près, $C_{TAD}(pl, d)$ détermine la valeur absolue de la différence entre l'intensité du niveau de gris du pixel pl de l'image gauche et l'intensité du niveau de gris du pixel pr de l'image droite entachée d'une disparité d (décalée de d).
- $C_{Census}(pl, d)$ détermine la distance de Hamming bit à bit entre les deux chaînes 8-bit des pixels pl et pr. La figure A.14 montre le processus de transformation

du voisinage autour du pixel central en une chaîne 8-bit (fenêtre de voisinage 3×3).

– Ainsi, un 0 est généré lorsque le pixel voisin étudié a une intensité inférieure à l'intensité du pixel central, un 1 sinon. Le coût de construction global $C(pl, d)$ est défini par l'équation (A.4) : il combine deux fonctions robustes, chacune ayant pour paramètre principal l'un des deux coûts présentés ci-avant. Les paramètres λ permettent alors de limiter les valeurs aberrantes.

$$\begin{aligned} C_{TAD}(pl, d) &= min(|i_{left}(pl) - i_{right}(pr)|, \sigma) \\ &= min(|i_{left}(x, y) - i_{right}(x - d, y)|, \sigma), \end{aligned} \tag{A.3}$$

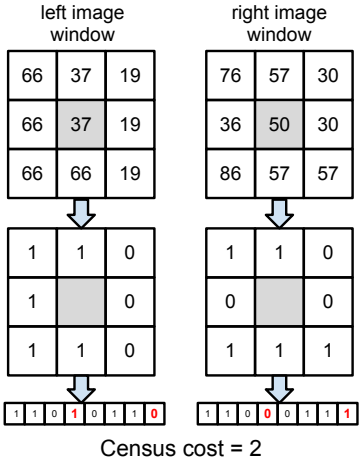


FIGURE A.14 – Exemple la transformation de Census

$$\begin{aligned} C(pl, d) =& \rho(C_{census}(pl, d), \lambda_{census}) \\ &+ \rho(C_{TAD}(pl, d), \lambda_{TAD}), \end{aligned} \tag{A.4}$$

$$\rho(c, \lambda) = 1 - exp(-c/\lambda). \tag{A.5}$$

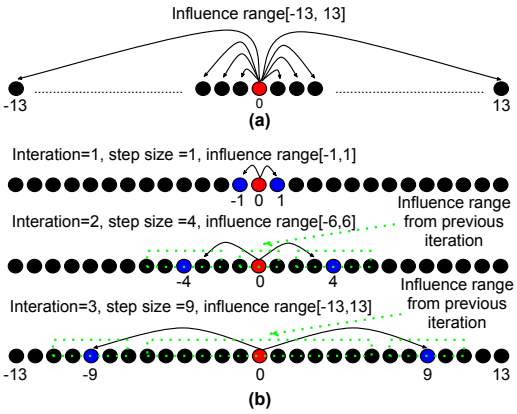


FIGURE A.15 – Exemple d'agrégation des coûts

4.2 Agrégation des coûts

Après construction, le coût doit être agrégé pour produire les disparités finales. Pour cela, nous appliquons une méthode pondérée, itérative et transversale sur l'image : ainsi, les coûts des pixels sont d'abord agrégés suivant un passage horizontal sur l'image (cf. Figure A.15), puis l'approche est réappliquée suivant un passage vertical. En fait, pour couvrir toute la gamme d'influence, plusieurs itérations sont nécessaires dans les deux directions : à chaque itération i, chaque pixel devient alors l'agrégation des coûts du pixel ciblé et de ces deux pixels voisin décalés de $-s$ et $+s$. s est le pas d'itération, qui, dans notre méthode, est le carré de i ($s = i^2$). Ce pas d'itération carré est suffisant pour les résultats à obtenir, tout en limitant la complexité de notre algorithme.

$$\begin{aligned} C(pl,d) += & C(pl+s,d) * w(pl,pl+s) \\ & + C(pl-s,d) * w(pl,pl-s) \end{aligned} \tag{A.6}$$

Figure A.15 illustre notre agrégation dans le sens horizontal (le même processus se répète dans le sens vertical).

4.3 Description en CAL de notre algorithme d'appariement stéréo

La figureA.16 présente la description haut-niveau en CAL de notre algorithme. Ainsi, les blocs "openImageL" et "openImagR" représentent les " unités fonctionnelles " (FU) qui chargent les images appariées gauche et droite ; le bloc "CostConstruction" construit les coûts par rangée ; le bloc "CostAggregation" contient les FUs qui agrègent les coûts dans les sens horizontal et vertical. Finalement, les blocs "select" et "show" détermine la disparité finale et l'affiche.

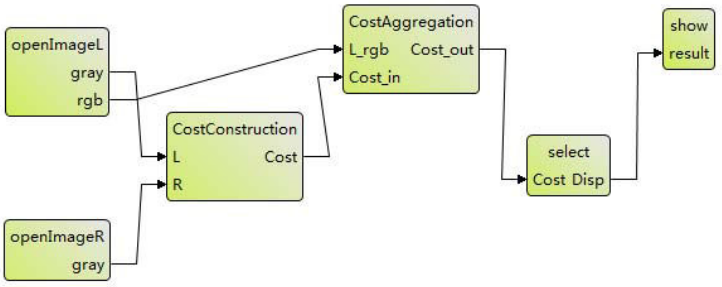


FIGURE A.16 – Flux de données de méthode d'appariement stéréo

Sur la figure 4.8, nous proposons une autre description de l'algorithme, destinée à distribuer la charge globale de travail sur les plateformes multi-DSP. En effet, la description présente une distribution de l'agrégation des coûts sur un nombre n fixe de canaux ($n = 5$ sur la figure) : l'agrégation finale sera alors la moyenne des agrégations de chaque canal (bloc "rebuild" du graphe). Cela correspond en fait au découpage de l'intervalle de disparité en n plus petits intervalles, soit par exemple $[0, 59]$ en 5 intervalles de profondeur 12 (de $[0, 11]$ à $[48, 59]$). Ce parallélisme optimise évidemment l'algorithme.

4.4 Expérimentations et conclusion

Ce chapitre a présenté notre algorithme temps réel d'appariement stéréo. Cet algorithme construit initialement les coûts en combinant les coûts TAD et Census, puis il agrège les coûts par une méthode itérative et transversale sur l'image. L'algorithme applique finalement la méthode WTA (Winner-Takes-All) pour produire les

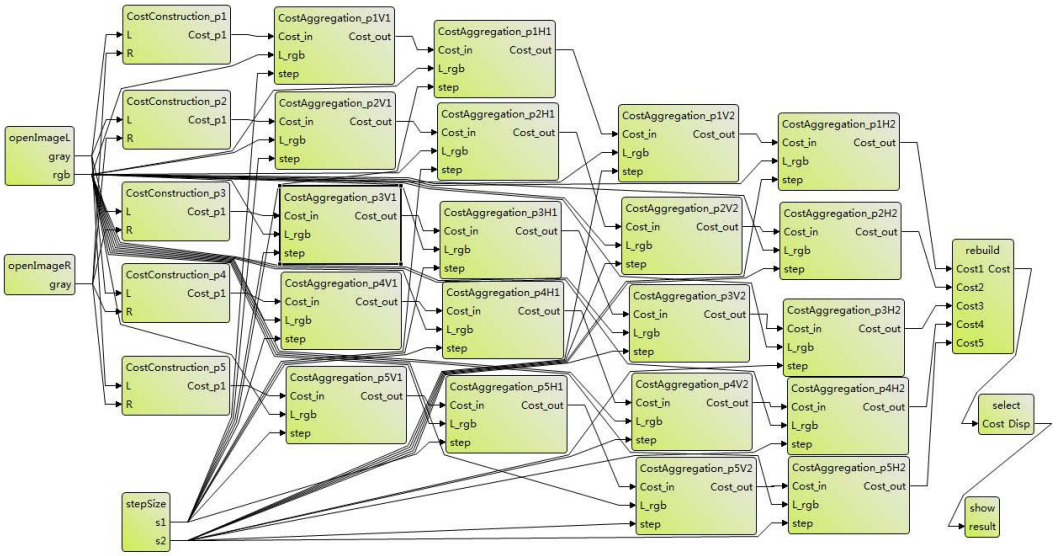


FIGURE A.17 – Flux de données de la méthode d'appariement stéréo parallèle proposé

disparités finales. Nous avons également proposé deux descriptions haut-niveau de notre algorithme d'appariement stéréo : l'une cible les outils Orcc et HMPP, l'autre cible les architectures multi-coeurs. Sous l'environnement de prototypage rapide, le code des noyaux OpenCL ou CUDA est généré automatiquement pour les plateformes GPU. Nos implémentations sur GPU s'avèrent alors 10 fois plus rapides que les implémentations à base de C. En outre, les expérimentations montrent que notre méthode Les résultats expérimentaux montrent que notre méthode délivre une meilleure précision d'appariement que les méthodes Census [43] et ESAW [103], dans les régions plates des images et les discontinuités. Elle surpasse les autres méthodes temps réel dans le compromis entre précision d'appariement et efficacité temporelle.

Chapitre 5 : Appariement basé movements pour la vidéo stéréo

Les méthodes locales d'appariement stéréo utilisent typiquement une région d'étude dans laquelle elles font les mêmes hypothèses : les pixels voisins du pixel étudié ont la même disparité, ou, sans connaissance a priori sur la disparité, ces pixels doivent avoir une intensité similaire. Aussi, trouver une région d'étude appropriée est un facteur clef pour les algorithmes statiques d'appariement stéréo et un facteur important pour les algorithmes d'appariement stéréo vidéo : plus la région d'étude s'avère précise, plus nous pouvons obtenir de précision sur les disparités. A cet effet, nous proposons une méthode qui construit la région d'étude à partir des vecteurs de mouvements, plus précisément à partir des amplitudes de mouvement. Nous appliquons ainsi un algorithme d'estimation de mouvement pour construire cette région, qui sera ensuite utilisée par notre algorithme d'appariement stéréo.

5.1 Construction de la région locale d'étude

Initialement, chaque image I codée en RGB de la séquence vidéo est convertie en niveau de gris i. Nous appliquons ensuite notre algorithme d'estimation de mouvement décrit dans le chapitre 3 pour obtenir des vecteurs de mouvement précis pour chaque bloc $2^n \times 2^n$. La Figure A.18 montre les amplitudes des mouvements en niveau de gris sur les axes x et y, et pour des blocs 8×8 : le blanc représente alors le mouvement nul et plus la couleur tend vers le foncé, plus l'intensité du mouvement est grande.

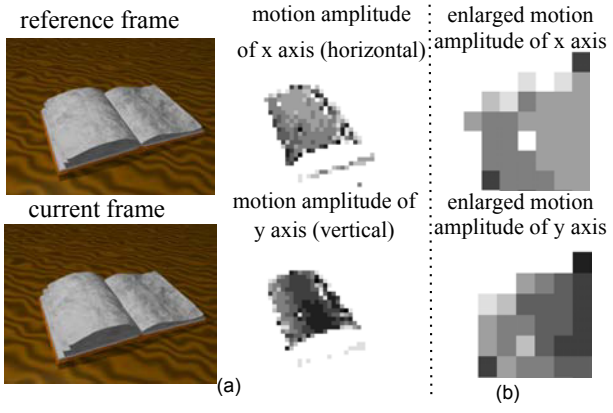


FIGURE A.18 – Amplitudes de mouvement de la séquence de test vidéo stéréo : book.

Nous construisons ensuite la région d'étude S_p sous le forme d'une " croix " adaptative, centrée sur le pixel p étudié, et constituée de bras horizontaux et verticaux (cf. figure A.19). Zhang *et al.* [107] construit une telle croix, puis détermine la longueur des bras à partir de la similarité des couleurs. Dans notre approche, nous émettons l'hypothèse suivante : ***Les blocs dont l'amplitude de mouvement est similaire, doivent avoir une même disparité sur la carte de profondeur***. A partir de cette hypothèse et pour modéliser la région d'étude en croix adaptée au pixel p, nous déterminons la longueur des bras à partir de la similarité des couleurs, comme Zhang, mais en y combinant les amplitudes de mouvement dx et dy sur les axes x et y.

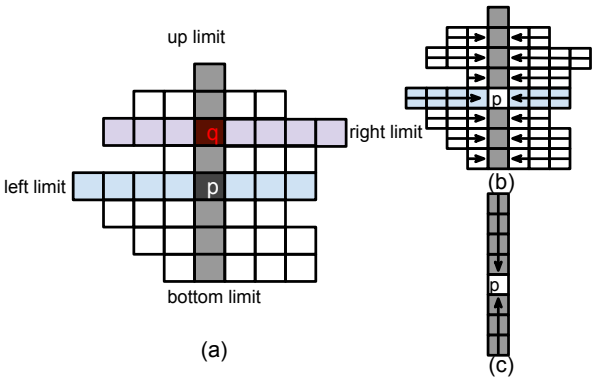


FIGURE A.19 – Zone de support de pixel sélectionné p

$$R = \underset{r\in[1,L]}{max}(r\ \Pi_{i\in[1,r]}\delta(p,p_i)), \tag{A.7}$$

$$\delta(p_1,p_2) = \begin{cases} 1,\ \max_{c\in[R,G,B]}(|I_c(p_1) - I_c(p_2)|) \leq \tau \\ 0,\ otherwise \end{cases}. \tag{A.8}$$

où I_c est l'intensité de la bande de couleur c ($c \in R, G, B$). τ est le seuil de fin de similarité de couleur.

5.2 Détermination de la disparité

L'adaptation de notre algorithme d'appariement stéréo à la région d'étude S_p est simple. Les coûts sont construits, puis agrégés (cf. chapitre 4), et la technique WTA

(Winner-Takes-All) est appliquée pour déterminer les disparités.

$$D(p) = \underset{d \in [d_{min}, d_{max}]}{\arg\min} \; C^{'}(p, d) \tag{A.9}$$

Où $d \in [d_{min}, ..., d_{max}]$ est l'ensemble de toutes les disparités possible.

5.4 Évalutation et Conclusion

Dans ce chapitre, nous avons proposé un algorithme d'appariement stéréo local basé sur le mouvement. Notre méthode construit tout d'abord la région d'étude adaptée et centrée sur le pixel p étudié : cette construction s'effectue par le biais des amplitudes des vecteurs de mouvements, obtenus à partir de notre algorithme d'estimation de mouvement décrit dans le chapitre 3. Notre algorithme d'appariement stéréo décrit dans le chapitre 4 est ensuite appliqué en tenant compte de la région d'étude obtenue. Les résultats expérimentaux montrent que notre approche est performante pour des séquences de test abondantes en mouvement, même fortement bruitées. Nous avons développé une interface graphique d'utilisation, constituée d'une zone d'affichage, de l'étalonnage de la caméra et de l'appariement stéréo. Notre méthode est ainsi testée sur des cas réels, mettant l'accent sur la disparité avec une caméra fixe ou avec une caméra mobile. Nos expérimentations présentent de meilleurs résultats que la méthode LASW de référence, pour un débit d'environ 20 fps (frame per second) grâce aux calculs sur GPU.

Chapitre 6 : Conclusion

Une meilleure expérience des utilisateurs et le besoin de toujours plus de précision font que les algorithmes de traitement d'image et vidéo deviennent de plus en plus complexes. En parallèle, la diversité des architectures peut conduire à des mises en oeuvre parfois incertaines et non portables des systèmes. Une approche au niveau système nécessite de modéliser les algorithmes et les architectures. Et construire une description haut-niveau d'une application, puis la distribuer sur un système embarqué hétérogène, est un véritable challenge. Cette thèse a ainsi analysé, décrit, implémenté et étudié les algorithmes d'estimation de mouvement et d'appariement stéréo, par le biais d'une méthodologie de prototypage rapide ; des descriptions haut-niveau en RVC-CAL des algorithmes ont été développées pour générer automatiquement le code approprié à l'architecture du système ciblé. Le contexte et les concepts généraux du prototypage rapide et des algorithmes déjà cités ont été initialement introduits dans le chapitre 2. Nous avons ainsi présenté les modèles de programmation parallèle, l'environnement de prototypage rapide axé sur trois outils, ainsi que les bases des deux algorithmes étudiés.

Puis, nous avons fait trois propositions. La première proposition décrite dans le chapitre 3 est la parallèlisation de l'algorithme d'estimation de mouvement et son implémentation sur une architecture hétérogène CPU+GPU. Par le biais notamment de nos optimisations sur la mémoire partagée et sur la vectorisation, notre algorithme est plus performant que les autres implémentations sur GPU, et présente un meilleur équilibre efficacité temporelle - rapport signal à bruit que les algorithmes d'estimation de mouvement de référence. De cette étude, nous avons extrait une méthode qui répartit précisément la charge de travail de l'application vidéo sur un système hétérogène. A partir de la description en RVC-CAL de l'algorithme et par le biais de l'environnement de prototypage rapide, nous pouvons alors générer le code OpenCl ou CUDA, suivant le système hétérogène ciblé.

La deuxième proposition décrite dans le chapitre 4 est l'amélioration d'un algorithme d'appariement stéréo temps réel, ciblant les plateformes GPU. Cet algorithme combine maintenant les coûts, puis les agrège par une méthode transversale

et itérative via un pas d'itération carré. Notre algorithme est meilleur que les méthodes de référence dans le compromis entre précision et efficacité temporelle. Deux descriptions haut-niveau de l'algorithme ont ainsi été proposées, utilisables par l'environnement de prototypage rapide.

La troisième proposition décrite dans le chapitre 5 est un algorithme d'appariement vidéo stéréo basé sur les mouvements dans la séquence vidéo. Cet algorithme combine les algorithmes étudiés précédemment : les vecteurs de mouvement obtenus par l'algorithme d'estimation de mouvement servent à construire la région d'étude adaptée au pixel étudié par l'algorithme d'appariement stéréo qui agrège alors les coûts par une méthode adaptative pondérée par les mouvements. Notre algorithme est meilleur que les autres méthodes d'appariement stéréo pour des séquences vidéo abondantes en mouvement, même fortement bruitées. Finalement, une interface utilisateur graphique a été développée, intégrant nos différents algorithmes.

List of Figures

Publications

Journal:

[1] **Jinglin ZHANG**, Jean-Francois NEZAN, Jean-Gabriel COUSIN, "Joint Motion-based Support Region Building in Stereo Video Matching"*, IET Electronics Letters, (Under Review).*

[2] **Jinglin ZHANG**, Jean-Francois NEZAN, Jean-Gabriel COUSIN, "Joint Motion Local Stereo Video Matching method for Real-time Disparity Estimation", *IEEE Transaction on Circuits and Systems for Video Technology, (Under Review).*

[3] **Jinglin ZHANG**, Jean-Francois NEZAN, Jean-Gabriel COUSIN, "Parallelized Motion Estimation Based on Prototyping Methodology and Heterogeneous Parallel System", *Journal of Signal Processing Systems, (Under Review).*

[4] Cong Bai, **Jinglin Zhang**, Zhi Liu, Wan-lei Zhao, "K-means based histogram using multi resolution feature vectors for color texture database retrieval", *Multimedia Tools and Applications, (Under Review).*

[5] Weizhi Lu, **Jinglin Zhang**, Kidiyo Kpalma, Joseph Ronsin, "Visual Tracking via Two-Stage Sparse Representation", *Image and Vision Computing, (Under Review).*

Conference:

[6] **Jinglin ZHANG**, Jean-Francois NEZAN, Jean-Gabriel COUSIN, "Implementation of Motion Estimation Based on Heterogeneous Parallel Computing System with OpenCL." In: *High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, pp. 41-45.

[7] **Jinglin ZHANG**, Jean-Francois NEZAN, Jean-Gabriel COUSIN, RAFFIN Erwan, "Implementation of stereo matching using a high level compiler for parallel computing acceleration." In *Proceedings of the 27th Conference on Image and Vision Computing New Zealand ACM,* pp. 279-283.

[8] **Jinglin ZHANG**, Jean-Francois NEZAN, Jean-Gabriel COUSIN, "Real-Time GPU-based Local Stereo Matching Method", *2013 Conference on Design & Architectures for Signal & Image Processing,* pp.209 – 214.

[9] Alexandre M, Jean-Francois N, Daniel M, **Jinglin ZHANG**, "Implementation of a Stereo Matching Algorithm onto a Manycore Embedded System", *IEEE International Symposium on Circuits and Systems 2014*.

Bibliography

[1] J. Eker and J. Janneck. Cal language report. In *Technical Report ERL Technical Memo UCB/ERL M03/48, University of California at Berkeley*, December 2003.

[2] IETR. Orcc. `http://orcc.sourceforge.net/`.

[3] IETR. Preesm. `http://preesm.sourceforge.net/website/`.

[4] CAPS. Hybrid multicore parallel programming (HMPP). `http://www.caps-entreprise.com/technology/hmpp/`.

[5] Jérôme Gorin, Matthieu Wipliez, Françoise Prêteux, and Mickaël Raulet. Llvm-based and scalable mpeg-rvc decoder. *J. Real-Time Image Process.*, 6(1):59–70, 2011.

[6] Ruirui Gu, J.W. Janneck, S.S. Bhattacharyya, M. Raulet, M. Wipliez, and W. Plishker. Exploring the concurrency of an mpeg rvc decoder based on dataflow program analysis. *Circuits and Systems for Video Technology, IEEE Transactions on*, 19(11):1646–1657, 2009.

[7] J.W. Janneck, I.D. Miller, D.B. Parlour, G. Roquier, M. Wipliez, and M. Raulet. Synthesizing hardware from dataflow programs: An mpeg-4 simple profile decoder case study. In *Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on*, pages 287–292, 2008.

[8] Maxime Pelcat, Slaheddine Aridhi, and Jean-Fran?ois Nezan. Optimization of automatically generated multi-core code for the lte rach-pd algorithm. *DASIP 2008*, 2008.

[9] Maxime Pelcat, Jonathan Piat, Matthieu Wipliez, Slaheddine Aridhi, and Jean-François Nezan. An open framework for rapid prototyping of signal processing applications. *EURASIP J. Embedded Syst.*, 2009:11:3–11:3, January 2009.

[10] Jonathan Piat, Shuvra S Bhattacharyya, Maxime Pelcat, Mickael Raulet, et al. Multi-core code generation from interface based hierarchy. In *Conference on Design and Architectures for Signal and Image Processing (DASIP) 2009*, 2009.

[11] M. Flynn. Very high-speed computing systems. *Proceedings of the IEEE*, 54(12):1901–1909, 1966.

[12] multiscale laboratory. Cuda2opencl. `http://www.multiscalelab.org/swan`.

[13] Peng Du, Rick Weber, Piotr Luszczek, and Stanimire Tomov. From cuda to opencl: Towards a performance-portable solution for multi-platform gpu programming. *Parallel Computing*, pages 391 – 407, 2012.

[14] Edoardo Paone, Gianluca Palermo, Vittorio Zaccaria, Cristina Silvano, Diego Melpignano, Germain Haugou, and Thierry Lepley. An exploration methodology for a customizable opencl stereo-matching application targeted to an industrial multi-cluster architecture. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '12, pages 503–512, 2012.

[15] P.O. Jinen, C.S. de La Lama, P. Huerta, and J.H. Takala. Opencl-based design methodology for application-specific processors. In *Embedded Computer Systems (SAMOS), 2010 International Conference on*, pages 223–230, 2010.

[16] STMicroelectronics. Stmicroelectronics p2012. `http://www.st.com/web/en/home.html`.

[17] J. Leskela, J. Nikula, and M. Salmela. Opencl embedded profile prototype in mobile device. In *Signal Processing Systems, 2009. SiPS 2009. IEEE Workshop on*, pages 279–284, 2009.

[18] Mail. Mail opencl sdk. `http://malideveloper.arm.com/develop-for-mali/sdks/mali-opencl-sdk/`.

[19] Qualcomm. snapdragon. http://www.qualcomm.com.au/products/snapdragon.

[20] A. Barak, T. Ben-Nun, E. Levy, and A. Shiloh. A package for opencl based heterogeneous computing on clusters with many gpu devices. In *Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS), 2010 IEEE International Conference on*, pages 1–7, 2010.

[21] Maurice Herlihy. A methodology for implementing highly concurrent data objects. *ACM Trans. Program. Lang. Syst.*, 15(5):745–770, November 1993.

[22] R. Aoki, S. Oikawa, R. Tsuchiyama, and T. Nakamura. Hybrid opencl: Connecting different opencl implementations over network. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 2729–2735, 2010.

[23] Wonyong Sung, Moonwook Oh, Chaeseok Im, and Soonhoi Ha. Demonstration of codesign workflow in peace. In *in Proc. of International Conference of VLSI Circuit, Seoul, Koera*, 1997.

[24] T. Grandpierre and Y. Sorel. From algorithm and architecture specifications to automatic generation of distributed real-time executives: a seamless flow of graphs transformations. In *Formal Methods and Models for Co-Design, 2003. MEMOCODE '03. Proceedings. First ACM and IEEE International Conference on*, pages 123–132, 2003.

[25] M. Mattavelli, I. Amer, and M. Raulet. The reconfigurable video coding standard [standards in a nutshell]. *Signal Processing Magazine, IEEE*, 27(3):159–167, 2010.

[26] E.A. Lee and T.M. Parks. Dataflow process networks. *Proceedings of the IEEE*, 83(5):773–801, 1995.

[27] Gilles Kahn. The semantics of simple language for parallel programming. In *IFIP Congress*, pages 471–475, 1974.

[28] W. Haid, Lars Schor, Kai Huang, I. Bacivarov, and L. Thiele. Efficient execution of kahn process networks on multi-processor systems using protothreads and windowed fifos. In *Embedded Systems for Real-Time Multimedia, 2009. ESTIMedia 2009. IEEE/ACM/IFIP 7th Workshop on*, pages 35–44, 2009.

[29] E.A. Lee and D.G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[30] G. Bilsen, M. Engels, R. Lauwereins, and J.A. Peperstraete. Cyclo-static data flow. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 5, pages 3255–3258 vol.5, 1995.

[31] B. Bhattacharya and S.S. Bhattacharyya. Parameterized dataflow modeling for dsp systems. *Signal Processing, IEEE Transactions on*, 49(10):2408–2421, 2001.

[32] Wei-Nien C. H.264/AVC motion estimation implmentation on Compute Unified Device Architecture (CUDA). In *Multimedia and Expo, 2008 IEEE International Conference on*, pages 697–700, 2008.

[33] C.E. Erdem, G.Z. Karabulut, E. Yanmaz, and E. Anarim. Motion estimation in the frequency domain using fuzzy c-planes clustering. *Image Processing, IEEE Transactions on*, 10(12):1873–1879, 2001.

[34] A.H. Fakih and J. Zelek. A factorized recursive estimation of structure and motion from image velocities. In *Computer and Robot Vision, 2007. CRV '07. Fourth Canadian Conference on*, pages 355–362, 2007.

[35] M. Werlberger, T. Pock, and H. Bischof. Motion estimation with non-local total variation regularization. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2464–2471, 2010.

[36] Sanyam Mehta, Arindam Misra, Ayush Singhal, Praveen Kumar, and Ankush Mittal. A high-performance parallel implementation of sum of absolute differences algorithm for motion estimation using cuda. In *HiPC Conf*, 2010.

[37] Martin S. Fast Motion estimation on Graphics Hardware for H.264 Video Encoding. *Multimedia, IEEE Transactions on*, 11(1):1–10, jan. 2009.

[38] Jiancong Luo, I. Ahmad, Yongfang Liang, and V. Swaminathan. Motion estimation for content adaptive video compression. *Circuits and Systems for Video Technology, IEEE Transactions on*, 18(7):900–909, 2008.

[39] Chuan-Yiu L. Multi-Pass and Frame Parallel Algorithms of Motion Estimation in H.264/AVC for Generic GPU. In *Multimedia and Expo, 2007 IEEE International Conference on*, pages 1603–1606, 2007.

[40] BrianM.H. Li and PhilipH.W. Leong. Serial and parallel fpga-based variable block size motion estimation processors. *Journal of Signal Processing Systems*, 51(1):77–98, 2008.

[41] F.Urban and JF.Nezan. HDS, a real-time multi-DSP motion estimator for MPEG-4 H.264 AVC high definition video encoding. *real-Time Image Processing*, 4(1):23–31, 2009.

[42] Middlebury stereo vision. `http://vision.middlebury.edu/stereo`.

[43] Martin Humenberger, Christian Zinner, Michael Weber, Wilfried Kubinger, and Markus Vincze. A fast stereo matching algorithm suitable for embedded real-time systems. *Computer Vision and Image Understanding*, 114(11):1180 – 1202, 2010.

[44] Qingxiong Yang, Liang Wang, and N. Ahuja. A constant-space belief propagation algorithm for stereo matching. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1458–1465, 2010.

[45] Xueqin Xiang, Mingmin Zhang, Guangxia Li, Yuyong He, and Zhigeng Pan. Real-time stereo matching based on fast belief propagation. *Machine Vision and Applications*, 23(6):1219–1227, 2012.

[46] A. Hosni, M. Bleyer, C. Rhemann, M. Gelautz, and C. Rother. Real-time local stereo matching using guided image filtering. In *Multimedia and Expo (ICME), 2011 IEEE International Conference on*, pages 1–6, 2011.

[47] Ke Zhang, Jiangbo Lu, G. Lafruit, R. Lauwereins, and L. Van Gool. Real-time accurate stereo with bitwise fast voting on cuda. In *Computer Vision*

Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on, pages 794–800, 2009.

[48] Ruigang Yang, Marc Pollefeys, and Sifang Li. Improved real-time stereo on commodity graphics hardware. In *Proceedings of the 2004 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'04) Volume 3 - Volume 03*, CVPRW '04, pages 36–42, 2004.

[49] Yosuke Miyajima and Tsutomu Maruyama. A real-time stereo vision system with fpga. In *Field Programmable Logic and Application*, volume 2778 of *Lecture Notes in Computer Science*, pages 448–457. 2003.

[50] N. Chang, Ting-Min Lin, Tsung-Hsien Tsai, Yu-Cheng Tseng, and Tian-Sheuan Chang. Real-time dsp implementation on local stereo matching. In *Multimedia and Expo, 2007 IEEE International Conference on*, pages 2090–2093, 2007.

[51] Alexis Michael Tourapis. Enhanced predictive zonal search for single and multiple frame motion estimation. In *proceedings of Visual Communications and Image Processing*, pages 1069–79, 2002.

[52] Lai-Man Po and Wing-Chung Ma. A novel four-step search algorithm for fast block motion estimation. *Circuits and Systems for Video Technology, IEEE Transactions on*, 6(3):313–317, 1996.

[53] Yao Nie and Kai-Kuang Ma. Adaptive rood pattern search for fast block-matching motion estimation. *Image Processing, IEEE Transactions on*, 11(12):1442–1449, 2002.

[54] A. Moradi, R. Dianat, S. Kasaei, and M.T.M. Shalmani. Enhanced cross-diamond-hexagonal search algorithms for fast block motion estimation. In *Advanced Video and Signal Based Surveillance, 2005. AVSS 2005. IEEE Conference on*, pages 558–563, 2005.

[55] Chun-Ho Cheung and Lai-Man Po. Novel cross-diamond-hexagonal search algorithms for fast block motion estimation. *Multimedia, IEEE Transactions on*, 7(1):16–22, 2005.

[56] Youngsub Ko, Youngmin Yi, and Soonhoi Ha. An efficient parallel motion estimation algorithm and x264 parallelization in cuda. In *Design and Architectures for Signal and Image Processing (DASIP), 2011 Conference on*, pages 1–8, 2011.

[57] Ronghui Cheng, Eryan Yang, and Ting Liu. Speeding up motion estimation algorithms on cuda technology. In *Microelectronics and Electronics (PrimeAsia), 2010 Asia Pacific Conference on Postgraduate Research in*, pages 93–96, sept. 2010.

[58] S. Grauer-Gray and C. Kambhamettu. Hierarchical belief propagation to reduce search space using cuda for stereo and motion estimation. In *Applications of Computer Vision (WACV), 2009 Workshop on*, pages 1–8, 2009.

[59] Yu-Cheng Lin, Pei-Lun Li, Chin-Hsiang Chang, Chi-Ling Wu, You-Ming Tsao, and Shao-Yi Chien. Multi-pass algorithm of motion estimation in video encoding for generic gpu. In *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, pages 4–8, 2006.

[60] M. Schwalb, R. Ewerth, and B. Freisleben. Fast motion estimation on graphics hardware for h.264 video encoding. *Multimedia, IEEE Transactions on*, 11(1):1–10, 2009.

[61] Chi-Wang Ho, O.C. Au, S.-H.G. Chan, Shu-Kei Yip, and Hoi-Ming Wong. Motion estimation for h.264/avc using programmable graphics hardware. In *Multimedia and Expo, 2006 IEEE International Conference on*, pages 2049–2052, 2006.

[62] NVIDIA. OpenCL Programming for the CUDA Architecture. v2.3.

[63] J.E. Stone. OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in Science Engineering*, 12:66–73, 2010.

[64] Jinglin Zhang, J.-F. Nezan, and J.-G. Cousin. Implementation of motion estimation based on heterogeneous parallel computing system with opencl. In *High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, pages 41–45, june 2012.

[65] Hui Wang, Yunjian Ge, Zhengshi Liu, Huanjin Liu, and Liang Xu. One curve-fit method for the evaluation of the total distortion of sinusoidal signal. In *Information and Automation (ICIA), 2010 IEEE International Conference on*, pages 1076–1081, 2010.

[66] Peijun Zhang, Xiaoxia Li, and Hongjun Wei. Study and application of curve fitting to automatic control. In *BioMedical Information Engineering, 2009. FBIE 2009. International Conference on Future*, pages 473–475, 2009.

[67] H.A. Kamal and M.H. Eassa. Solving curve fitting problems using genetic programming. In *Electrotechnical Conference, 2002. MELECON 2002. 11th Mediterranean*, pages 316–321, 2002.

[68] Jian Sun, Nan-Ning Zheng, and Heung-Yeung Shum. Stereo matching using belief propagation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(7):787–800, 2003.

[69] A. Klaus, M. Sormann, and K. Karner. Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 3, pages 15–18, 2006.

[70] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(11):1222–1239, 2001.

[71] Michael Bleyer and Margrit Gelautz. Graph-cut-based stereo matching using image segmentation with symmetrical treatment of occlusions. *Image Commun.*, 22(2):127–143, February 2007.

[72] S. Birchfield and C. Tomasi. Depth discontinuities by pixel-to-pixel stereo. In *Computer Vision, 1998. Sixth International Conference on*, pages 1073–1080, 1998.

[73] Xun Sun, Xing Mei, shaohui Jiao, Mingcai Zhou, and Haitao Wang. Stereo matching with reliable disparity propagation. In *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2011 International Conference on*, pages 132–139, 2011.

[74] S. Forstmann, Y. Kanou, Jun Ohya, S. Thuering, and A. Schmitt. Real-time stereo by using dynamic programming. In *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW '04. Conference on*, pages 29–29, 2004.

[75] Ramin Zabih and John Woodfill. Non-parametric local transforms for computing visual correspondence. In *Computer Vision ECCV 94*, volume 801 of *Lecture Notes in Computer Science*, pages 151–158, 1994.

[76] Xing Mei, Xun Sun, and Mingcai Zhou. On building an accurate stereo matching system on graphics hardware. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 467 –474, nov. 2011.

[77] Zucheul Lee, R. Khoshabeh, J. Juang, and T.Q. Nguyen. Local stereo matching using motion cue and modified census in video disparity estimation. In *Signal Processing Conference (EUSIPCO), 2012 Proceedings of the 20th European*, pages 1114–1118, 2012.

[78] Cuong Cao Pham and Jae Wook Jeon. Domain transformation-based efficient cost aggregation for local stereo matching. *Circuits and Systems for Video Technology, IEEE Transactions on*, 23(7):1119–1130, 2013.

[79] Leonardo De-Maeztu, Arantxa Villanueva, and Rafael Cabeza. Stereo matching using gradient similarity and locally adaptive support-weight. *Pattern Recognition Letters*, 32(13):1643 – 1651, 2011.

[80] Xiaozhou Zhou and P. Boulanger. Radiometric invariant stereo matching based on relative gradients. In *Image Processing (ICIP), 2012 19th IEEE International Conference on*, pages 2989–2992, 2012.

[81] M. Kaaniche, A. Benazza-Benyahia, B. Pesquet-Popescu, and J. Pesquet. Vector lifting schemes for stereo image coding. *Image Processing, IEEE Transactions on*, 18(11):2463–2475, 2009.

[82] Hoang Trinh and David McAllester. Unsupervised learning of stereo vision with monocular cues. In *Proc. BMVC*, pages 72.1–72.11, 2009.

[83] S. Hermann and T. Vaudrey. The gradient - a powerful and robust cost function for stereo matching. In *Image and Vision Computing New Zealand (IVCNZ), 2010 25th International Conference of*, pages 1–8, 2010.

[84] Peter Pinggera12, Toby Breckon, and Horst Bischof. On cross-spectral stereo matching using dense gradient features. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, 2012.

[85] M. Okutomi and T. Kanade. A multiple-baseline stereo. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 15(4):353–363, 1993.

[86] R.T. Collins. A space-sweep approach to true multi-image matching. In *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR '96, 1996 IEEE Computer Society Conference on*, pages 358–363, 1996.

[87] R. Szeliski and P. Golland. Stereo matching with transparency and matting. In *Computer Vision, 1998. Sixth International Conference on*, 1998.

[88] Andrew Speers and Michael Jenkin. Tuning stereo image matching with stereo video sequence processing. In *Proceedings of the 2012 Joint International Conference on Human-Centered Computer Environments*, HCCE '12, pages 208–214, 2012.

[89] Liang Wang and Ruigang Yang. Global stereo matching leveraged by sparse ground control points. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3033–3040, 2011.

[90] Ke Zhu, Matthias Butenuth, and Pablo d¡¯Angelo. Comparison of dense stereo using cuda. 6554:398–410, 2012.

[91] A.V. Nefian, K. Husmann, M. Broxton, V. To, M. Lundy, and M.D. Hancher. A bayesian formulation for sub-pixel refinement in stereo orbital imagery. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pages 2361–2364, 2009.

[92] Annika Kuhl. Comparison of stereo matching algorithms for mobile robots. *Centre for Intelligent Information Processing System*, pages 4–24, 2005.

[93] M. Bleyer and M. Gelautz. Temporally consistent disparity maps from uncalibrated stereo videos. In *Image and Signal Processing and Analysis, 2009.*

ISPA 2009. Proceedings of 6th International Symposium on, pages 383–387, 2009.

[94] Matan Protter A and Michael Elad A. Sparse and redundant representations and motion-estimation-free algorithm for video denoising. *Proceedings of SPIE, the International Society for Optical Engineering*, 2007.

[95] Mahmoud Ghoniem, Youssef Chahir, and Abderrahim Elmoataz. Video denoising and simplification via discrete regularization on graphs. *Advanced Concepts for Intelligent Vision Systems*, 5259:380–389, 2008.

[96] K. Heath and L. Guibas. Multi-person tracking from sparse 3d trajectories in a camera sensor network. In *Distributed Smart Cameras, 2008. ICDSC 2008. Second ACM/IEEE International Conference on*, pages 1–9, 2008.

[97] Kuk-Jin Yoon and In-So Kweon. Locally adaptive support-weight approach for visual correspondence search. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 924–931, june 2005.

[98] Minglun Gong, Ruigang Yang, and Liang Wang. A performance study on different cost aggregation approaches used in real-time stereo matching. *International Journal of Computer Vision (IJCV)*, 2007.

[99] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vision.*

[100] Shang-Te Yang, Tsung-Kai Lin, and Shao-Yi Chien. Real-time motion estimation for 1080p videos on graphics processing units with shared memory optimization. In *Signal Processing Systems, 2009. SiPS 2009. IEEE Workshop on*, pages 297–302, 2009.

[101] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(2):328–341, 2008.

[102] I. Andorko, P. Corcoran, and A.P. Bigioi. Proposal of a universal test scene for depth map evaluation. *Consumer Electronics, IEEE Transactions on*, 59(2), 2013.

[103] Wei Yu, Tsuhan Chen, and J.C. Hoe. Real time stereo vision using exponential step cost aggregation on gpu. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pages 4281 –4284, nov. 2009.

[104] R. Khoshabeh, S.H. Chan, and T.Q. Nguyen. Spatio-temporal consistency in video disparity estimation. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 885–888, 2011.

[105] S.H. Chan, R. Khoshabeh, K.B. Gibson, P.E. Gill, and T.Q. Nguyen. An augmented lagrangian method for total variation video restoration. *Image Processing, IEEE Transactions on*, 20(11):3097–3111, 2011.

[106] Kuk-Jin Yoon and In-So Kweon. Adaptive support-weight approach for correspondence search. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(4):650–656, 2006.

[107] Ke Zhang, Jiangbo Lu, and G. Lafruit. Cross-based local stereo matching using orthogonal integral images. *Circuits and Systems for Video Technology, IEEE Transactions on*, 19(7):1073–1079, 2009.

[108] Christian Richardt, Douglas Orr, Ian Davies, Antonio Criminisi, and Neil A. Dodgson. Real-time spatiotemporal stereo matching using the dual-cross-bilateral grid. In *Proceedings of the European Conference on Computer Vision (ECCV)*, volume 6313 of *Lecture Notes in Computer Science*, page 510¨C523, September 2010.

[109] Zouyuhua. Stereo calibration framework. `http://my.csdn.net/chenyusiyuan`.

[110] K. Barnard, L. Martin, A. Coath, and B. Funt. A comparison of computational color constancy algorithms. ii. experiments with image data. *Image Processing, IEEE Transactions on*, 11(9):985–996, 2002.

INSA de RENNES
Service des Formations

AVIS DU JURY SUR LA REPRODUCTION DE LA THESE SOUTENUE

Titre de la thèse:
Prototyping methodology of image processing applications on heterogeneous parallel systems

Nom Prénom de l'auteur : ZHANG JINGLIN

Membres du jury :
- Monsieur NEZAN Jean-François
- Monsieur COUSIN Jean-Gabriel
- Monsieur MOREAU Guillaume
- Monsieur PAINDAVOINE Michel
- Monsieur HOUZET Dominique

Président du jury : Michel PAINDAVOINE

Date de la soutenance : 19 Décembre 2013

Reproduction de la these soutenue

☒ Thèse pouvant être reproduite en l'état
☐ Thèse pouvant être reproduite après corrections suggérées

Fait à Rennes, le 19 Décembre 2013

Signature du président de jury

M.P.

Le Directeur,

M'hamed DRISSI

Résumé

Le travail présenté dans cette thèse s'inscrit dans un contexte de manipulation croissante d'images et de vidéo sur des systèmes embarqués parallèles. Les limitations et le manque de flexibilité dans la conception actuelle de ces systèmes font qu'il est de plus en plus compliqué de mettre en œuvre les applications, en particulier lorsque le système est hétérogène. Or, non seulement Open Computing Language (OpenCL) est un nouveau cadre pour utiliser pleinement la capacité de calcul des processeurs généraux ou embarqués, mais, en outre, des outils de prototypage rapide sont disponibles pour la conception des systèmes, leur but étant de générer un prototype fiable ou de mettre en œuvre de manière automatique les applications de traitement d'images et vidéo sur les systèmes embarqués.

L'objectif général de cette thèse est d'évaluer et d'améliorer les processus de conception pour les systèmes embarqués, particulièrement ceux fondés sur des approches flot de données (haut niveau d'abstraction) et OpenCL (niveau intermédiaire d'abstraction). Cet objectif ambitieux fait l'objet de plusieurs projets dont le projet collaboratif COMPA, mettant en œuvre les outils Orcc, Preesm et HMPP. Dans ce cadre, cette thèse vise à valider et évaluer ces outils sur des applications d'estimation de mouvement et d'appariement stéréo. Nous avons ainsi modélisé ces applications dans le langage haut-niveau RVC-CAL. Puis, par le biais des trois outils Orcc, Preesm et HMPP, nous avons généré et vérifié du code C, OpenCL et CUDA, pour des plates-formes hétérogènes CPU multi-cœur et GPU. L'implémentation des algorithmes sur la puce embarquée MPPA multi-cœur (many-core) de la société KALRAY, a été étudiée.

Pour atteindre l'objectif, nous avons proposé trois algorithmes. Le premier est un estimateur de mouvement parallélisé pour un système hétérogène constitué d'un CPU et d'un GPU : pour cette implantation, nous avons développé une méthode qui équilibre la répartition des charges entre CPU et GPU. Le second algorithme est une méthode d'appariement stéréo en temps réel : elle utilise une combinaison de fonctions de coût et une agrégation des coûts par pas d'itération carré ; nos résultats expérimentaux surpassent les autres méthodes en offrant un compromis intéressant entre la complexité de l'algorithme et sa précision. Le troisième algorithme est une méthode d'appariement stéréo basée sur le mouvement : elle utilise les vecteurs de mouvements issus du premier algorithme pour déterminer la région d'étude nécessaire pour le second algorithme ; nos résultats montrent que l'approche est particulièrement efficace lorsque les séquences de test sont riches en mouvement, même bruitées.

Mots-clés : méthodologie de prototypage, système hétérogène, estimation de mouvement, appariement stéréo, OpenCL, HMPP.

Abstract

The work presented in this thesis takes place in a context of growing demand for image and video applications on parallel embedded systems. The limitations and lack of flexibility of current design with parallel embedded systems make increasingly complicated to implement applications, particularly on heterogeneous systems. But Open Computing Language (OpenCL) is a new framework for fully employ the capability of computation of general purpose processors or embedded processors. In the meantime, some rapid prototyping tools to design systems are proposed to generate a reliably prototype or automatically implement the image and video applications on embedded systems.

The goal of this thesis was to evaluate and to improve design processes for embedded systems, especially based on the dataflow approach (high level of abstraction) and OpenCL approach (intermediate level of abstraction). This challenge is tackled by several projects including the collaborative project COMPA which studies a framework based on the Orcc, Preesm and HMPP tools. In this context, this thesis aims to validate and to evaluate the framework with motion estimation and stereo matching algorithms. For this aim, algorithms have been described using the high-level RVC-CAL language. With the help of Orcc, Preesm, and HMPP tools, we generated and verified C code or OpenCL code or CUDA code for heterogeneous platforms based on multi-core CPU and GPU. We also studied the implementations of these algorithms onto the last generation of many-core for embedded system called MPPA and developed by KALRAY.

We proposed three algorithms. One is a parallelized motion estimation method for heterogeneous system based on one CPU and one GPU: we developed one basic method to balance the workload distribution on such heterogeneous system. The second algorithm is a real-time stereo matching method that adopts combined costs and costs aggregation with square size step to implement on laptop's GPU platform: our experimental results outperform other baseline methods about tradeoff between matching accuracy and time-efficiency. The third algorithm is a joint motion-based video stereo matching method that uses the motion vectors calculated by the first algorithm to build the support region for the second algorithm: our experimental results outperform the stereo video matching methods in the test sequences with abundant movement even in large amounts of noise.

Keywords: prototyping methodology, heterogeneous system, motion estimation, stereo matching, OpenCL, HMPP.

N° d'ordre : 13ISAR 35 / D13 - 35
Institut National des Sciences Appliquées de Rennes
20, Avenue des Buttes de Coëmes • CS 70839 • F-35708 Rennes Cedex
Tel : 02 23 23 82 00 - Fax : 02 23 23 83 96