Runtime mapping of dynamic dataflow applications on heterogeneous multiprocessor platforms

Dinh Thanh Ngo

THÈSE / UNIVERSITÉ DE BRETAGNE SUD

sous le sceau de l'Université Européenne de Bretagne

Pour obtenir le titre de
DOCTEUR DE L'UNIVERSITÉ DE BRETAGNE SUD
Mention : STIC
École Doctorale SICMA

présentée par

Thanh Dinh NGO

Préparée à l'Unité Mixte de Recherche n° 6285 Lab-STICC

Laboratoire des Sciences et Techniques de l'Information, de la Communication et de la Connaissance

UFR Sciences et Sciences de l'Ingénieur

Runtime mapping of dynamic dataflow applications on heterogeneous multiprocessor platforms

Thèse soutenue le 19 Juin 2015,
devant la commission d'examen composée de :

Pr. Sébastien PILLEMENT
Professeur, Université de Nantes, IETR Lab / Rapporteur

Dr. Samy MEFTALI
Maître de Conférences, HDR, Université des Sciences et Technologies de Lille / Rapporteur

Pr. Jean-François NEZAN
Professeur, Institut National des Sciences Appliquées de Rennes / Examinateur

Pr. Emmanuel CASSEAU
Professeur, Université de Rennes 1, Lannion / Examinateur

DR. Jean-Philippe DIGUET
Directeur de thèse / Directeur de recherche CNRS

Dr. Kevin MARTIN
Co-directeur de thèse / Maître de conférences

Résumé

La complexité et le nombre toujours plus grandissant des applications, notamment les standards vidéo, nécessite d'étudier des méthodes et outils pour leur déploiement sur des architectures elles aussi toujours plus complexes. En effet, afin d'atteindre les performances requises en matière de temps d'exécution ou consommation énergétique, les architectures modernes proposent des éléments de calculs hétérogènes, où chacun est spécialisé pour une fonction précise. Cette thèse s'appuie sur le modèle flot de données pour la spécification de l'application. Ce modèle permet d'exposer explicitement le parallélisme spatial et temporel de l'application à travers un réseau d'acteurs interconnectés par des canaux de type FIFO. Les acteurs, en charge du calcul, peuvent exhiber un comportement statique ou dynamique. Les derniers standards vidéo contraignent à s'appuyer sur les modèles dynamiques pour obtenir une spécification fonctionnelle. Les besoins de calcul sont alors dépendants des données à traiter. Le déploiement d'une application dynamique ne peut donc se faire à l'aide des approches statiques existantes dans la littérature. L'objectif de cette thèse est de proposer des algorithmes efficaces permettant de déployer à la volée une application flot de données dynamique sur une architecture multiprocesseurs hétérogène. La première contribution est un algorithme qui permet de trouver rapidement une solution de déploiement de l'application. La deuxième contribution est un algorithme basé sur les mouvements pour adapter en cours d'exécution le déploiement, en réponse aux aspects dynamiques de l'application.

Abstract

Modern multimedia applications are subject to an increasing complexity with widespread standards. This has led to the interest in dataflow approach that offers a powerful perspective on parallel computations at high level. In the meantime, the emergence of massively parallel architectures has revealed the trend towards heterogeneous Multi-Processor System-on-Chips (MPSoCs) to offer a better performance and energy tradeoff than their homogeneous counterparts. However, this also imposes challenges to the mapping of multimedia applications on such complex architectures.
This thesis presents an adaptive methodology for mapping dataflow applications on heterogeneous MPSoCs. This thesis focuses on video decoders specified in RVC-CAL language, a dedicated dataflow language for video applications. Existing static approaches cannot capture all behaviors in dynamic dataflow applications. Thus, this requires to adapt the mapping according to the input data. The algorithm offers some adaptive parameters combined with our analytical communication model to improve a performance while considering load balancing. We evaluate our algorithms on a set of randomly generated benchmarks and real video decoders like MPEG4-SP and HEVC. Experimental results reveal that our mapping methodology is fast enough (in milliseconds) and the runtime remapping significantly improves the initial mapping. In the remapping process, we take the migration cost into account because the reconfiguration time also contributes to the overall performance.

n d'ordre : 000000000

Université de Bretagne Sud

Centre de Recherche Christiaan Huygens - Rue de Saint-Maudé - 56100 Lorient, FRANCE
Tél : + 33(0)2 97 87 45 62 Fax : + 33(0)2 97 87 45 27

Contents

1 Introduction **1**
1.1 Evolution and trends in parallel systems . . . 1
1.1.1 In General Purpose Domain . . . 1
1.1.2 In Embedded Domain . . . 2
1.1.3 Embedded parallel platforms . . . 4
1.1.3.1 Homogeneous versus heterogeneous platform . . . 4
1.1.3.2 Memory architectures in MPSoCs . . . 6
1.1.4 Embedded system design . . . 7
1.2 Dataflow approach . . . 10
1.2.1 Dataflow models of computation . . . 11
1.2.1.1 Kahn process networks . . . 11
1.2.1.2 Dataflow process networks . . . 12
1.2.1.3 Synchronous dataflow . . . 13
1.2.1.4 Cyclo-static dataflow . . . 13
1.2.1.5 Quasi-static dataflow . . . 13
1.2.2 Taxonomy of Dataflow models of Computation . . . 14
1.2.3 Existing tools used in this thesis . . . 14
1.2.3.1 Open RVC-CAL Compiler: Orcc . . . 14
1.2.3.2 SDF For Free: SDF^3 . . . 16
1.2.3.3 System-Level Architecture Model: S-LAM . . . 16
1.2.4 Case studies - RVC-CAL applications . . . 16
1.3 Mapping problem . . . 18
1.3.1 Problem definition . . . 18
1.3.2 Challenges in mapping problem . . . 19
1.4 Our Contributions . . . 20
1.5 Outline . . . 21

2 Mapping Methodologies of Dataflow Applications on Parallel Architectures **23**
2.1 Dataflow Programming Models . . . 23
2.1.1 Embedded dataflow models . . . 23
2.1.1.1 Static dataflow models . . . 24
2.1.1.2 Dynamic dataflow models . . . 25
2.1.2 Dataflow tools for RVC . . . 25
2.1.2.1 OpenDF . . . 25
2.1.2.2 Orcc . . . 26
2.2 Embedded parallel architectures . . . 26
2.3 Mapping Methodologies . . . 28
2.3.1 Static mapping . . . 29

2.3.2 From static mapping to dynamic mapping 31
2.3.3 Dynamic mapping 32
2.3.3.1 On-the-fly mapping 32
2.3.3.2 Hybrid mapping 34
2.3.3.3 The perspective of hybrid mapping with runtime remapping 37
2.4 Conclusion 37

3 Communication Model Based Embedded Mapping 41
3.1 Problem definition 42
3.1.1 Application model 42
3.1.2 Architecture model 43
3.1.3 Communication model 43
3.2 Heuristic mapping algorithm 45
3.2.1 Evaluation metrics 45
3.2.2 GB4M2 Algorithm 47
3.2.2.1 Initialization phase 47
3.2.2.2 Computation phase 47
3.2.2.3 Communication phase 49
3.3 Experimental results 50
3.3.1 Simulation on Cadence virtual system platform - VSP . . . 50
3.3.2 METIS - Graph partitioning for heterogeneous multiprocessor architectures 51
3.3.3 Results with SDF benchmarks 51
3.3.4 Results with real video applications 52
3.3.4.1 The need of run-time mapping 55
3.3.4.2 MPEG4-SP and HEVC decoder 56
3.4 Conclusion 57

4 Move Based Algorithm 59
4.1 Problem definition 60
4.1.1 Application model 60
4.1.2 Architecture model 61
4.1.3 Communication model 62
4.2 Move based mapping algorithm 63
4.2.1 Parameters and evaluation metrics 63
4.2.2 Pre-processing - PP 65
4.2.3 Runtime mapping initialization - RMI 66
4.2.3.1 Algorithm principle 66
4.2.3.2 Computation phase 68
4.2.3.3 Communication phase 68

4.2.4 Runtime remapping - RR 69
4.2.4.1 Finding the possible moves 69
4.2.4.2 Trade-off between migration cost and performance improvement 70
4.2.5 Runtime scenario based simulation - RSS 72
4.3 Experimental results 73
4.3.1 Setup environment 73
4.3.2 The need of runtime remapping 74
4.3.3 Generated application graphs 75
4.3.4 Impact of migration cost at runtime 79
4.3.5 Real application graphs 79
4.4 Conclusion 83

5 Conclusion **85**
5.1 Conclusion 85
5.2 Perspectives 87
5.2.1 Short term 87
5.2.2 Long term 88

Bibliography **88**

1 Introduction

In modern embedded systems, the complexity of systems is rapidly increasing in hardware as well as software by the way of exploring parallelism and power saving. Indeed, embedded devices evolve towards heterogeneous Multi-Processor System-on-Chips (MPSoCs) that include more and more functional heterogeneous components on a single chip to satisfy the high performance and energy efficiency requirement of the embedded market. In the meantime the embedded software keeps growing exponentially to solve more difficult technical problems. The emergence of massively parallel architecture, along with the necessity of new parallel programming models, has revived the interest on dataflow models thanks to its ability to express parallelism. In consequence, the need to develop innovative methodologies and tools for mapping application specification onto such architecture platforms is also growing in current and upcoming embedded systems. As a design point of view, this process is necessary to bridge the gap between hardware efficiency and software flexibility while respecting time-to-market.

1.1 Evolution and trends in parallel systems

1.1.1 In General Purpose Domain

In desktop computing and high performance computing (HPC), researches are characterized by the assumption of homogeneous architectures and the goal of reducing the application makespan that is the total length of the schedule of an application. Current parallel programs typically depend on multi-threading, in which the application is expressed as a set of parallel tasks. Programming models are classified according to the way tasks or processes interact, as shared memory (e.g., Pthreads [1] or OpenMP [2]) and distributed memory (e.g., Message Passing Interface MPI [3]).

Pthreads, or Portable Operating System Interface (POSIX) Threads, are built on the top of sequential languages like C by providing libraries. Although Pthreads has its place in specialized situations, and in the hands of expert pro-

grammers, the unstructured nature of Pthreads constructs makes difficult the development of correct and maintainable programs [4]. In OpenMP, the use of threads is highly structured compared with Pthreads. The characteristics of OpenMP allow for a high abstraction level, making it well suited for developing HPC applications in shared memory systems.

As opposed to Pthreads and OpenMP, MPI was designed for distributed memory architectures. Over the last two decades, MPI has become the dominant HPC standard approach.

Since the trend in architecture design moved toward to heterogeneous architectures, programming models were developed to apply for heterogeneous architectures. In this area, two well-known programming models are Open Computing Language (OpenCL) [5] and Nvidia's Compute Unified Device Architecture (CUDA) [6].

In summary, all of the above mentioned programming models won in specific general purpose parallel domain. However, they are difficult to use in deeply embedded systems because of multi-thread safeness and runtime overhead when implementing them on modern embedded architectures.

1.1.2 In Embedded Domain

Today, embedded systems are everywhere and much more widespread than other computing systems with billions sold every year [7]. As can be seen in Table 1.1, Gartner predicts that the traditional PC market will follow the downward trend but sales of mobile phones are expected to reach 2 billion units in 2015. Gartner also estimates that smartphone sales will represent 88 percent of global mobile phone sales by 2018, up from 66 percent in 2014.

Table 1.1: Worldwide device shipments by segment (thousands of units)-Source: Gartner 6-2014 [8]

Device type	2013	2014	2015
Traditional PCs (Desk-Based and Notebook)	296,131	276,221	261,657
Ultramobiles, Premium	21,517	32,251	55,032
PC Market Total	317,648	308,472	316,689
Tablets	206,807	256,308	320,964
Mobile Phones	1,806,964	1,862,766	1,946,456
Other Ultramobiles (Hybrid and Clamshell)	2,981	5,381	7,645
Total	2,334,400	2,432,927	2,591,753

Recent reports have shown a significant drop in sales of desktop computers

while they are significantly increasing in smartphones, tablets and other embedded devices. The increasing usage of embedded systems also brings major challenges for designers. These challenges are quantifiable goals in embedded systems: real-time performance, restricted resources, power dissipation and market cost. Especially in multimedia embedded system, these systems process streams of data, data being e.g. audio, all kinds of sensing data, videos and graphics.

High-performance multimedia applications, such as video codecs, are becoming increasingly dynamic and complex. Fig. 1.1 illustrates the evolution of video compression standards in the past 20 years. The latest generation video codec is High-Efficiency Video Coding (HEVC), which can support 8K Ultra High Definition video, with a picture size up to 8192x4320 pixels.

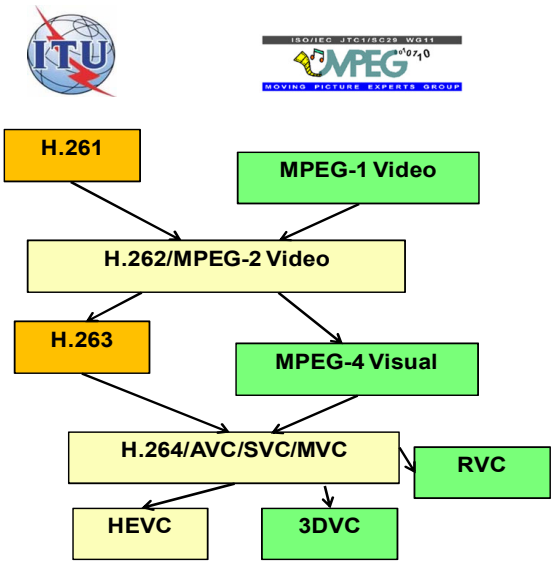


Figure 1.1: Evolution of video compression standards [9]

In the meantime, because of the attractive use of dataflow programming for the development of complex signal processing systems, various dataflow model of computations (MoCs) have been proposed and studied by academia and industries. The key characteristic of MoC is that it offers a powerful perspective on parallel computations at high level. In consequence, dataflow is gaining renewed popularity. Both academia and industry use dataflow as a programming paradigm, not only for performance analysis but also for design optimization. They can be classified in terms of static models, e.g. Synchronous Dataflow (SDF), Boolean Dataflow (BDF), Cyclo-Static Dataflow (CSDF) ..., and dynamic models, e.g. Scenario Aware Dataflow (SADF), Kahn Process Networks (KPN), Dataflow Process Networks (DPN).

At the target architecture level, heterogeneous Multi-Processor System on

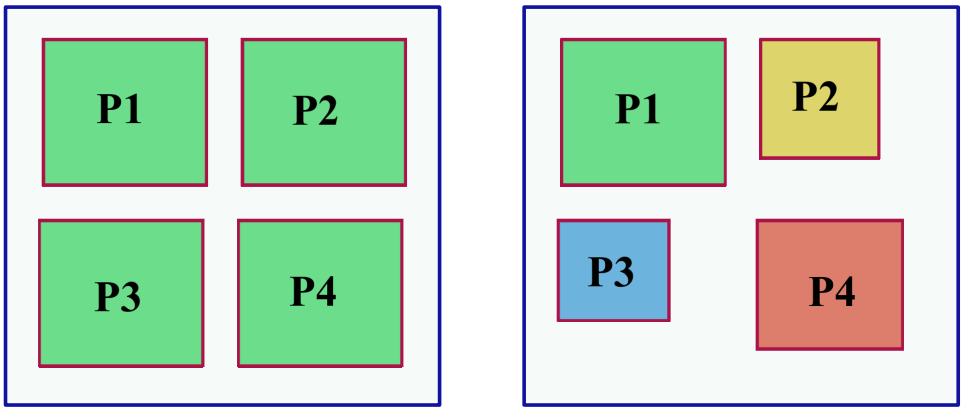
Chip (MPSoC) architectures are becoming emerging platforms for developing modern multimedia embedded systems because they are capable of providing better performance and energy trade-offs than their homogeneous counterparts.

However, they also bring some challenges concerning the mapping of multimedia applications on such a complex system. This requires to develop efficient mapping methodologies that can handle the increasing complexity both in applications and architectures. Moreover, it also requires highly flexible and re-usable design processes to deal with an exponential evolution in multimedia embedded systems.

1.1.3 Embedded parallel platforms

1.1.3.1 Homogeneous versus heterogeneous platform

MPSoCs are becoming a popular solution for multimedia embedded systems thanks to the advantages in parallelism and flexibility. MPSoCs can be divided into two categories: homogeneous and heterogeneous MPSoCs.



(a) Homogeneous (b) Heterogeneous

Figure 1.2: Multiprocessor architecture platform

- Homogeneous platforms are composed of the same processor type, e.g. Fig. 1.2a.
- Heterogeneous platforms are composed of different types of processors, e.g. Fig. 1.2b.

As an example of heterogeneous platform, Fig. 1.3 shows the overall structure of the OMAP 4 platform. This architecture was designed to drive smartphones, tablets and other multimedia-rich mobile devices.

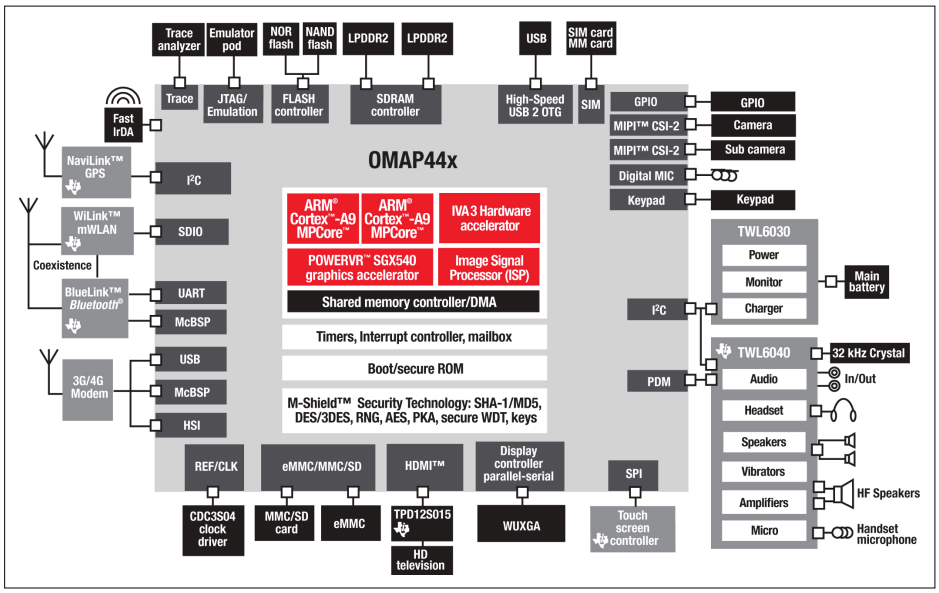


Figure 1.3: A OMAP44x system diagram

An important question that arises when designing an embedded system is whether a homogeneous or heterogeneous MPSoC should be used. Amdahl's law [10] is used to find the maximum expected improvement to an overall system when only part of the system is improved. The modern version of Amdahl's law states that if you enhance a fraction f of a computation by a speedup S, the overall speedup is:

$$Speedup_{enhanced}(f, S) = \frac{1}{(1-f) + \frac{f}{S}} \tag{1.1}$$

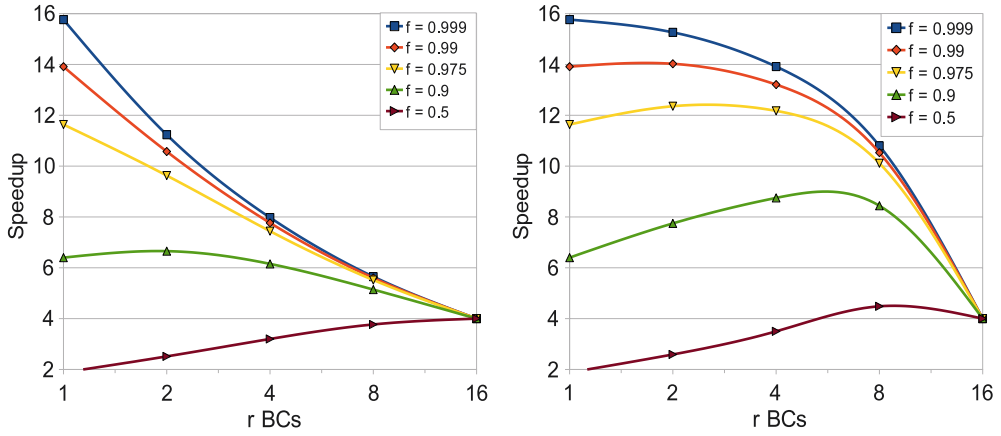
To complement this law in the multicore era, the authors in [11] offer a corollary of a simple model of multicore hardware resources. For homogeneous multiprocessors with n resources, suppose we can use the resources of r base-cores (BCs) to build one bigger core/processor, which gives a performance (relative to 1 base-core) of perf(r). If a resources for n base cores BCs are available on a chip, and all BCs are replaced with n/r bigger cores, the overall speedup is:

$$Speedup_{homo}(f, n, S) = \frac{1}{\frac{1-f}{perf(r)} + \frac{f.r}{perf(r).n}} \tag{1.2}$$

For heterogeneous multiprocessors, there are more possibilities to redistribute the resources on a chip. If only r BCs are available with 1 bigger core, the overall

speedup is:

$$Speedup_{heter}(f, n, S) = \frac{1}{\frac{1-f}{perf(r)} + \frac{f}{perf(r)+n-r}} \tag{1.3}$$



(a) Homogeneous systems (b) Heterogeneous systems

Figure 1.4: Comparison of speedup obtained by combining r smaller cores into a bigger core in homogeneous and heterogeneous systems [11, 12]

Figure 1.4 shows the speedup obtained for both homogeneous and heterogeneous systems, for different fractions of parallelizable software. This figure assumes $perf(f) = \sqrt{r}$. The x-axis shows the number of base cores that are combined into one larger core. In total there are resources for 16 BCs. The end-point for the x-axis is when all available resources are replaced with one big core. As can be seen, the corresponding speedup when using a heterogeneous system is much greater than homogeneous system. We also obtained similar performance speedups for other bigger chips, i.e. larger than 16 BCs. This demonstrates that *a heterogeneous platform can offer better speedup than a homogeneous platform.*

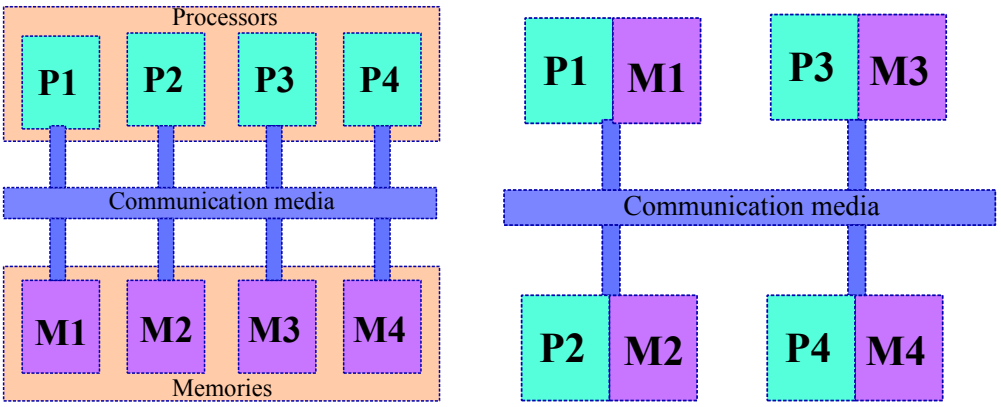
1.1.3.2 Memory architectures in MPSoCs

The memory architecture of multi-core platforms impacts directly the programming of processors. Therefore, the programming of shared memory platforms and distributed memory platforms is usually very different.

Shared memory architectures are usually classified based upon memory access times [13], as follow:

- Uniform Memory Access (UMA): In this architecture, all the processors share the physical memory uniformly, as shown in Fig. 1.5a, which means that all the processors have an equal access and access times to memory.

- Non-Uniform Memory Access (NUMA): All processors do not have equal access time to all memories. In other words, the memory access time varies with the memory location relative to the processor. Fig. 1.5b shows an example of NUMA architecture. In NUMA, when a processor accesses a remote memory, memory local to another processor or memory shared between processors, the access time is slower than it does from its own local memory.



(a) Uniform memory access (b) Non-uniform memory access

Figure 1.5: Shared memory architectures

Distributed memory systems, Fig. 1.6, require a communication network to connect inter-processor memory. Each processor has its own local memory and operates independently. When a processor needs to access to data from another processor, it is usually the task of a programmer to define how and when data is communicated by means of message passing techniques.

1.1.4 Embedded system design

As previously mentioned in Subsection 1.1.2, dataflow MoCs are widely used in multimedia domain. In order to select one of such dataflow models, a designer should take into account the trade-off among different criteria. Fig. 1.7 shows the comparison of different dataflow MoCs based on several criteria such as expressiveness, practicality, efficiency and analyzability. Acording to [14], DPN is the most suitable model for modern multimedia applications, which become increasingly complex and dynamic.

Traditional design methodology focused a single application approach, which means that an application is mapped on a set of architectures. Usually, the designers must rewrite their source code not only to optimize performance but also to

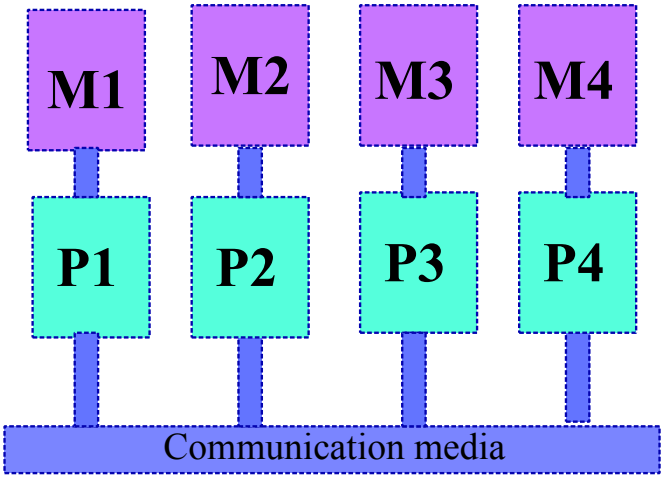


Figure 1.6: Distributed memory architecture

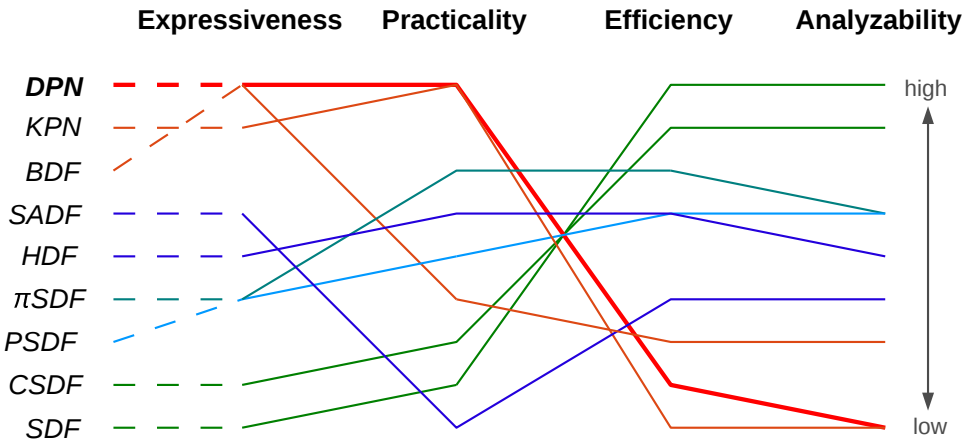


Figure 1.7: Comparison of dataflow MoCs, extending the classification system introduced by Stuijk et al. [15], which shows that DPN is the most suitable model for a practical programming language [14]

adapt these descriptions to an appropriate architecture platform. In other words, the monolithic approach of the reference software required complete rewrites of the code for new standards and failed to take advantage of the overlap in functionality. To improve the re-use and time-to-market, platform-based design methodology [16, 17] is being employed. The platform-based design methodology, as illustrated in Fig. 1.8, no longer maps a single application to an architecture that is optimal for this single application. Instead, it maps an application onto a hardware/software platform that can also be used for different applications from the same application space. The platform is defined as a family of architectures so that the application designer considers it as a common/generic platform, an essential feature to achieve re-usability.

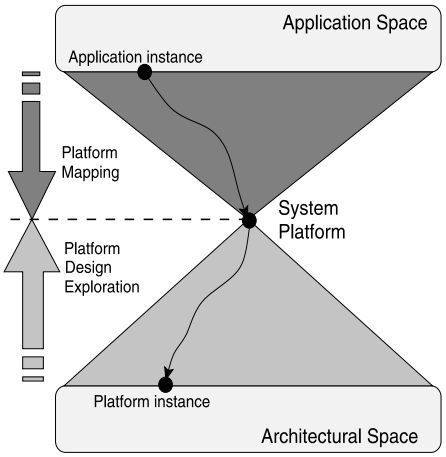


Figure 1.8: Platform-based design approach [16, 17, 12]

At a design point of view, the designers of embedded systems should take into account three aspects of the field: **architectures, applications** and **methodologies** as presented in Fig. 1.9.

An embedded system designer has to consider hardware as well as software design and how to tradeoff between the two. On one hand, the designer exploits all architecture components including processor architecture, memory organization, interconnection network and so on. On the other hand, the software architectures determine how we can take advantage of parallelism to improve performances. Moreover, the designer have to deeply understand about characteristics of their applications to optimize the design. The methodologies consist of modeling, analysis, simulation, synthesis and verification, which play an important role for successful embedded system design.

To handle the raising complexity in multimedia embedded system, models have been used to provide high-level of abstractions. Analysis and simulation tools are necessary to evaluate the efficiency and the design cost. Synthesis tools

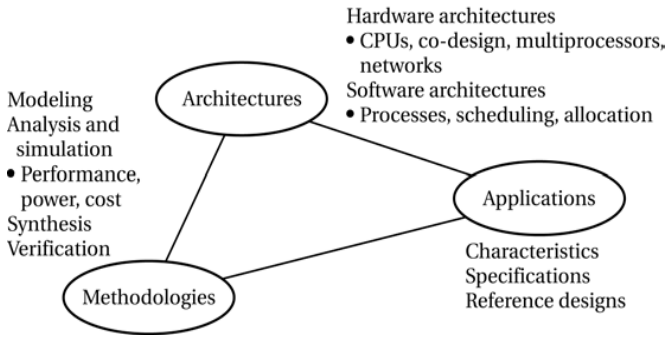


Figure 1.9: Aspects of embedded design [7]

are responsible to transform high-level specifications into optimized implementations. During the embedded design process, verification is required to ensure that the system is reliability.

1.2 Dataflow approach

Our methodology relies on the Y-chart approach for multimedia embedded system design which is first proposed by Kienhuis et al. [18]. It is a systematic design flow for design space exploration (DSE).

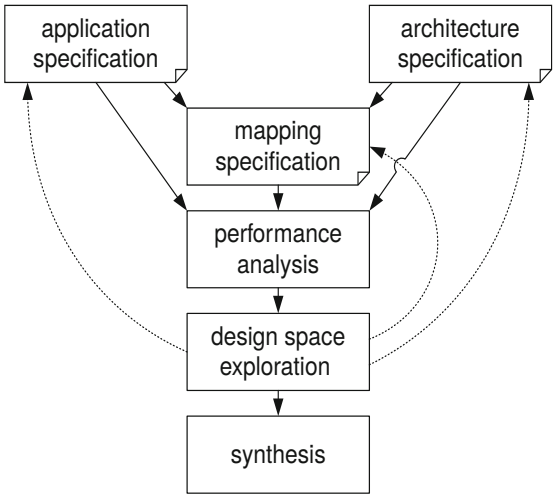


Figure 1.10: Y-chart approach for designing MPSoC [18, 19]

The key idea underlying this approach is to explicitly separate the specifications as shown in Fig. 1.10. This allows to use modular system-level specifications to facilitate rapid system modifications concerning the application, architecture and mapping.

1.2.1 Dataflow models of computation

Current trends in multimedia applications use dataflow model of computations (MoCs) to define the behaviors of a program described as a dataflow graph.

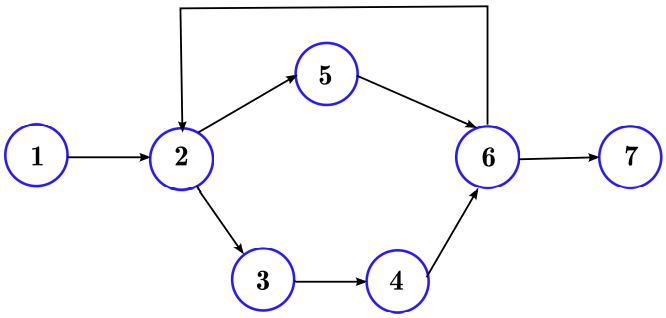


Figure 1.11: Dataflow graph with 7 actors and 8 FIFO channels

A dataflow graph is a directed graph as shown in Fig. 1.11 whose vertices are actors and edges are unidirectional FIFO channels with unbounded capacity, connected between ports of actors. Fig. 1.12 illustrates a network of actors which can interact by exchanging data (called tokens) through channels. Each actor has its input and output ports. During a process (also referred as firing an action), the actor consumes input tokens, produces output tokens and changes its internal state.

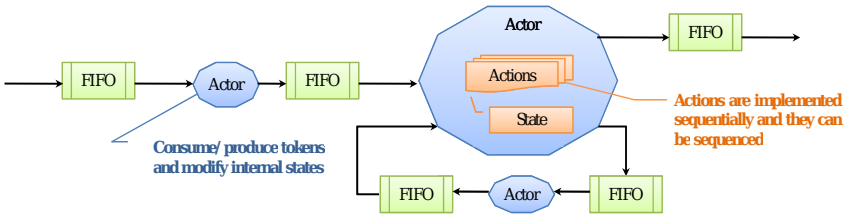


Figure 1.12: A network of actors [20]

Various dataflow MoCs were studied due to their attraction in signal processing domain.

1.2.1.1 Kahn process networks

Kahn process network (KPN) [21] is represented as a graph G=(V,E) such as V is a set of vertices that are called processes and E is a set of unidirectional

edges representing unbounded communication channels based on First In First Out (FIFO) principles.

In KPN, writing to the FIFO channels are non-blocking, i.e. a write returns immediately while reading to the FIFO channels are blocking. This means that when a process attempts to read data from an empty input FIFO channel, it must wait until the buffer has enough tokens to satisfy the read.

Each FIFO channel can carry an infinite sequence denoted X=$[x_1, x_2, \ldots]$, where each x_i is an atomic data object called a token. If the sequence X is a prefix odering of sequence Y, we can express as $X \sqsubseteq Y$, e.g. Given X=$[x_1, x_2]$ and Y=$[x_1, x_2, x_3]$ so we have X $\sqsubseteq$ Y. The empty sequence is denoted $\perp$, and is obviously a prefix of any other sequence. S^p is the set of p-tuples of sequences on the p FIFO channels. This means that X=$[x_1, x_2, \ldots, x_p] \in S^p$ represents the sequence consumed/produced by a process. The length of a sequence is given by $|X|$. When an actor is fired, its firing function F that consumes sequences of tokens on p input ports and produces sequences of tokens on q output ports defined as:

$$F : S^p \rightarrow S^q \tag{1.4}$$

1.2.1.2 Dataflow process networks

Dataflow process network (DPN) model [22] is closely related to KPN model. DPN allows to model any algorithm even non-deterministic ones. In this model, an application is represented as a directed graph G=(V,E) where V is a set of vertices that is called actors and E is a set of edges that represent FIFO channels. DPN adds expressiveness to the KPN model by allowing actors to test an input port either absence or presence of data. This avoids process suspension and resumption incurred in most implementations of KPN. Thus scheduling a DPN does not require context-switching nor concurrent processes. In DPN model, an actor executes (or fires) when at least one of its firing rules is satisfied. In case several firing rules are satisfied at the same time, a single one is chosen based on priority and its corresponding firing function is executed. Each firing consumes input tokens and produces output tokens. An actor can have N firing rules:

$$R = [R_1, R_2, \ldots, R_N] \tag{1.5}$$

A firing rule R_i is a finite sequence of patterns, one for each of the p input ports of the actor:

$$R_i = [R_{i,1}, R_{i,2}, \ldots, R_{i,p}] \in S^p \tag{1.6}$$

A pattern $R_{i,j}$ is a (finite) sequence. A firing rule i is satisfied if and only if $R_{i,j} \sqsubseteq X_j$, where X_j is the sequence of unconsumed tokens at input j. For some

firing rules, $R_{i,j}$ might be empty lists, $R_{i,j} = \perp$. In other words, any available sequence at input j is acceptable. Let symbol '*' denote a token wildcard, the sequence [*] is a prefix of any sequence with at least one token.

1.2.1.3 Synchronous dataflow

Synchronous Dataflow (SDF) model is a static dataflow model, in which an actor produces or consumes a fixed number of tokens per firing. SDF graphs can be scheduled at compile-time with bounded memory.

It may have a single firing rule, which is valid for any sequence S^p of a certain size on its inputs [23]. In case an actor has several firing rules, an actor is SDF if all its firing rules have the same consumption. In particular, any two firing rules R_a and R_b of an SDF actor must satisfy:

$$|R_a| = |R_b| \tag{1.7}$$

All the firing functions of an SDF actor must also produce a fixed number of tokens on the output ports:

$$|f(s_a)| = |f(s_b)| \qquad \forall s_a \in S^p, \forall s_b \in S^p \tag{1.8}$$

1.2.1.4 Cyclo-static dataflow

The cycle-static dataflow (CSDF) [24] extends SDF actors by allowing the number of tokens produced and consumed to vary in a periodic fashion. This variation is modeled with a state in the actor, which returns to its initial value after a defined number of firings. CSDF model has all characteristics of SDF model.

1.2.1.5 Quasi-static dataflow

Dataflow modeling is the question of striking the right balance between expressive power and analyzability: On the one hand, synchronous and cyclo-static dataflow limit the algorithms to be modeled as graphs with fixed production and consumption rates for their predictability and their strong properties that allow powerful optimizations to be applied. On the other hand, dynamic dataflow offers a large expressiveness, until Turing-completeness, able to describe complex algorithms with variable and data-dependent communication rate that makes their analyze and optimization ultimately harder.

Quasi-static dataflow differs from dynamic dataflow in that there are techniques that statically schedule as many operations as possible so that only data-dependent operations are scheduled at runtime [25]. An alternative to model quasi-static dataflow is the Parameterized Dataflow (PSDF) [26].

In order to select one of such dataflow models, the designers need to take into account a tradeoff between expressiveness and analyzability.

1.2.2 Taxonomy of Dataflow models of Computation

Dataflow MoCs are defined as subsets of the more general DPN model. Fig. 1.13 shows the taxonomy of dataflow models of computation.

SDF was first introduced by Lee and Messerschmitt in 1987 [23]. It is the least expressive DPN model but easier to analyze.

CSDF [24] extends SDF actors by allowing the number of tokens, both producer and consumer, to vary cyclically. Most of the studies in dataflow domain use static dataflow models because they are more analytical and predictable at compile-time. As a tradeoff between expressiveness and predictability, the definition of Quasi-static dataflow model QSDF was introduced [25]. Parameterized dataflow [26] is a higher-level approach to model quasi-static behavior by the extension of existing dataflow model using parameters modifiable at runtime.

KPN was proposed by Kahn in 1974 [21] as a general perpose scheme for parallel programming. DPN [22], also known as Dynamic dataflow model (DDF), is closely related to KPN. DPN is more expressive than KPN.

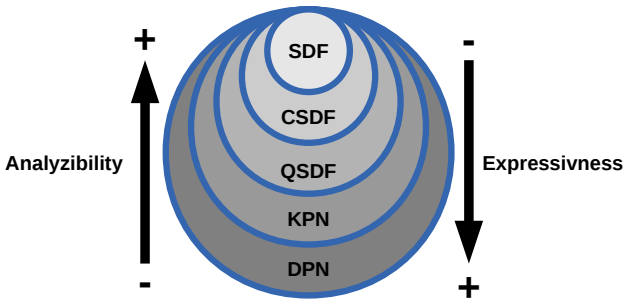


Figure 1.13: Dataflow Models of Computation [27]

With the increasing complex multimedia applications, the analyzability-expressiveness tradeoff moved towards more expressive models. In a practical point of view, DPN is well suited to model modern multimedia applications [27], by offering Turing completeness while also keeping an intuitive description.

1.2.3 Existing tools used in this thesis

1.2.3.1 Open RVC-CAL Compiler: Orcc

The rapid evolution in video codec standards increases the need to develop an innovating framework to overcome the lack of interoperability between the several

video codecs in the market. The Moving Picture Experts Group (MPEG) has introduced the Reconfigurable Video Coding framework (RVC) [28, 29], which offers reconfiguration, reusability and platform independent dataflow models. An RVC codec is described by using a domain-specific language, called CAL Actor Language (CAL) [30]. One of ubiquitous tools based on RVC-CAL framework is Orcc [31, 32].

Orcc [31, 33, 32] is an open-source toolkit dedicated to develop RVC-CAL applications. Orcc is a complete Eclipse based Integrated Development Environment, which aims at providing a compiler infrastructure to allow software/hardware code to be generated from dataflow descriptions. The compiler is able to translate RVC-CAL applications into an equivalent description not only software but also hardware languages for various platforms (FPGA, DSP,GPP, etc). In consequence, there are numerous back-ends in Orcc that target different languages (C, C++, LLVM, VHDL, Verilog etc.).

Currently, Orcc is supported to generate source code with the following back-ends:

- The C back-end produces an application described in portable ANSI C (Windows, Linux, Mac) like Pthreads with multi-core ability. The C back-end can be considered as the main backend of Orcc. In fact, this backend is used by the most part of the developers for the development of new application.

- The Java back-end generates class-based Java code that has the advantage of being used seamlessly within the Eclipse environment.

- The Jade back-end [34] produces LLVM assembly code targeted to be used by the JIT Adaptive Decoder Engine (Jade), which provides reconfigurability for software that runs as fast as the code generated by the C back-end.

- The TTA backend implements a full co-design to design a multi-softcore platform based on the Transport-Trigger Architecture and generates the software code executed on the processors using the TTA-based Co-design Environment (TCE) [35].

- The Verilog back-end generates Verilog description using the OpenForge tools.

- The Promela back-end generates Promela code that can be used with the SPIN model-checker to analyze properties of RVC-CAL applications.

Other back-ends are still developing to deal with a wide rage of modern platforms in the market. Moreover, some advanced analysis tools have developed to offer

predictable behaviors for dynamic dataflow applications. For example, Orcc has a dynamic analysis tool for actor classification [36, 27] to detect predictable behavior within a network. Orcc also offers a profiling tool to gather some useful profile information such as computation workload and communication workload of an application.

1.2.3.2 SDF For Free: SDF^3

SDF^3 is an open source tool [37] for generating random SDF graphs. It also offers many SDF analysis and transformation techniques as well as a function to visualize SDF graphs. The tool supports to generate different SDF graph benchmarks used in this thesis.

1.2.3.3 System-Level Architecture Model: S-LAM

S-LAM [38] is developed as a part of a prototyping tool called PREESM for Parallel and Real-time Embedded Executives scheduling Method. S-LAM provides a simple description of modern architecture platforms at high level of abstraction. Its description is a topology graph defining the data exchanges in modern platform such as heterogeneous architecture.

1.2.4 Case studies - RVC-CAL applications

Orcc provides a complete environment for users to exploit and develop current and future video decoders by using dynamic dataflow programming. We use the existing RVC-CAL applications such as MPEG4 Part 2 and MPEG High Efficiency Video Coding (HEVC) which are implemented in Orcc to study all dynamic behaviors of these complex dataflow applications.

MPEG4 Part 2 standard, also known as MPEG4 visual, was developed by the Moving Picture Experts Group (MPEG), a working group of the International Organization for Standardization (ISO). It was standardized in 1999 by the joint ISO/ITU (the International Telecommunication Union). It provides a highly flexible toolkit of coding techniques and resources. There is a set of coding tools, organised into 'profiles', recommended groupings of tools suitable for certain applications. Classes of profiles comprise 'simple' profiles (coding of rectangular video frames), object-based profiles (coding of arbitrary-shaped visual objects), still texture profiles (coding of still images or 'texture'), scalable profiles (coding at multiple resolutions or quality levels) and studio profiles (coding for high-quality studio applications).

Fig. 1.14 shows RVC-based description of the MPEG4 Part 2 SP decoder. The structure of the application graph can be partitioned into three parts, each one corresponding to a dedicated processing: parsing, residual decoding and motion

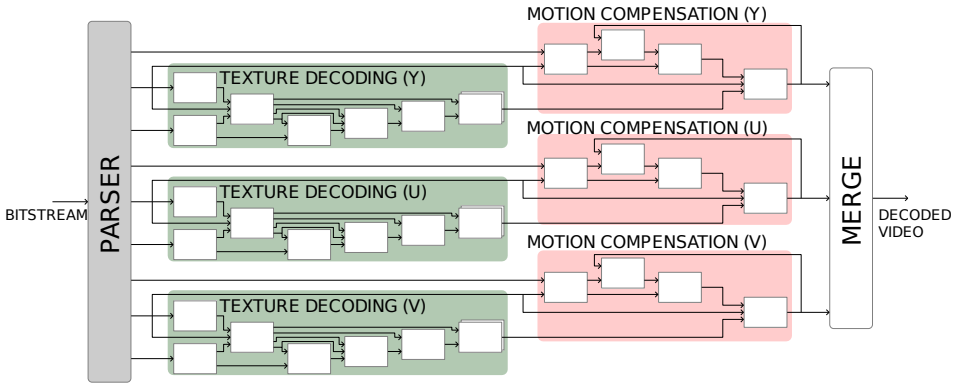


Figure 1.14: MPEG4 part 2 SP decoder [39]

compensation. To increase the parallelism exposed within the decoder, the parser can separate the processing of each image components in three parallel paths (Y, U and V). At the end of the processing, the image components are then merged back.

MPEGH Part 2, also known as MPEG HEVC/ H.265, is the newest video coding standard, developed conjointly by ISO/ITU. HEVC is improving the data compression rate, as well as the image quality, in order to handle modern video constraints such as the high image resolution 4K (3840 x 2160) and 8K (7680 x 4320). Another key feature of this new video coding standard is its capability for parallel processing that offers scalable performance on the trendy parallel architectures [14, 40]. The first version of the standard was completed, approved, and published in 2013. The second version was completed and approved in 2014 and published in early 2015. An implementation of the HEVC decoder using the RVC framework is presented in Fig. 1.15.

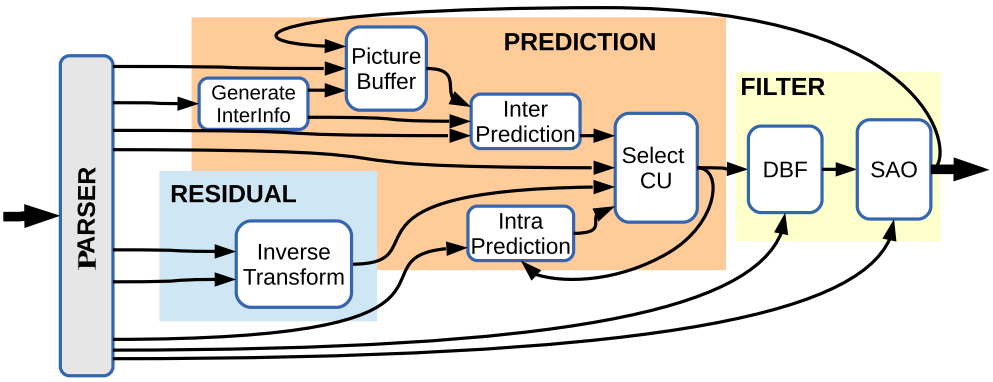


Figure 1.15: MPEG-H part 2 SP decoder [35]

Table 1.2 summarizes the properties of each description of these well-known

decoders. Their properties are the profile of the decoder, the parallelization of the decoding for each component, the number of actors and FIFO channels. The RVC-based video decoders are described with an average granularity (at block level), contrary to the traditional coarse-grain dataflow (at frame level). This fine-grain streaming approach induces a high potential in pipeline parallelism and the use of small communication channels, usually between 512 and 8192 rooms [14].

Table 1.2: Statistics about the RVC-CAL description of several MPEG video decoders [14]

Standard	Profile	Version	YUV	#Actors	#FIFOs
MPEG-4 Part 2	Simple Profile	RVC	yes	41	143
		Xilinx	no	34	86
		Ericsson	yes	42	105
		EPFL	no	13	29
		Irisa	yes	41	104
MPEG HEVC	Main	RVC	no	34	109
	Still Picture	RVC	no	31	74

1.3 Mapping problem

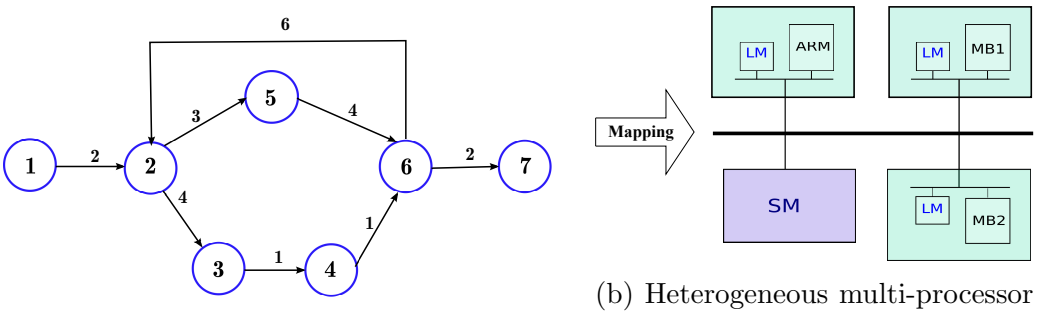
1.3.1 Problem definition

MPEG RVC defines RVC-CAL applications as dynamic dataflow applications, which are based on DPN model. An application is represented as a directed graph $DPN = (V, E)$, where the vertex set V is a set of actors $\mathbb{A} = [i_1, i_2, \ldots, i_n]$ and the edge set E is a set of FIFO channels $\mathbb{F} = [f_1, f_2, \ldots, f_k]$. Each FIFO channel carries a sequence of tokens $\mathbb{X} = [x_1, x_2, \ldots]$, where each x_i is called a token. In the application, the actors may be compliant with different models of computation (MoC) and so consume and produce a fixed or variable number of tokens, additionally the execution time can be more or less variable. Fig. 1.16a shows a simple dataflow application with 7 actors and 8 FIFO channels.

A heterogeneous MPSoc is composed of different processor types $\mathbb{P} = [P_1, P_2, \ldots, P_m]$, storage elements and interconnect. Each processor has its local memory (LM) and communicates with other processors through several buses and shared memories (SM). In our project, we target the Zynq platform. This architecture includes two ARM processors and multiple Microblaze processors, which have different hardware accelerators and frequencies as well. Thus, we

model the target platform as $MPSoC = (\mathbb{P}, \mathbb{CM})$, where $\mathbb{CM}$ is defined as a set of communication models, which are available in the platform.

Fig. 1.16b gives an example of a simple architecture platform with one ARM processor (that can run the mapping algorithm), two Microblaze (MB) with one hardware accelerator and all processors communicate through one bus and one shared memory (SM).



(a) Dataflow application

(b) Heterogeneous multi-processor platform

Figure 1.16: Dataflow application mapping on heterogeneous multi-processor platform

In this context, given a dynamic dataflow application, e.g. Fig. 1.16a, which is based on different MoCs, we assume that profiling mechanisms are available on the target platform so that we can measure at runtime the number of tokens produced and consumed by each actor as well as the execution time of actors. We then aim to map the dataflow actors onto various computation and communication resources, e.g. Fig. 1.16b, with the objective to optimize the application throughput. Finding a high-quality mapping solution for such a dynamic dataflow application on heterogeneous platform is an NP complete problem. This is why we consider the dataflow application at high-level description and propose a heuristic approach in order to produce an efficient solution in few milliseconds.

The application mapping problem has been addressed as one of the most urgent problem to be solved for implementing embedded systems [41, 42, 43].

1.3.2 Challenges in mapping problem

As mentioned in the previous section, the complexity of modern multimedia systems are increasing both in applications, e.g. video codecs, and parallel architectures, e.g. heterogeneous platforms. This brings more challenges for mapping such a complex system. Therefore, it is difficult and time consuming to find an optimal solution that satisfies all performance and power constraints. Moreover, mapping problem is known as NP-hard problem [44]. This is why many studies

employ heuristics based on the application domain knowledge to find a nearly optimal solution. In regard to application model, static models are restricted in the kinds of applications they can express. This also means that static mapping approaches are not suitable because they can not handle the behaviors of dynamic applications. As a result, the trend moves towards runtime/dynamic mapping techniques. Runtime mapping methodologies face the challenge to map tasks/actors of a application on MPSoC platform without advance knowledge about a dynamic changing in workload and actor migration in the platform at design time. Furthermore, runtime mapping may have some following requirements:

- A fast solution in order not to degrade the overall system performance.
- Ability to capture runtime varying workloads in the systems and runtime changing environments.
- Mapping a new task/actor into the system, which needs some information concerning available resources.
- A flexible approach, which has some adaptive parameters to deal with a wide range of applications at runtime.
- Ability to do a remapping of application, when a current mapping is not sufficiently optimal.

1.4 Our Contributions

This thesis aims at providing an adaptive mapping method for dynamic dataflow applications on heterogeneous multiprocessor platform at runtime. In order to explore all issues and opportunities related to dynamic dataflow applications, we consider several video decoders which are already specified as Dataflow Process Network (DPN) in RVC-CAL framework. In this context, the objective is to maximize a system throughput when mapping RVC-CAL applications on any heterogeneous multiprocessor platform. Since DPN model is used to express dynamic behaviors of an RVC-CAL application, the mapping method can not longer be static. Thus, it requires a runtime mechanism to handle behaviors of a dynamic application. The mapping algorithm has to be fast enough to produce a decision of mapping at runtime. It should have some adaptive parameters to improve a performance and it may be able to do a remapping when a current mapping is not sufficiently optimal at online. Since the reconfiguration time also contributes to the overall performance, we should take a migration cost into account. Indeed, different mappings may incur different migration costs [45].

This thesis makes the following contributions as illustrated in Fig. 1.17:

- We present a novel hybrid algorithm, which takes the advantages both at design time and runtime, for mapping a dataflow application on heterogeneous multiprocessor platforms. As many studies, we also take load balancing into account. At design space exploration, we rely on the Y-chart co-design approach to make our mapping algorithm independent from application model and architecture model. By using this approach, we consider the co-design flow based on higher-level of abstractions that are necessary to deal with the growing complexity of embedded systems. Additionally, we do not need to find all mapping solutions at DSE as a reference solution for using at runtime. Instead, we propose a processing budget according to a dataflow application. This method help us to save exploration time as well as a memory for storing all mapping solutions at design time. We propose an analytical communication model for estimating the delay (latency) of the data on the communication media at runtime. This communication model is flexible since it can apply either NoC or Bus based architecture. The advantages of our algorithm are adaptability, predictability and a fast solution at runtime. This work has been published in DASIP 2014 conference [46].

- Further, we exploit dynamic behaviors of dataflow applications to demonstrate the need of doing the runtime remapping as well as illustrate the impact of migrated actor at remapping phase. We then introduce a Move Based Algorithm, namely MBA, which is compliant with Bus and NoC models takes both computation cost and communication cost while allowing remapping dataflow actors at runtime onto heterogeneous MPSoC. MBA supports the adaptivity at runtime and also takes migration cost when doing the remapping actor.

- Our mapping method can apply for mapping a dynamic dataflow application which is based on RVC-CAL framework onto any heterogeneous multiprocessor platform. This work has been submitted for review to Journal of Signal Processing Systems - Springer.

- We conduct the experiments with our runtime scenario based simulation for both randomly generated dataflow graphs and MPEG4-SP applications.

1.5 Outline

The remainder of this thesis is organized as follows.

- Chapter 2 reveals not only the interest in dataflow programming models but also the emergence of MPSoC platforms. We provides a wide survey

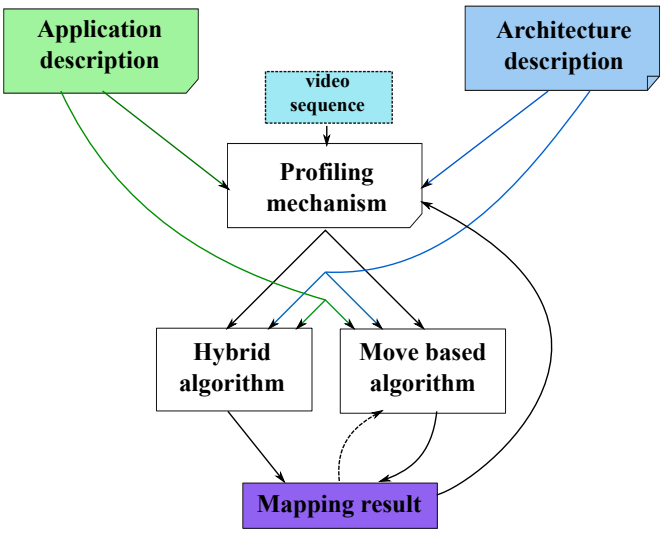


Figure 1.17: Overview of our design flow based on Y-chart approach

and classification of mapping methodologies in the literature and shows the emerging trends for multimedia embedded systems.

- Chapter 3 presents an analytical communication model. We propose a novel hybrid algorithm for mapping dataflow actors on heterogeneous MPSoC. This algorithm follows a greedy fashion to produce a fast mapping solution at runtime.
- Chapter 4 extends our approach in Chapter 3 with MBA algorithm to offer the runtime remapping.
- Chapter 5 concludes the thesis and also gives some perspectives.

2

Mapping Methodologies of Dataflow Applications on Parallel Architectures

In this chapter, variety of dataflow programming models are first discussed in Section 2.1. As target architectures for these dataflow applications, we then introduce embedded parallel architectures in Section 2.2. In this consideration, plenty of researchers have addressed application mapping problem as one of the most urgent problem to be solved.

Since this thesis proposes a novel method for mapping dataflow actors on heterogeneous platforms, the remaining section focuses on classification of the mapping methodologies in the literature and also highlights the emerging trends in the mapping methods.

2.1 Dataflow Programming Models

As a rapid evolution in multimedia applications as well as architecture platforms, graphical programming models became quite popular, since they provide to algorithm designers a natural way of specifying an application. Commercial examples include MATLAB Simulink [47], National Instruments LabVIEW [48] as well as Synopsys Signal Processing Work system (SPW) [49] and System Studio [50]. Dataflow semantics are a common underpinning of most graphical programming models.

2.1.1 Embedded dataflow models

Dataflow models, as introduced in Subsection 1.2.1, have achieved a dominant position in performance analysis as well as design optimization of multimedia embedded systems. Dataflow MoCs can be split into two categories: one is static models and the other one is dynamic models.

2.1.1.1 Static dataflow models

There are numerous dataflow MoCs, which are known as static models. This kind of model assumes that actors have a fixed token production and consumption on each firing.

SDF is also known as Weighted Marked Graphs in Petri Net literature. Schedulability and memory consumption of SDF applications can be known at compile-time [51]. Pioneering works on SDF graphs were published by Lee et al. in [23]. Prof. Edward Lee is famous with the Ptolemy project [52]. This project studies modeling, simulation, and design of concurrent, real time, embedded systems. In this project, different kinds of dataflow models have been developed and exploited. A free tool set for generating and analyzing SDF, CSDF and so-called Scenario-Aware Dataflow (SADF) are available in the SDF^3 project [37]. Diverse studies in the literature investigate SDF model in the field of multimedia applications such as [53, 54, 55, 56, 57, 58]. In [59], authors present a method for throughput analysis of SDF applications. Their approach is based on explicit state-space exploration and avoids the translation to Homogeneous SDF (HSDF) application. HSDF model is a special case of SDF model in which all token production and consumption rates are 1. In [60], authors address the problem of mapping HSDF applications on multiprocessor platform with the objective of maximizing application throughput by using Sat-based techniques. The authors in [54] propose a method to compute throughput of an SDF applications in which the execution time of actors can be parameters. To explore the parallelism with heterogeneous architectures, authors in [57] present a methodology for improving the system throughput by using SDF transformations. Many researchers [56, 60, 61, 58] consider their applications as SDF or HSDF model. This kind of models is also known as static model, which is easier to analyze and predict at design-time. Researchers in [55, 62] provide a complete approach to solve the allocation and scheduling of SDF applications on MPSoCs.

CSDF model extends SDF with the notion of state. With CSDF model, authors in [63] present a practical and accurate throughput analysis since their method can give tight estimates on the minimum throughput. A comparison between SDF and CSDF model were explored in [64]. The need for a tradeoff between expressiveness and predictability has brought the definition of so-called "quasi-static" dataflow model [26, 65, 66].

As static dataflow models are restricted in the kinds of applications they can express, these models can not express the dynamic behaviors of modern video applications. This leads to many studies of MoCs that can express the dynamic behaviors of modern multimedia applications.

2.1.1.2 Dynamic dataflow models

In contrast to static model, dynamic dataflow models are able to capture the behaviors of dynamic applications. In dynamic dataflow domain, it is impossible to know production and consumption behavior of actors at compile time since each actor has a set of firing rules and can be fired if one of them is satisfied.

SADF [67] extends SDF with scenarios, which represent different modes of operation based on resource requirements. This makes it possible to capture a dynamic behavior of application to save resources. SADF improves SDF in terms of expressiveness to express dynamism. Different scenarios may differ in their execution time and communication rates. However, all scenarios are generated by a probability of occurrence and each scenario can be modeled with SDF model. Authors in [15] surveys SADF and compares different dataflow MoCs according to their expressiveness, expressiveness, analyzability and implementation efficiency.

KPN is another MoC that can be used to express behaviors of dynamic application. However, KPN requires a complex run-time mechanism that leads to a large implementation overhead [15]. Lee et al. were pioneers in a theory of dataflow process network (DPN) [22]. DPN is a special case of KPN but it can be used to model the most general form of dataflow MoCs. Therefore, recent researches employ KPN, DPN models as in [68, 69, 70, 15, 71].

Hence, with embedded multimedia becoming more complex, the trade-off between analyzability and expressiveness moved towards more expressive models.

2.1.2 Dataflow tools for RVC

The initial work for introducing the MPEG Reconfigurable Video Coding (RVC) framework [29] started in 2004. Both academia and industries have developed a set of tools supporting RVC framework. The key characteristics of MPEG RVC are flexibility, re-usability and platform independent dataflow models. In this innovative framework, the MPEG RVC working group has adopted CAL (the Cal Actor Language) [30] as part of their standardization efforts as shown in Fig. 2.1. CAL actor language was developed at University of California at Berkeley in 2001. This language was born from Ptolemy II project [52] to modeling complex signal processing systems dedicated to software and hardware code synthesis.

2.1.2.1 OpenDF

The Open Dataflow Environment (OpenDF) [72] is a dataflow toolset. OpenDF framework is composed of editing the CAL actor, some analysis tools and multi-target compiler. The compiler in OpenDF supports three back-ends to generate code for different target platforms. The first one is a back-end for generation of

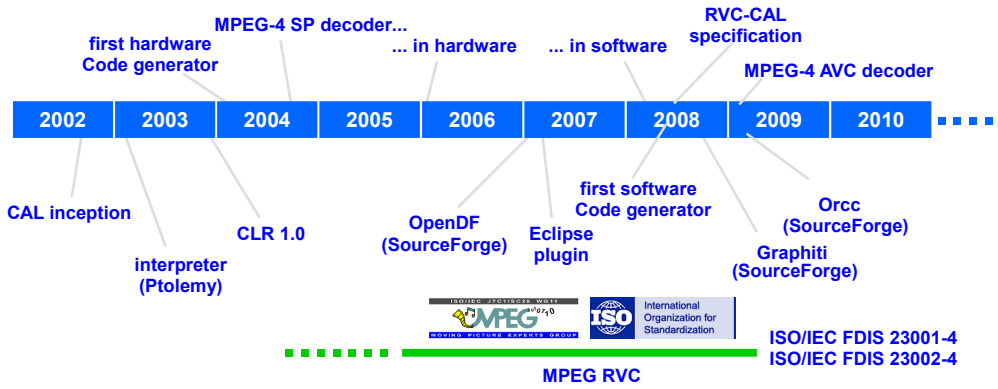


Figure 2.1: Pictorial representation of CAL language and tools development and the timeline of the steps of MPEG RVC standardization [39]

HDL (VHDL/Verilog) [73], and the second back-end generates C code for integration with the SystemC tool chain [74]. The last one is dedicated to embedded platforms based on ARM processor [75]. As mentioned in [33], OpenDF has some technical limitations and is no longer maintained. This leads to the birth of Open RVC-CAL Compiler (Orcc) project in 2009.

2.1.2.2 Orcc

As mentioned in Subsection 1.2.3, Orcc is a complete framework, which has been developed by Orcc team and widely used by academic and industrial researches. Indeed, Orcc composes of rich eclipse-based editors, integrated simulators and multi-target development tools [32]. Many synthesis tools for hardware and software co-design have been developed by Orcc communities [33, 76, 77, 78, 79, 80].

2.2 Embedded parallel architectures

Early works ignored data communication and focused on scheduling [56, 60]. The authors target homogeneous MPSoCs with the objective of maximizing system throughput. Fig. 2.2 shows typical examples of how hardware evolution today with OMAP and Snapdragon platform families. The solid curve presents the development of OMAP family from Texas Instruments (TI) [82]. The OMAP5430, which includes ARM Cortex-A15 MPCore (with 4 cores each), two ARM Cortex-M4, a graphic processor (PowerVR), a C64x DSP, an Image Video Audio accelerator (IVA) and an image signal processor, contains 14 processing elements (PEs). We can observe the similar trend with Snapdragon family from Qualcomm [83], the dash curve in Fig. 2.2.

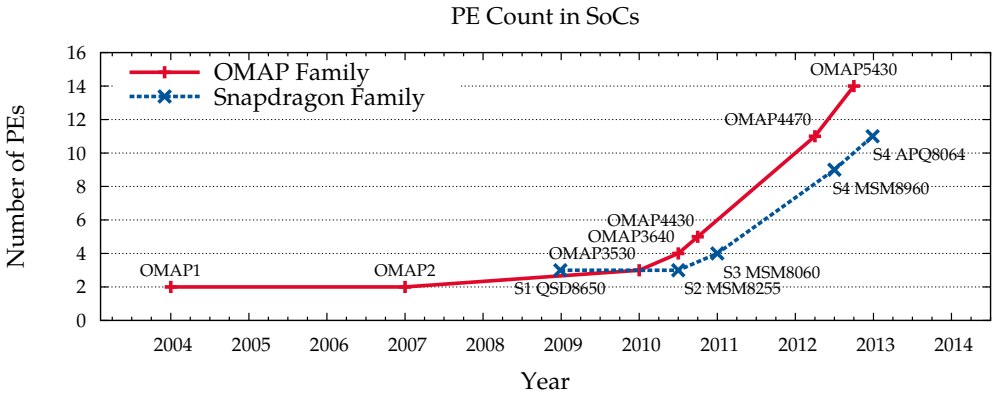
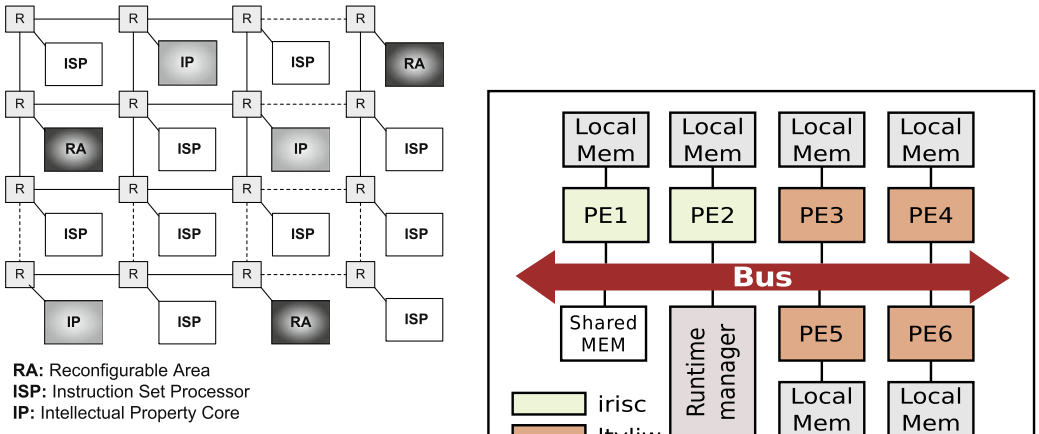


Figure 2.2: SoC Trends: Processing elements in TI OMAP Processors and Qualcomm Snapdragon Processors [81]

As the current trend towards to a heterogeneous platform for high performance and energy efficiency, heterogeneous architectures with many different communication standards including hardware support are available. *Therefore, the role of communication during application synthesis can no longer be ignored.* Indeed, the latency increases with the number of processors connected to it [84]. In [70, 71], communication is modeled by annotating latencies on the edges of the application graph. Other authors treat communication mapping in the context of single inter-processor communication (IPC) [58] which does not correspond to the situation in today's MPSoCs. In [58], additional actors, namely send and receive, are bound on the buses in addition to original computation actors that are bound on processors. The approach relies on an ILP formulation. Our goal is to embed the application to be typically executed by an ARM processor and there is not yet any ILP solver for this kind of embedded processor.

For such a complex platform, we need to estimate the delay for data to be transmitted that increases with the traffic. This is typically observed in NoC [87], where the latency increases with the injection rate. For the sake of clarity, we will use the term latency to denote the time taken by one data to travel on the bus. In [88], the authors investigate the performance of mapping algorithms in NoC based heterogeneous MPSoCs with the objective of NoC congestion minimization. They employ the Minimum Maximum Channel Load (MM), the Minimum Average Channel Load (MA) and the Best Neighbor (BN) to reduce the occupancy of the NoC channels and also the execution time of the heuristics. Thereafter, the authors in [85] present a number of communication-aware runtime task mapping heuristics on NoC based MPSoCs. They extend MPSoC architecture used in [88] to support more than one task for each processing node. Their NoC based MPSoC architecture, as shown in Fig. 2.3a, contains a set of different nodes such as Instruction Set Processor (ISP), Reconfigurable Area (RA) and Intellec-



(a) NoC based MPSoC architecture [85]

(b) Bus based MPSoC architecture [86]

Figure 2.3: Example of NoC and Bus based MPSoC architecture

tual Property Core (IP). They interact via a communication network with 2D mesh topology. The communication network uses wormhole packet switching, handshake control flow, input buffer and deterministic XY routing algorithm. Recently, many works consider NoC as a communication scheme [89, 61, 68, 90]. Their communication models are limited to NoC and very basic since they are a number of hops, which do not correspond to flexibility communication media in heterogeneous MPSoCs.

In [86], authors propose *communication primitives*, which can capture the variety of IPC software interfaces in today's MPSoCs. They target the Densely Connected Platform (DCP) in Fig. 2.3b in which PEs can communicate by various means like TI's Keystone [91]. The PEs irisc and ltvliw are in-house cycle accurate models developed with Synopsys Processor Designer [92]. The bus in the DCP is a transaction accurate model of an AMBA AHB bus. For the bus and the memories, models from the Synopsys IP library [93] are used. The runtime manager in Figure 2.3b controls the execution of processes on the different PEs. However, this kind of model only applies for bus based architectures with and without pipelined communication.

2.3 Mapping Methodologies

The application mapping problem, which has been identified as one of the most urgent problem to be solved for implementing embedded systems [41, 42, 43], is a NP-hard problem [44]. There are different criteria to classify the mapping methodologies such as optimization goal, target architecture, workload, etc. Fig.

2.4 shows a taxonomy of mapping methodologies based on workload scenarios. In general, there are two kinds of workload, either static or dynamic. For static workload, the mapping method can perform optimization at design time. However, for dynamic workload, the variations in terms of workload occur at runtime. This leads to classify as static and dynamic/runtime methodologies respectively. Both methodologies target either homogeneous or heterogeneous multiprocessor architectures. The type of platform can be fixed platform or generic platform. When the mapping methodology considers a fixed platform, this means that the mapping method depends on a specific platform. In case the mapping methodology applies for generic platform, it can work with a wide range of platform platform. Examples of generic heterogeneous architectures are the Texas Instruments OMAP platforms [82] or Zynq platforms [94], which contain a mixture of dedicated programmable cores, various hardware accelerators, different kinds of interconnects and memory architectures.

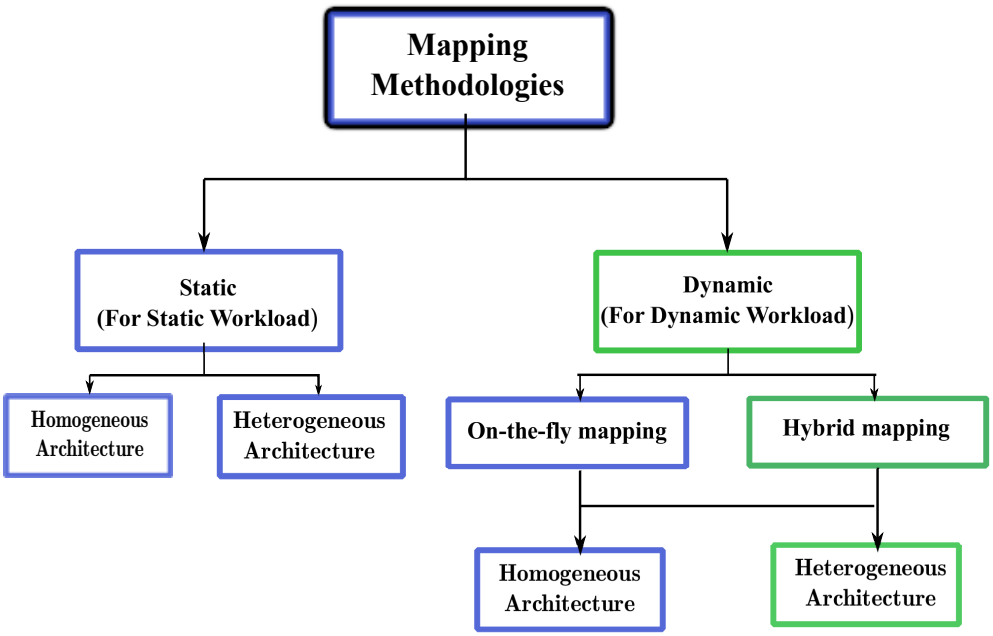


Figure 2.4: A Taxonomy of Mapping Methodologies, Extending the Classification in the Survey Introduced by [43]

2.3.1 Static mapping

Static mapping strategies are an off-line mapping which has global view of the system at design time. These methods can explore more thoroughly system information to achieve optimal mapping solutions. They are suitable for static

Table 2.1: Classification of static mapping methodologies

Ref.	MoC	Platform	Optimization Goal
[56]	SDF	Fixed-HO	Memory, Throughput
[60]	HSDF	Fixed-HO	Throughput
[54]	SDF	N/A	Throughput
[63]	CSDF	Fixed-HE	Accurate Throughput
[62]	SDF	Fixed-HE	Throughput
[57]	SDF	Generic-HE	Throughput
[55]	SDF	Fixed-HO	Solving time, Throughput
[59]	SDF	N/A	Throughput
[95]	N/A	Fixed-HO	Execution time
[96]	N/A	Fixed-HO	Execution time
[53]	SDF	Fixed-HE	Resource allocation
[97]	SDF	Fixed-HO	Efficient synthesis

workload applications and fixed platforms as well. Most of the mapping methodologies reported in the literature are static mapping techniques [98]. Table 2.1 reveals different static approaches for static applications, based on different static models such as SDF, CSDF, in different manners and optimization goals as well. The methodologies target homogeneous (HO.) or heterogeneous (HE.) multi-core platforms. There are numerous optimization goals, e.g. throughput, execution time, solving time, resource allocation, energy consumption ..., from different studies in the state of the arts [43]. As throughput is one of the most important metrics in the domain of multimedia to evaluate the system performance, authors in [59] present a method for throughput analysis of SDF graphs. In [53], authors propose a resource allocation for SDF graphs, which benefits from throughput analysis in [59]. By using cyclo static dataflow model (CSDF), authors [63] give a tight estimate on the minimum throughput for an application mapped on a multiprocessor system.

Various strategies have been reported to solve the static mapping problem. For example, Genetic Algorithm (GA) is used in [57], Simulated Annealing (SA) in [95], branch-and-bound and SAT solving in [60] and Integer Linear Programming (ILP) in [96]. Authors in [57] use GA algorithm to maximizing throughput of SDF application on a heterogeneous platform. The results indicate that their approach outperforms than other techniques such as Flextream [99] with replication heuristic and optimal ILP mapping. The disadvantages of this method

are high computational cost and a large buffer sizes for an application implementation. With SA algorithm in [95], authors solve the task mapping problem based on simultaneous optimization of execution time and memory consumption. Their SA algorithm starts with an initial solution in which all task graph nodes are mapped to a single processing element and then iterates through various mapping candidates to find a better solution. Authors in [96] propose an ILP formulation for task mapping and scheduling problem to reduce system execution time. These approaches provide efficient and optimal mapping solutions that can be used as reference but they take a lot of time for searching solution of large scale problems such as applications with large number of tasks mapped on a platform with a lot of processors. Some approaches improve computational cost by other search based mapping strategies. Authors in [60] address the mapping problem of HSDF graph onto a multiprocessor platform with the objective of maximizing system throughput. They combine both branch-and-bound and SAT-solving to explore the design space of all possible actor-to-processor mappings. Their integrated approach reduces the solving time compared with Logic Based Benders Decomposition approach [100] significantly. In [55], authors propose a complete algorithm based on Constraint Programming to solve the allocation and scheduling problem of SDF graph onto a multi-core platform. Their objective is minimum throughput requirement while reducing the solving time of the algorithm.

Broadly, static approaches have interesting results in terms of performance. Moreover, they show lack of flexibility: all the parameters of the dataflow application need to be known and fixed at design time. If the applications or platforms change, then re-computation is necessary. Further, they are unable to handle dynamic behaviors of dynamic dataflow applications. Even if these mapping methodologies are not suitable for runtime varying workloads, they can be considered as a reference solution or initial mapping at runtime.

2.3.2 From static mapping to dynamic mapping

Authors in [101] present an iterative probabilistic analysis to accurately predict the performance of multi-application mapped on a multiprocessor platform. They measure that the runtime complexity of their algorithm is only 300 μs with ten applications on a 500 MHz processor. Nonetheless, they consider SDF graph applications in context of multi-application, the dynamic behavior is only the variation of execution time of applications. In [102, 15], the authors express the dynamic behavior of an application by describing several static scenarios. Consequently, the programmer has to predict all possible scenarios and describe them in a static way. Schor et al. [68] present a whole scenario-based design flow for mapping streaming application modeled by KPN onto on-chip many-core system. In [58], authors address multi-objective mapping problem of SDF graphs onto het-

erogeneous multiprocessor platforms based on a combination of an evolutionary algorithm with an ILP. They obtain up to 12x runtime efficiency compared to the global ILP without compromising throughput optimality. However, the runtime complexity of their algorithm significantly increases with the number of actors in a SDF graph application. Our goal is to embed the application to be typically executed by an ARM processor and there is not yet any ILP solver for this kind of embedded processor.

Authors in [61] propose a method, as shown in Fig. 2.6 for efficient mapping of throughput constrained applications on MPSoC platforms. Their method outperforms the time required (in milliseconds) to map throughput constrained multimedia applications on a 4x4 MPSoC platform. In application model, they still consider multimedia applications as SDF model.

These approaches employ static dataflow model like SDF to exploit the dynamic behaviors of the system. In other words, they do not use the dynamic models to explore the dynamic behaviors of a system directly.

2.3.3 Dynamic mapping

In contrast to the static mapping, dynamic mapping is an online mapping method. Hence, it can handle dynamic behaviors of workloads at runtime. With dynamic dataflow applications, we can not know dynamic behaviors such as computation time, communication time of each task and runtime changing environments at design time. In dynamic mapping, the time taken to map each task is important since it contributes to overall application execution time. Some evolution algorithms (GA, SA ...) and ILP methods are not acceptable for runtime mapping in embedded systems since they have high computational costs with large scale problems as mentioned above. Therefore, some heuristic methods like greedy fashion are used to trade-off between efficient mapping and mapping overhead.

The dynamic mapping methodologies reported in the literature can be divided into three kinds of approaches, namely *on-the-fly*, *hybrid* and *hybrid with runtime remapping (Hybrid+R)*.

2.3.3.1 On-the-fly mapping

On-the-fly mapping is an online mapping method which is not based on any prior analysis as illustrated in Fig. 2.5. In other words, all the processing of this method is performed at runtime. Table 2.2 classifies based on MoC, platform and optimization goal of this mapping methodologies. Authors in [103] present heuristics with on-the-fly task mapping on heterogeneous MPSoCs. The target architecture contains software and hardware processing element(PE), where each PE can support only one task. The authors apply two runtime mapping algo-

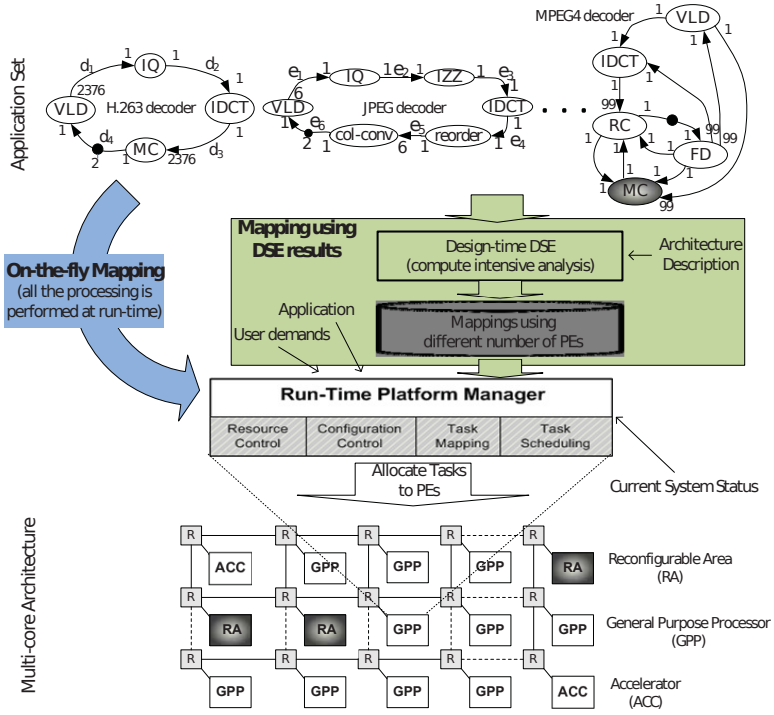


Figure 2.5: On-the-fly and hybrid mapping flow [43]

rithms: first free (FF) and nearest neighbor (NN) combined with their proposed algorithms: path load (PL) and best neighbor (BN) to reduce the solving time of FF and NN algorithms. These heuristics are suitable with NoC architecture platforms. Authors in [104] introduce a runtime task assignment heuristic for efficiently mapping the tasks in a multi-core system containing FPGA fabric tiles. This heuristic is capable of managing a configuration hierarchy and improves the task assignment performance.

Table 2.2: Classification of on-the-fly methodologies

Ref.	MoC	Platform	Optimization Goal
[103]-2010	N/A	HE.	Communication overhead
[104]-2008	N/A	HE.	Mapping time
[105]-2011	N/A	HE.	Execution time, Resource utilization
[85]-2010	N/A	HO.	Energy consumption, Communication overhead
[101]-2010	SDF	Fixed-HE	Performance

In [105], authors introduce runtime self-adaptability task allocation on hetero-

geneous MPSoCs. They propose a set of key parameters, which allow dynamically adjusting based on current resource utilization at runtime. They also introduce an adaptive clustering approach for efficient reduction of communication load. In this study, applications are represented as task graphs and their target platforms are based on the architecture suggested in GENESYS with TDMA-arbitrated NoC. Authors in [85] present mapping heuristics which allow more than one task can be supported by each PE. Their proposed heuristics are capable of alleviating Network-on-Chip (NoC) congestion bottlenecks when compared to existing alternatives. To reduce the communication overhead, their heuristics try to map the adjacent communicating tasks on to the same PE. By doing this way, the authors show that the energy saving can be up to 44% when mapping tasks of applications onto an 8×8 NoC-based MPSoC.

Since on-the-fly strategies start mapping an application without any previous analysis, they take more time to perform the required analysis at runtime. The need of low complexity in mapping method at runtime has led to the formulation of hybrid mapping methodologies.

2.3.3.2 Hybrid mapping

Hybrid mapping strategies, as depicted in Fig. 2.5 or Fig. 2.6, combine design space exploration (DSE) at design-time with the runtime management to select the best mapping by considering the workload and resource availability. This approach takes the advantages of both static and on-the-fly strategies. The hybrid mapping strategies have been investigated in various other works. According to the survey in [43], hybrid mapping is one of the upcoming trend of mapping technology at run time for modern video applications. This approach includes two steps: design time step and runtime step, which allows to have the benefits both at design time and run time analysis to improve the performance.

Recent work on mapping methodologies of dataflow applications are classified and listed in Table 2.3. As can be seen, different MoCs for the applications are employed. The methodologies target homogeneous (HO) or heterogeneous (HE) MPSoC. The type of platform can be fixed platform or generic platform. The table also reveals the communication model (Comm.) which is used.

Kaushik et al. [89] present a hybrid mapping method for balancing between computation and communication load. Their technique performs clustering at design time and then does actual mapping on NoC platform at runtime to reduce the communication overhead and improve the load balancing. However, this approach can not be directly used in heterogeneous platforms, since the execution time may differ for the same task on different processor types. Moreover, this approach may happen to no solution when the number of clusters is greater than the number of processors in the platform.

Table 2.3: Various approaches for dynamic mapping of multimedia applications

Ref.	MoC	Platform	Comm.	Map.
[58]-2012	SDF	Generic-HE	IPC	Hybrid
[61]-2013	SDF	Generic-HE	NoC	Hybrid
[90]-2013	SDF	Generic-HO	NoC	Hybrid+R
[68]-2012	KPN	Fixed-HO	NoC	Hybrid
[15]-2011	SADF	Generic-HE	x	N/A
[71]-2013	DPN	Generic-HO	Constant	Hybrid
[69]-2013	KPN	Generic-HE	Yes	N/A
[89]-2011	N/A	Fixed-HO	NoC	Hybrid
[86]-2012	KPN	Fixed-HE	Yes	N/A
[106]-2010	x	Fixed-HE	N/A	Hybrid
[70]-2013	KPN	Generic-HO	Constant	Hybrid
[107]-2015	KPN	Generic-HE	Constant	Hybrid+R
Ours	DPN	Generic-HE	Yes	Hybrid+R

Singh et al. [61] propose a hybrid method for efficient mapping of applications on heterogeneous MPSoCs. The approach first performs extensive design time analysis of the set of applications at different resource combination to provide all mapping solutions. This is followed by a runtime strategy to select the best mapping from available mapping solutions subject to available resources and desired throughput. In this approach, the authors consider the communication cost based on NoC architecture.

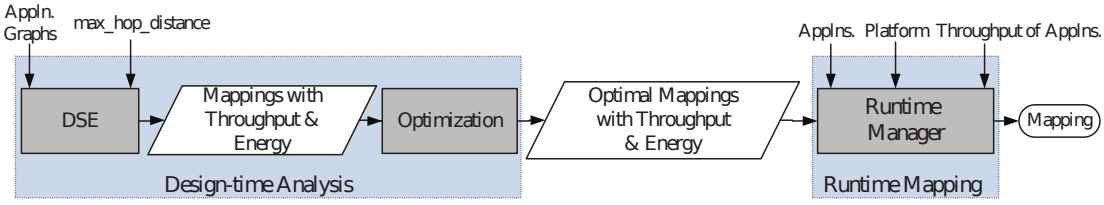


Figure 2.6: Hybrid mapping strategy [61]

Schranzhofer et al. [106] aim to maintain low power consumption over the system lifetime. They compute offline mapping templates and then choose an appropriate pre-computed template online.

Schor et al. [68] propose a scenario based design flow, as shown in Fig. 2.7,

for mapping multi-application on heterogeneous many-core systems. At design time analysis, a set of optimal mapping is computed for using at runtime according to runtime manager and event monitor. However, they use an evolution algorithm, which may degrade an overall performance with large problem size, to optimize the mappings of KPN applications at runtime. Moreover, they need a memory space to store a set of mappings that are individually valid for a subset of scenarios.

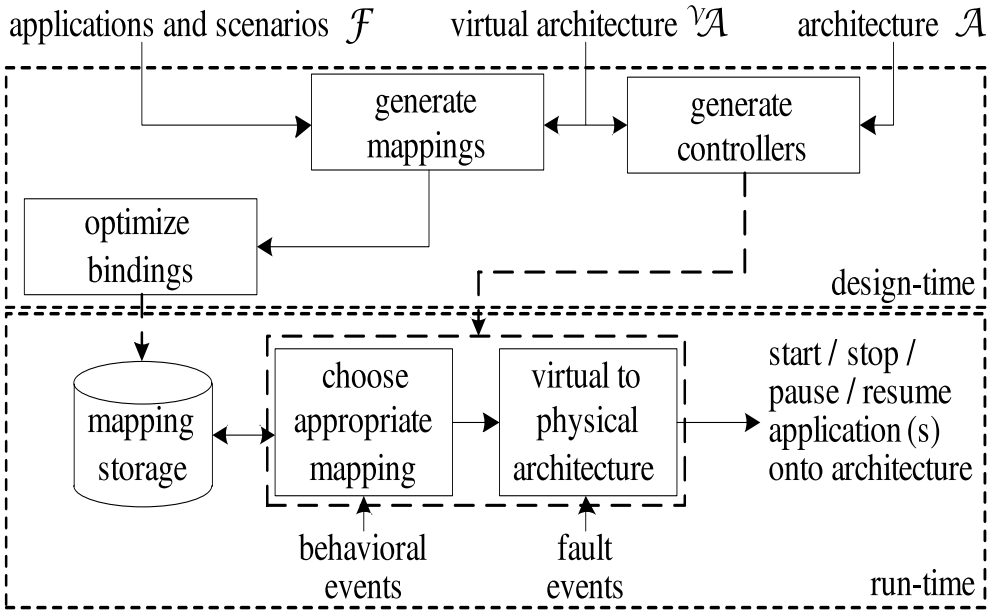


Figure 2.7: Overall mapping optimization approach with design time and runtime component [68]

Castrillon et al. [69] present MAPS framework for supporting multiple dataflow application mapped on heterogeneous MPSoCs. The framework offers different means of performance estimation and provides variety of mapping heuristics, e.g. computation balancing, output rate balancing and simulated mapping. They consider applications as KPN model and architecture platforms containing a list of processing elements and communication primitives. In this research, the user has the freedom to select the heuristic to apply or can guide the tool to try out all possible combinations.

Yviquel et al. [71] can deal with unpredictable behaviors since their approach can produce a high performance in terms of throughput in few milliseconds. They are using the partitioning tool METIS [108] that we present in Sec.3.3.2 and use for comparison. However they consider first a homogeneous multicore architecture and secondly a very specific architecture based on TTA processors.

Quan et al. [70] propose a scenario-based runtime task mapping algorithm

for MPSoCs. They obtain 11.3% performance improvement and 13.9% energy saving compared to just using a static mapping strategy. This algorithm only works with homogeneous platforms.

While there are some efforts in the hybrid strategy trend [61, 68, 89, 70], their analysis results do not address the problem of mapping storage at design time and still ignore dynamical optimization of the mappings at runtime, i.e. keeping the same mapping solution during the execution of certain workload scenario, or apply on specific platforms.

2.3.3.3 The perspective of hybrid mapping with runtime remapping

Hybrid mapping with runtime remapping is a hybrid approach with more adaptive features to allow remapping an application online. Runtime remapping have been proposed in the case of processor failures on NoC [109]. The authors focus on minimizing performance degradation rather than minimizing migration costs.

The approach in [90] presents runtime resource management in that mapping of tasks are changed at runtime. They use the static scheduling information to minimize power consumption under throughput and deadline constraints respectively. However, they use SDF model and consider NoC based many-core architecture, which is not the case for generic architecture.

Quan and Pimentel [107] propose a hybrid mapping including three steps: design time preparation, runtime mapping initialization and runtime mapping customization. This approach allows for remapping an application at runtime. The design time step exploits optimal mappings and store the pre-optimized mappings in system memory for further mapping optimization. At runtime, the mapping initialization process dynamically optimizing the throughput under a predefined energy budget based on the pre-optimal mappings of the corresponding applications stored on the system when the new workload scenario emerges. Afterwards, mapping customization is performed to further improve the performance of the mapping during the execution of a certain workload scenario. In this study, the algorithm tries to keep the migrated actors as low as possible to reduce the migration overheads. Although hybrid mapping with runtime remapping becomes a promising method for solving mapping problem of dynamic dataflow applications on heterogeneous multiprocessor platforms, they still lack of in-depth exploration such as the trade-off between the design time and runtime step, the impact of the migration cost when allowing remapping at runtime.

2.4 Conclusion

To sum up, there are two main methodologies to tackle the mapping problem of dataflow applications on embedded MPSoCs: static mapping and dynamic/run-

time mapping. The methodologies are analyzed to highlight their advantages as well as drawbacks. Most of the mapping methodologies reported in the state of the art fall under static mapping [43, 98]. Nevertheless, they can not handle dynamic behaviors of the modern video applications. Recently, hybrid mapping method becomes an emerging trend in solving the mapping problem in embedded multimedia domain. With no doubt, the advantages of hybrid method make it become a promising strategy in the field of mapping methodologies. However, most of these approaches consider that all application behaviors must be known entirely at design time and analysis results can be applicable only to the analyzed platform. This kind of approach usually searches all possible mapping solutions at design time and then makes a decision of mapping at runtime based on the solution at design time. Thus, this also imposes challenge to investigate efficient exploration strategies that should overcome the *exploration time bottleneck* and *high memory usage* for storing all mapping solutions at design time. As a result, the design time analysis needs to be repeated when the application set or platform changes [43]. Moreover, they consider all tasks/actors in a dynamic application are fully Kahn Process Network, which might not be hold in modern multimedia applications, such as video codecs. Actually, most signal processing applications are far from being entirely dynamic [36]. In other words, a part of signal processing application is static behaviors.

Existing hybrid strategy typically assumes that the mapping of an application is not changed at runtime after launched. For example, many approaches do not support *the handling of dynamic behaviors such as a dynamic changing in workload and actor migration in a platform* that is not known at design time. Hence its capability to support the dynamic system behavior is limited since it may lead to higher probability of mapping failure as well as inefficient resource usage.

Since their nascent development and lack of in-depth examination in the literature, the hybrid methodologies have some open issues that need to be addressed and solved in the future. To the best of our knowledge, the existing runtime mapping heuristics in the state of the art are not suitable to apply for mapping of RVC-CAL applications on heterogeneous MPSoCs. In this context, a main challenge is to propose a solution that can be fast enough to execute dynamically the actor mapping on heterogeneous architectures. It may first make a decision at the reception of the video based on default profiling data (e.g. last decoding) without introducing any significant latency. Then the objective is to recompute at runtime mapping decisions according to observations of real execution times. Selection (actor mapping) is one the main steps of self-adaptive systems with observation and configuration.

To overcome the limitation of those hybrid mapping techniques, we propose a novel hybrid mapping algorithm, as depicted in Fig. 2.8, for heterogeneous

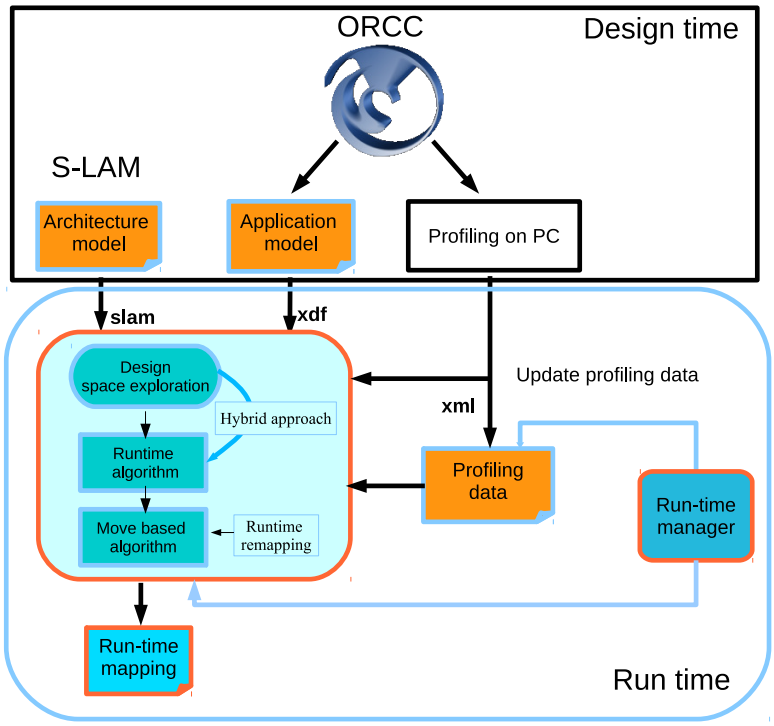


Figure 2.8: Our mapping methodology

platforms, which avoids the exploration time bottleneck and memory footprint. At runtime, our algorithm can adapt to dynamic behaviors both in application model and architecture model and allows runtime remapping of an actor with move based algorithm. Most of the existing runtime mapping techniques face the challenges in load balancing, both computation and communication. However, the mapping techniques reported in literature do not have an *analytical communication model* that can apply either NoC or Bus based architecture. Moreover, they lack of considering *the impact of migration cost* at runtime which may have an important contribution to overall performance. In our approach, we consider load balancing combined with our analytical communication model. Further, we propose a move based algorithm for remapping actor and also take the migration cost into account. These features make our approach different from others.

3

Communication Model Based Embedded Mapping

Mapping a dataflow application onto a heterogeneous multiprocessor platform cannot longer be static. It has to adapt dynamically depending on the data and on the communication between the computation cores. This is typically the case for mobile devices that run multimedia applications. This chapter presents an algorithm fast enough to be executed at run-time. In addition to computation cost, our approach relies on a communication model to estimate the delay for transmitting data. The algorithm is compared with METIS tool for random dataflow graphs and two video decoders, MPEG4-SP and HEVC, considering heterogeneous multiprocessor platforms composed of 4 to 8 processors and 6 accelerators. Results on a virtual Zynq platform show that our algorithm is about 40x faster than METIS tool for the same throughput (frames per second) on a platform with 8 processors and 6 accelerators.

Our contribution is a solution that makes mapping decision possible with high video frame rates. We also consider an heterogeneous architecture that can be reconfigurable in case of FPGA with an architecture model based on softcores augmented with hardware accelerators. The different components of the heterogeneous platform need to exchange data through a communication media (a bus or a NoC). This communication is likely to become the bottleneck and should be taken into account by the mapping algorithm.

The contributions of this chapter are:

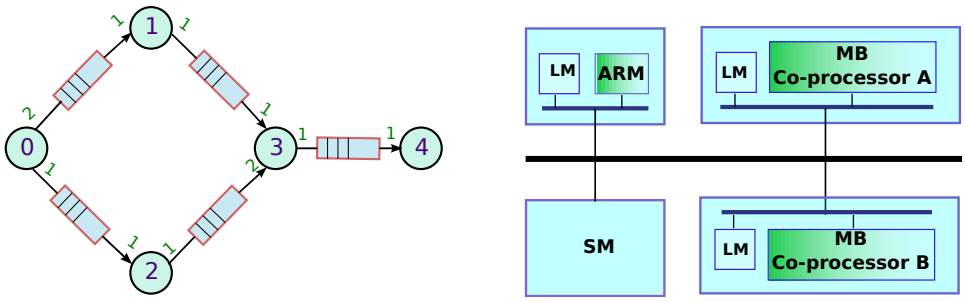
- A communication model for estimating the latency of the data on the communication media.
- An algorithm for mapping dataflow actors onto heterogeneous MPSoC.
- Experimental results on a virtual Zynq platform for randomly generated dataflow graphs and MPEG4-SP and HEVC video decoder applications.

In the next sections, we give the formulation of the problem, then we introduce our algorithm. Finally we present extensive results that demonstrate the viability and the efficiency of our approach.

3.1 Problem definition

In this section, we define the problem of runtime mapping of dataflow actors on heterogeneous multiprocessor platforms.

Given a dynamic dataflow application, which is based on different MoCs, we assume that profiling mechanisms are available on the target platform so that we can estimate and adapt at runtime the number of tokens produced and consumed by each actor as well as the execution time of actors. We then aim to map the dataflow actors onto various computation and communication resources with the objective to optimize the application throughput. Finding a high-quality mapping solution for such a dynamic dataflow application on heterogeneous platform is an NP complete problem. This is why we consider the dataflow application at high-level description and propose a heuristic approach in order to produce an efficient solution close to optimal in few milliseconds. In addition, we have to take load balancing between computation and communication into account.



(a) Dataflow graph (b) Architecture platform

Figure 3.1: Dataflow graph and architecture platform specified in XDF and SLAM files respectively

3.1.1 Application model

A dataflow application is specified with a high-level description model. It is defined as a directed graph $G = (V, E)$, where the vertex set V is a set of actors and the edge set E is a set of FIFO channels. Fig. 3.1a shows an example of dataflow graph. An actor is a computational entity with interfaces (input and output ports), internal states and parameters. Actors can interact by exchanging

data (called tokens) through channels, unidirectional First In - First Out (FIFO) buffers. During a process, the actor consumes input tokens, produces output tokens and changes its internal state. The actors may be compliant with different models of computation (MoC) and so consume and produce a fixed or variable number of tokens, additionally the execution time can be more or less variable. In practice the graph is specified in an XDF file, which is an input of our mapping algorithm as shown in Fig. 3.3.

3.1.2 Architecture model

The specification of the heterogeneous platform is based on an architecture model that can include different types of processors with or without hardware accelerators. We consider an architecture model, where each processor has its local memory (LM) and communicates with other processors through several buses and shared memories (SM). In this chapter, we consider a heterogeneous platform where processors communicate via one bus and one shared memory. This architecture includes one ARM processor and multiple Microblaze processors, which have different hardware accelerators and frequencies as well. This choice allows to make a fair comparison with METIS tool that supports only this kind of architecture platform (i.e. an implicit bus-based architecture). Fig. 3.1b gives an example of a simple architecture platform with one ARM processor (that can run the mapping algorithm), two Microblaze (MB)) with one hardware accelerator and all processors communicate through one bus and one shared memory (SM).

3.1.3 Communication model

The latency of a communication medium (bus, NoC) evolves with contention occurrences. But access conflicts cannot be estimated with simulations when fast partitioning is required. That is why we propose to use analytical models that can be updated online if necessary. These models are based on what is observed in the domain of NoC, where the latency is usually measured for different injection rates.

In our approach, the communication model gives the relationship between use-rate and latency since the use-rate can be estimated at run-time according to mapping decisions or estimates. We propose a generic and parametric communication model as in Fig. 3.2. The aim is to have a unique model that can fit with different communication standards, which are supported in the platform. A model is based on two linear functions y_1 and y_2, the intersection between the two curves correspond to the saturation threshold (U_T). These functions can express the communications of a NoC, i.e. the red curve in Fig. 3.2, or a bus, i.e. the green line where $y_1 = y_2$. In practice the parameters of y_1 and y_2 can be adapted

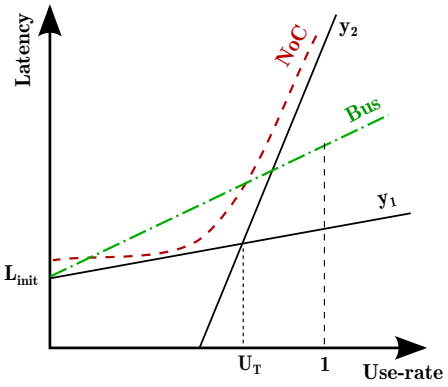


Figure 3.2: Communication model

online according to real profilings as depicted in Fig. 3.3.

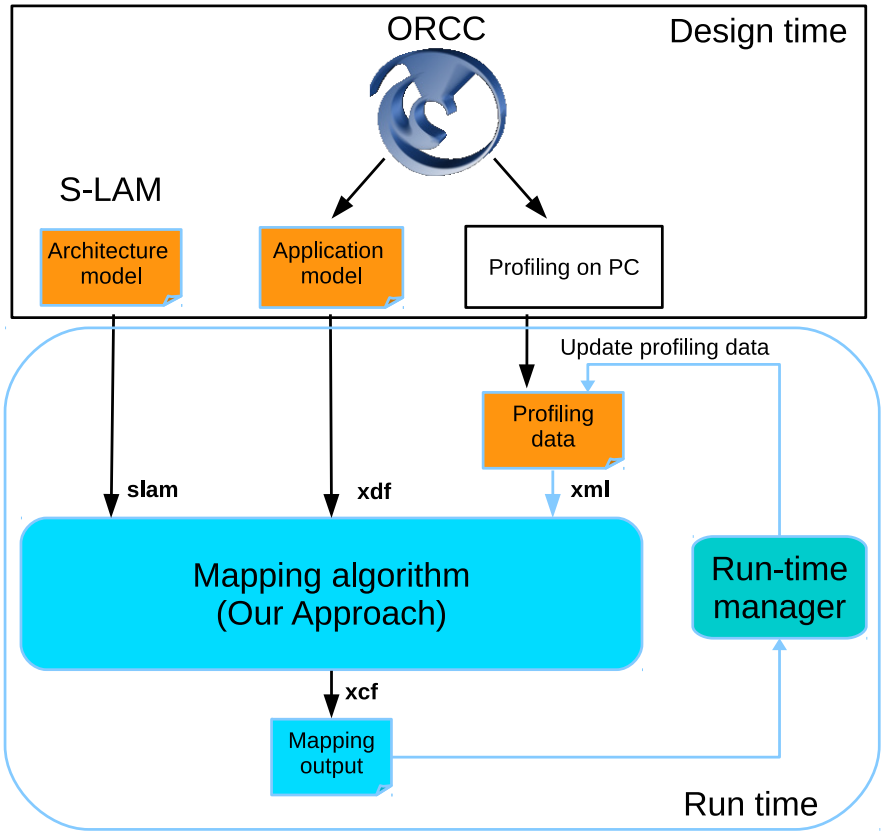


Figure 3.3: Overall flow of our heuristic mapping algorithm

3.2 Heuristic mapping algorithm

This section is organized as follow. First, we define some metrics that are necessary to evaluate the mapping quality of dataflow actors between our algorithm and METIS. Then, we present our approach, called GB4M2, for solving the mapping problem of dataflow actors on heterogeneous multiprocessor platforms.

3.2.1 Evaluation metrics

Table 3.1 presents all the parameters and variables used to formalize our mapping algorithm.

Table 3.1: Parameters and variables used for the mapping algorithm

Parameter	**Definition**
Application graph	
NbActor	Number (Nb) of actors
NbFIFO	Nb of FIFO channels
Architecture graph	
NbProcessor	Nb of processors
$MemCode_j$	Code memory size of processor j
$MemData_j$	Data memory size of processor j
$Freq_j$	Frequency of processor j
Profiling data	
R_i	Nb of firings of actor i per iteration
W_1^i	Total computation time of actor i
W_2^i	Instruction code size of actor i
W_k^e	Amount of data on channel k
T_{ij}	Execution time of actor i on processor j
T_c	Communication time per token
P_k, C_k	Nb of token producers and consumers on channel k
METIS	
NbCon	Nb of constraints
$Asize_i$	Amount of data consumed by actor i
$P[i]$	Nb of the partition of actor i
$Nadj_i$	Nb of adjacent actors other than $P[i]$

The total computation cost of actor i is the sum of computation times of this actor on all processors on the heterogeneous multiprocessor platform. It means

that all processors can execute all actors, otherwise the average computation cost can be used instead. This parameter is computed as the metric for the first sorting in our heuristic algorithm:

$$W_1^i = \sum_{\forall j} R_i * T_{ij} \tag{3.1}$$

The amount of data to be sent to actor i^{th} is the sum of all incoming data of this actor:

$$Asize_i = \sum_{\forall k:dst(k)=a_i} W_k^e(src, dst) \tag{3.2}$$

The amount of data on channel k^{th} stands for the number of tokens, which transfer between the source actor and the destination actor of this channel:

$$W_k^e(src, dst) = R_{src} * P_k + R_{dst} * C_k \tag{3.3}$$

Instruction memory usage of each processor is the total code size of all actors which are mapped on this processor:

$$memUsage_j = \sum_{\forall i:P[a_i]=j} W_2^{a_i} \tag{3.4}$$

Computation time of each processor is the sum of the execution time of all actors which are mapped on this processor:

$$compT_j = \sum_{\forall i:P[a_i]=j} R_i * T_{ij} \tag{3.5}$$

Communication time of each processor:

$$commT_j = \sum_{\forall i:P[a_i]=j} Asize_i * Nadj_i * T_c \tag{3.6}$$

Total communication volume is the sum of all communication time of each processor:

$$totalV = \sum_{\forall j} commT_j \tag{3.7}$$

Period of each processor:

$$Period_j = compT_j + commT_j \tag{3.8}$$

Throughput:

$$Th = \frac{1}{max(Period_j)} \tag{3.9}$$

3.2.2 GB4M2 Algorithm

Our solution is based on a greedy method, this choice is motivated by the objective of being fast to produce a solution while considering all the constraints of the problem. In contrast to conventional existing mapping strategies, our method differs in following aspects: i) it relies on a factor called alpha, which represents a ratio of processor cycle budget considered during the first step (computing-oriented) mapping; ii) it takes into account both computation and communication workloads; iii) it performs both actor and data mappings. The algorithm is composed of 3 steps: initial, computation and communication phases.

An overview of the heuristic environment is presented in Fig. 3.3 and the algorithm is presented in Algo. 1. The algorithm needs the information from the application model, architecture model and profiling data. From these input information, the algorithm produces a mapping solution.

3.2.2.1 Initialization phase

Since the dynamic behavior of our applications makes them unpredictable at design-time, the GB4M2 is based on low-cost profiling analysis of the execution. Consequently, we assume that profiling mechanisms are available on the target platform. The initialization is based on a default profiling that provides the execution time of each actor on different processors. Then, the actor is selected with the processor where it has a minimum execution time. This initial phase helps to compute the initial period (line 2) of each processor without considering the communication time. However, this step may lead to a solution where all actors will be mapped on their best processors. It means that some processors are not used while others have a lot of actors. We expect a load balancing among processors so we calculate an initial average period and use this value as a processor budget for all processors.

3.2.2.2 Computation phase

Next, we introduce a factor α, which represents a ratio of the processor budget. This factor can vary in the range of $]0, 1]$. In this step, the algorithm performs a sorting of all actors in a descending order based on the value of their total computation cost ($W_1^{a_i}$) Equ. 3.1. A fast sorting algorithm (a bubble sort in the current version) is used. Then, the algorithm picks up the first actor i in the sorting list LA1, i.e which presents the largest computation cost. Next, the

Algorithm 1 GB4M2 Algorithm

Input: application graph, architecture graph, profiling data
Output: mapping of actors and data

1: // initialization step
2: $Period\_init \leftarrow select\_min\_Tcomp(nbActor, nbProcessor)$
3: // computation step
4: $LA1 \leftarrow Actor\_sorting\_Tcomp(nbActor)$
5: **while** $procUse_j \leq \alpha$ **do**
6: $k \leftarrow index\_first(LA1)$
7: $map\_actor\_Tcomp(k, nbProcessor)$
8: update $procUse_j$
9: $k \leftarrow k + 1$
10: **end while**
11: // communication step
12: $period\_estimate \leftarrow max(procUse_j)$
13: **if** $k < nbActor$ **then**
14: $LA2 \leftarrow actor\_sorting\_Tcomm(k, in\_out\_comm(k))$
15: $tcommG \leftarrow \sum_{k \in index(LA2)}(in\_token(a_k) + out\_token(a_k))$
16: $latency \leftarrow Linit$
17: $Tcomm \leftarrow 0.5 * latency * \beta * tcommG$
18: $Cuse \leftarrow Tcomm/period\_estimate$
19: **end if**
20: **for** $i = k$ to $nbActor$ **do**
21: update $latency$
22: **for** $j = 0$ to $nbProcessor - 1$ **do**
23: $tcommGuess_j(i, nbProcessor)$
24: $periodGuess_j(i, nbProcessor)$
25: **end for**
26: $map\_actor\_Tcomm(min(periodGuess_j), nbProcessor)$
27: update $procUse_j$
28: update $Tcomm$
29: update $Cuse$
30: **end for**

algorithm selects the best processor according to the minimum T_{ij}. In case the memory capacity is not satisfied, then the algorithm selects the next minimum execution time of this actor on the remaining available processors. After the first actor is mapped (line 5), the algorithm updates the processing use (line 6) of the selected processor. The algorithm applies the same procedure for the next candidate in the first sorting list until the processing use of all processors is greater than α, which is a percentage of the processor budget.

3.2.2.3 Communication phase

The last step starts with the remaining actors which are not mapped. Then, it applies the second sorting for all unmapped actors. This sorting is based on total incoming and outcoming data for each actor. The idea is to take the communication cost into account in this phase. Hence, the algorithm takes the first actor in the second sorting list and consider all the connections with other actors in the network. If this actor and its adjacent actor are mapped on the same processor, the communication time is zero. In order to deal with unknown information when we estimate the use-rate of the bus (% usage of available bandwidth), we introduce a factor β that represents the ratio of remaining data transfers associated to unmapped tasks that will use the communication media (e.g. bus). Then the latency is estimated with the communication model (Fig. 3.2) and the algorithm makes a decision of mapping. Then, the algorithm applies the same procedure for the next unmapped actor in the second sorting list.

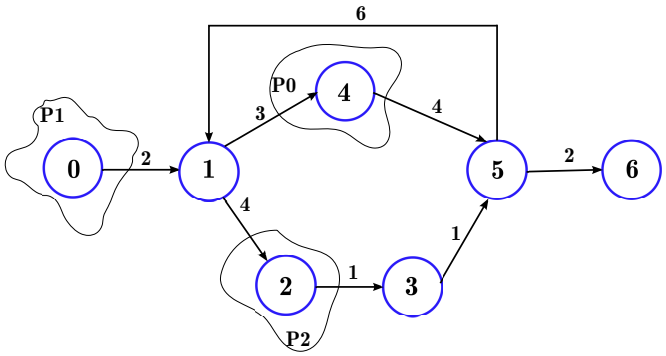


Figure 3.4: Example of a graph application

Fig. 3.4 gives an example of a graph application, which includes 7 actors $\{a_0, a_1, \ldots, a_6\}$. The target platform has three processors $\{P_0, P_1, P_2\}$. In the computation phase, actor a_0, a_2 and a_4 are mapped on P_1, P_2 and P_0 respectively. Then, we have the remained actors with the second sorting list LA2=$\{a_1, a_5, a_3, a_6\}$. In order to map actor a_1, the algorithm takes into account all communication (line 21) between a_1 and its adjacent actors in the graph

application. At this step, a_5 is not mapped so $tcommGuess_j(index(a_1), 3) = latency * \{(4+2+\beta*6), (4+3+\beta*6), (2+3+\beta*6)\}$. From this information, the algorithm computes $periodGuess_j$ (line 22). Then, the algorithm decides to map actor a_1 on a processor where a_1 has the minimum value of periodGuess. Next, the algorithm updates the $procUse_j$, $Tcomm$ and $Cuse$ (line 25 to 27). All actors in $\{a_5, a_3, a_6\}$ follows the same procedure as actor a_1.

3.3 Experimental results

In this section, we present the results of our experiments about the actors mapping over the heterogeneous multiprocessor platforms. We apply our GB4M2 algorithm for a static dataflow as well as a dynamic dataflow. In order to evaluate our heuristic approach more thoroughly, we make the comparison with METIS tool [108] both in terms of throughput and in terms of solving time. All dataflow applications are implemented in two parts. For the static dataflow applications, we use SDF3 tool [37] to generate SDF benchmarks. For the dynamic dataflow application, we examine our methodology with MPEG4 Part 2 Simple Profile (MPEG4-SP) and High Efficiency Video Coding (HEVC). Implementation of MPEG4-SP and HEVC are carried out with the Orcc [31] tool as in [71]. In order to make a fair comparison between our heuristic approach and METIS, we transform those application benchmarks into METIS' graphs and use the same evaluation metrics for both METIS and our heuristic. METIS supports a function to measure its solving time and we also use this function to get our solving time. The System-Level Architecture Model (S-LAM) [38] is employed for the specification of different sets of heterogeneous multiprocessor platforms. The number of processors varies from 4 to 8, combined with several hardware accelerators. All experiments were conducted in Cadence virtual system platform (VSP) [110] simulation and Linux environment on HP ProBook with core i5-2520M CPU@2.5GHz and 8GB memory.

3.3.1 Simulation on Cadence virtual system platform - VSP

The algorithm was implemented for Xilinx Zynq-7000 and simulated through the Cadence Virtual System Platform (VSP) [110]. This tool simplifies the development of virtual prototypes by providing an automated modeling and faster hardware/software debugging framework through SystemC-TLM models, with enough accuracy to replace hardware for development purposes. The platform is composed of five main components: i) 32-bit processor Cortex-A9MP of Imperas OVP [111]; ii) four cache-coherent memories; iii) bank of registers; iv) 32-bit bus;

and v) peripherals. The system supports embedded Linux operating system. The cross-compile tool chain is also provided by Imperas OVP.

One key aspect in embedded systems is memory and the binary size of our algorithm is only 95.5 kB on ARM processor whereas the binary size of METIS is 692 kB.

3.3.2 METIS - Graph partitioning for heterogeneous multiprocessor architectures

METIS is the well-known and successful graph partitioning tool. It has been developed at the University of Minnesota and distributed as an open source. The algorithms implemented in METIS are based on the recursive-bisection, multi-level k-way, and multi-constraint partitioning schemes [112]. This tool quickly produces high quality partitionings for a wide variety of irregular graphs [113]. Since METIS supports multi-constraint partitioning and allows for target partition weights, it can be adapted to compute partitionings that balance the computations on heterogeneous architectures. Furthermore, the partitioning objective in METIS is to minimize the edge-cut so the communications among the processors are also minimized.

3.3.3 Results with SDF benchmarks

We evaluate our heuristic approach on a variety of random SDF graphs. There are 14 SDF graph benchmarks with a number of actors that varies from 40 to 160. Token transfers on each channel, in/out degrees of actors are randomly created within minimum and maximum bounds and with different average and variance. Additionally we generate an array of various execution times for each actor on each processor. We compare the throughput and the solving time between our heuristic and METIS. The results of the experiments show that the throughput decreases when the number of actors increase. Fig. 3.5 and Fig. 3.6 depict the throughput comparison between METIS and GB4M2. The value of alpha in GB4M2 varies from 0.2 to 1. It has to be noted that setting an alpha value of 1 in the algorithm means that the algorithm never moves to the step where communication is taken into account, so it performs *Computation Phase* only.

As it can be seen, GB4M2 achieves better throughput in the majority of benchmarks. This is especially true with 8 processors and alpha larger than 0.4. In particular, when the number of actors is 120 and alpha is 0.4, GB4M2 achieves 54% improvements in terms of throughput. During the experiments, we observe that when the alpha value is equal to 0.4, the bandwidth requirement is close to the result from METIS while maintaining a higher throughput than METIS. Therefore, using alpha as the guideline for mapping not only achieves

better trade-off between throughput and bandwidth, but also provides better throughput than METIS. Fig. 3.7 illustrates the speed-up in terms of solving time. As it can be seen, GB4M2 has a significant speed-up of at least one order of magnitude in terms of solving time for all benchmarks. Even with the large number of actors, e.g. 160, we got 48.89x faster than METIS for 4 processors and 31.51x faster than METIS for 8 processors. The speedup tends logically towards decreasing since METIS is designed for dealing with thousands of nodes (i.e. actors). Finally another interesting result given in Fig. 3.8 and 3.9 is the used bandwidth with the different solutions. It illustrated how the α parameter is used to find a balance between throughput and bandwidth use. It also shows how the aggressive and unique minimization of the bandwidth (case of METIS) can negatively impact the performance, this strategy is counterproductive when the bandwidth is available and that's why an online estimation of the communication cost is necessary.

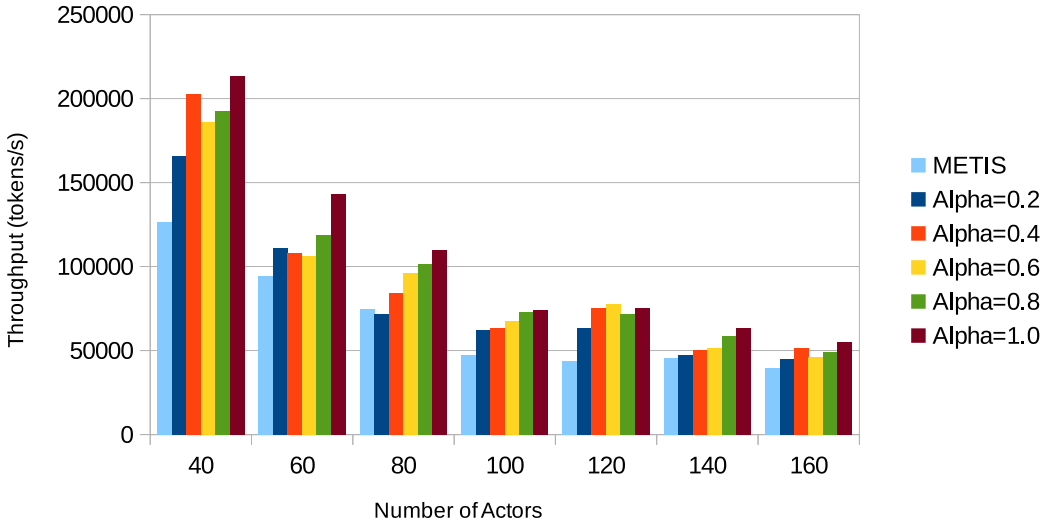


Figure 3.5: Throughput: METIS vs GB4M2 - SDF3 - 4 processors

3.3.4 Results with real video applications

We use two video decoders, which are developed by the RVC group: MPEG4-SP and the new MPEG-H Part 2 (known as HEVC/H.265). Table 3.2 describes the properties of MPEG-4 and HEVC, as well as the number of actors and FIFO channels. The C code of these decoders are generated with Orcc tool [31]. Options in the tool can allow for generating additional code for intrusive profiling. All

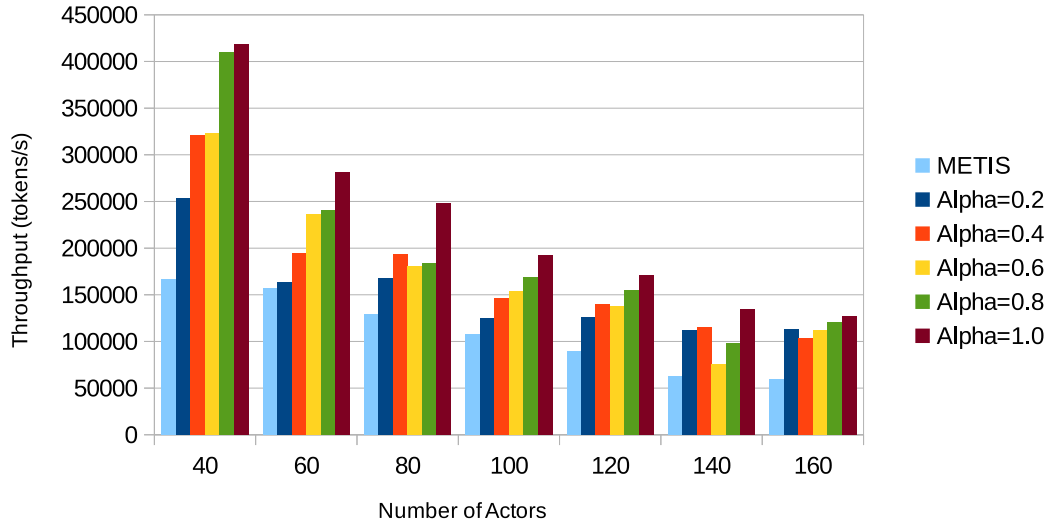


Figure 3.6: Throughput: METIS vs GB4M2 - SDF3 - 8 processors

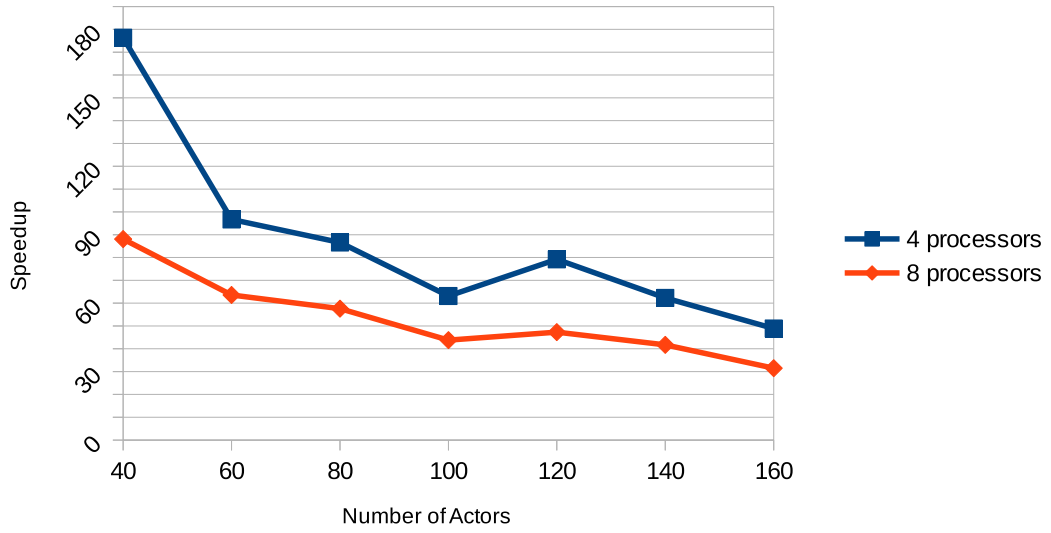


Figure 3.7: Speed-up in terms of solving time - SDF3

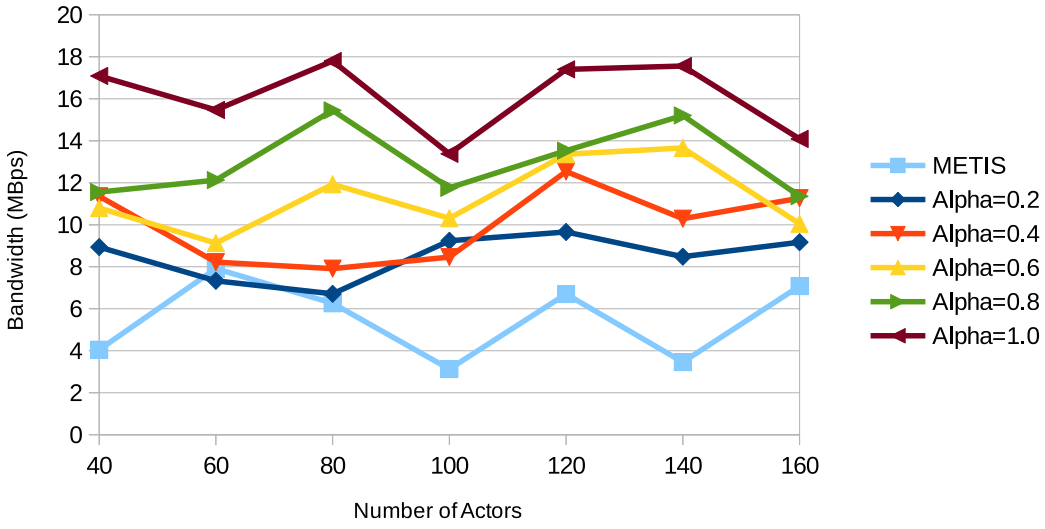


Figure 3.8: Bandwidth requirements - SDF3 - 4 processors

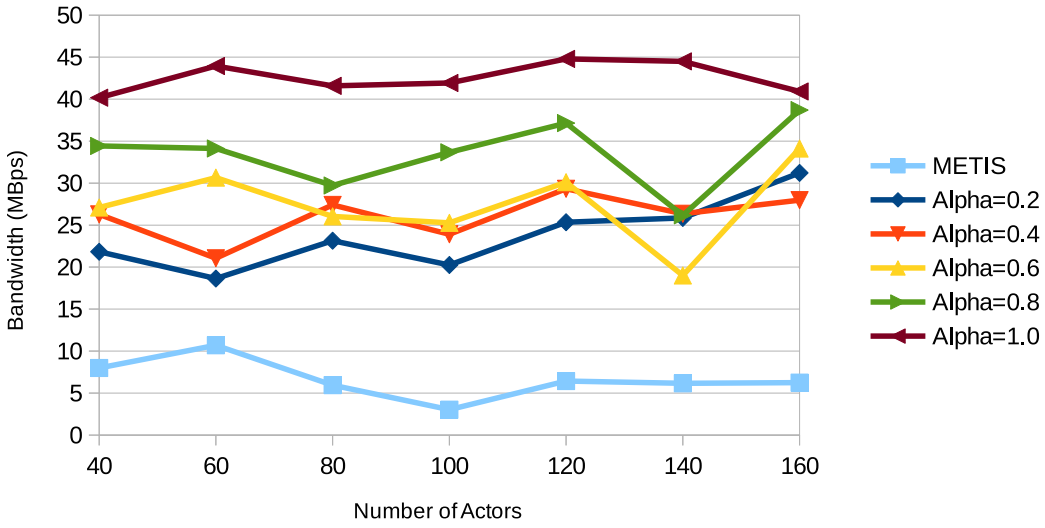


Figure 3.9: Bandwidth requirements - SDF3 - 8 processors

actors are classified [36] as different MoCs such as SDF, CSDF, KPN and DPN. All experiments made from different video sequences. In particular, *Foreman* and *Hit* videos are used as input videos of MPEG-4 decoder. The resolution of all MPEG-4 videos are 352x288, which corresponds to CIF format. *Kristen and Sara*, *Four people* and *Johnny* are input videos of HEVC decoder. All HEVC video sequences have 1280x720 resolution.

The first profiling is performed on a desktop computer. For these experiments, we specify the size of the FIFO channels used for communication between 512 and 8192 bytes. We introduce some usual hardware accelerators that can provide significant speed-up when associated with Microblaze (MB) softcore on Xilinx FPGAs. Our objective is the runtime mapping of actors on different heterogeneous architecture corresponding to different possible terminals, so we consider in this example four different platforms based on 1 ARM and 7 MB processors. By using S-LAM modelling, we specify 4 different heterogeneous platforms detailed in Table 3.3.

Table 3.2: Properties of different tested MPEG video decoders

Decoder	Profile	YUV	#Actors	#FIFOs
MPEG-4 Part 2	SP	yes	41	104
HEVC/ H.265	Main	no	27	185

Table 3.3: Accelerators used in different platforms

Platform	MB1	MB2	MB3	MB4	MB5	MB6	MB7
7.1	Merger	IDCT	Parser	Inter	IQ+IAP	Add	IDCT
7.2	IQ+IAP	IDCT	Parser	Inter	IDCT	Merger	IDCT
7.3	Parser	xIT	Intra	Inter	Merger	DPB	xIT
7.4	Merger	xIT	Intra	Inter	xIT	Parser	xIT

3.3.4.1 The need of run-time mapping

According to profiling information, we observe the percentage differences in terms of workload of each actor. The workload of an actor is defined as the ratio of the computation time in given time interval. Table 3.4 and 3.5 show percentage differences in workload between two different input video sequences of MPEG-4 decoder and HEVC decoder respectively. These tables just pick up several actors in the decoder, which present more different workload. Then, we compare the

Table 3.4: Example of percentage difference in terms of workload of different MPEG4 video sequences

Actor	Foreman	Hit	% difference
decoder_texture_U_idct2d	64	46	32.73
decoder_texture_V_IQ	49	38	25.29
decoder_texture_V_idct2d	69	38	57.94

Table 3.5: Example of percentage difference in terms of workload of different HEVC video sequences

Actor	Kristen & Sara	Four People	% difference
HevcDecoder_xIT	615	546	11.89
HevcDecoder_InterPred_Inter_y	1642	1392	16.48
HevcDecoder_SAOFilter_Sao_U	193	214	10.32

actor mapping result of different video sequences. The results show that some actors in the decoder have different mapping on the same target platform. This is true both in MPEG-4 and HEVC decoders. For example, we get 9.76 % different actor mapping between *Foreman* and *Hit* videos. With HEVC decoder, the difference in actor mapping is 33.33 % between *Kristen and Sara* and Four People videos. We also get 25.93 % differences when we compare the actor mapping result of *Four People* and *Johnny* videos. This difference can be explained by the fact that we have some dynamic actors both in MPEG4-SP and HEVC decoders. These differences clearly demonstrate the need of run-time mapping for dynamic dataflow actors.

3.3.4.2 MPEG4-SP and HEVC decoder

The results of both experiments in terms of throughput (frames per second) are shown in Table 3.6. We almost have better throughput in comparison with METIS. For MPEG-4 decoder, the GB4M2 always achieves higher throughput with the platform 7.x than METIS. We also have the speed-up when we compare the throughput of the HEVC applications between GB4M2 and METIS with the platform 7.3. However, the GB4M2 has slightly lower throughput with platform 7.4 than METIS. This can be explained that MPEG-4 decoder has well implementation while HEVC decoder is more complexity and still being developed. As indicated in Table 3.7, we obtain a significant speed-up in terms of solving time for different sets of heterogeneous platform. GB4M2 took only 4.54 ms for

Table 3.6: Result with MPEG4-SP and HEVC mapped on 8 processors in the heterogeneous platform

Input video	Platform	GB4M2	METIS	Speed-up
MPEG4-Foreman	7.1	51.37 fps	51.06 fps	1.01x
	7.2	50.13 fps	49.40 fps	1.01x
MPEG4-Hit	7.1	49.26 fps	46.70 fps	1.05x
	7.2	48.00 fps	46.52 fps	1.03x
HEVC-Kristen	7.3	10.38 fps	6.84 fps	1.52x
	7.4	9.71 fps	9.78 fps	0.99x
HEVC-FourPeople	7.3	13.16 fps	9.13 fps	1.44x
	7.4	9.09 fps	10.25 fps	0.89x
HEVC-Johnny	7.3	14.38 fps	10.29 fps	1.40x
	7.4	9.63 fps	9.87 fps	0.98x

finding the mapping solution of the *Foreman* video sequence. The results show that we offer a better trade-off between performance (throughput) of the application and the solving time of the GB4M2. We conclude that our GB4M2 is more efficient at run-time than METIS. These examples also demonstrate that in case of a reconfigurable architecture, our solution can be used to decide conjointly and at runtime, the best platform and the best actor mapping for the dataflow application to be executed.

3.4 Conclusion

In this chapter we present a new mapping algorithm for heterogeneous multiprocessor architectures that includes a simple analytical communication model. Our solution is extremely fast and can be used at runtime for the mapping of actors of dynamic dataflow applications for which, by definition, no optimal solution can be found offline. Reconfigurable video coder is a typical domain where such a solution is required. We have compared our algorithm with the fast METIS partitioning algorithm. We demonstrate on a large set of representative synthetic graphs as well as on MPEG4-SP and HEVC real life examples that our algorithm is at least one order of magnitude faster (hundred of μs) on an ARM embedded processor, while producing better results when the alpha parameter is larger than 0.4. Our approach is between two extreme partitioning techniques: solutions that consider bandwidth as available but do not take it into account and methods like

Table 3.7: Solving time

Input video	Platform	GB4M2	METIS	Speed-up
MPEG4-Foreman	7.1	4.54 ms	200 ms	44.08x
	7.2	3.74 ms	190 ms	50.76x
MPEG4-Hit	7.1	4.25 ms	190 ms	44.66x
	7.2	3.63 ms	190 ms	52.34x
HEVC-Kristen	7.3	5.33 ms	130 ms	24.39x
	7.4	4.99 ms	120 ms	24.05x
HEVC-FourPeople	7.3	4.88 ms	130 ms	26.64x
	7.4	4.99 ms	120 ms	24.05x
HEVC-Johnny	7.3	4.88 ms	120 ms	24.59x
	7.4	5.10 ms	140 ms	27.45x

METIS that minimize communication without taking advantage of the available bandwidth to improve the throughput.

Considering the evolution towards highly variable dataflow applications based on an increasing impact of dynamic actors, we must target at runtime the best matching between dataflow graphs and heterogeneous multiprocessor platform. Thus the mapping must be dynamically adapted depending on data and on communication loads between the computation cores. This is typically the case for mobile devices that run multimedia applications.

4
Move Based Algorithm

The problem of mapping a dataflow application, e.g. a network of computational actors, on a multiprocessor platform can be modeled as a problem of partitioning where the cells are the dataflow actors and the partitions are the processors. While the benefit of executing a computational part by one processor rather than another one is usually well shown, the migration overhead is also usually not considered. This chapter presents a dynamic mapping algorithm that is performed at runtime, based on a single-move possibility that jointly considers the cost and benefit of possible migrations. The method is first applied on a set of randomly generated benchmarks with different features and different scenarios. Then it is applied to a MPEG4 simple profile video decoder with different input sequences. The results systematically show that the runtime mapping significantly improves the initial mapping. It is fast enough to be executed at runtime in order to track the best mapping according to data variations. The other observation is that not considering the migration cost of the new mapping could lead to worst performance than the original one.

In comparison with the state of art, the contributions in this chapter can be stated as follows:

- A fast solution at design time step of hybrid mapping methods while solving the problem of storage requirements.

- An analytical communication model for estimating the delay (latency) of the data on the communication media. This communication model is flexible since it can apply either NoC or Bus based architecture.

- A novel hybrid algorithm, namely Move Based Algorithm, compliant with Bus and NoC models takes both computation cost and communication cost while allowing remapping dataflow actors at runtime onto heterogeneous MPSoC. This algorithm supports the adaptivity at runtime and also takes migration cost when doing the remapping actor.

- Dealing with modern multimedia applications which are based on RVC-CAL framework.
- Simulation results with our runtime scenario based simulation for both randomly generated dataflow graphs and MPEG4-SP applications

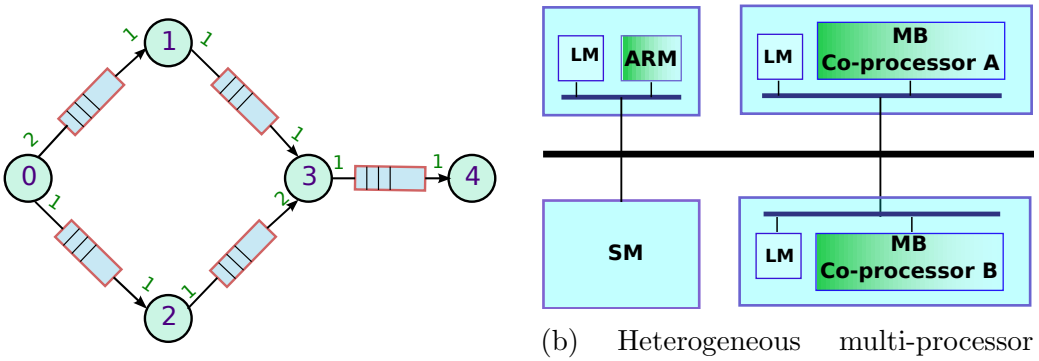
The remainder of this chapter is organised as follows. Section 4.1 provides some definitions of the problem. Next, we present a detailed description of our hybrid mapping algorithm in Section 4.2. Our proposed method is composed of two main steps. Firstly, the design time gets some information from the application model, architecture model and profiling information. In this step, we just need to find a processing budget as explained in Subsection 4.2.2. By doing this way, we solve the problem of exploration time and memory footprint as mentioned above. At second step, we split into two phases, runtime mapping initialization and runtime remapping to further improve the performance. Section 4.3 introduces the experimental environment and presents the results of our experiments. Section 4.4 concludes the chapter.

4.1 Problem definition

In this section, we define the problem of runtime mapping of dataflow actors on heterogeneous multiprocessor platforms. Given a dynamic dataflow application, which is based on different MoCs, we assume that profiling mechanisms are available on the target platform so that we can measure at runtime the number of tokens produced and consumed by each actor as well as the execution time of actors. We then aim to map the dataflow actors onto various computation and communication resources with the objective to optimize the application throughput. Finding a high-quality mapping solution for such a dynamic dataflow application on heterogeneous platform is an NP complete problem. This is why we consider the dataflow application at high-level description and propose a heuristic approach in order to produce an efficient solution in few milliseconds. In addition, we have to take load balancing between computation and communication into account.

4.1.1 Application model

In this chapter, we target the multimedia application domain. We use different MoCs to specify the application. An application is represented as a directed graph $G = (V, E)$, where the vertex set V is a set of actors $\mathbb{A} = [i_1, i_2, \ldots, i_n]$ and the edge set E is a set of FIFO channels. Each FIFO channel carries a sequence of tokens $\mathbb{X} = [x_1, x_2, \ldots]$, where each x_i is called a token. Fig. 4.1a shows an



(a) Dataflow graph

(b) Heterogeneous multi-processor platform

Figure 4.1: Dataflow graph and architecture platform specified in XDF and SLAM files respectively

example of dataflow graph. An actor is a computational entity with interfaces (input and output ports), internal states and parameters. Actors can interact by exchanging data (called tokens) through channels, unidirectional First In - First Out (FIFO) buffers. During a process, the actor consumes input tokens, produces output tokens and changes its internal state. The actors may be compliant with different models of computation (MoC) and so consume and produce a fixed or variable number of tokens, additionally the execution time can be more or less variable. In practice the graph is specified in an XDF file, which is an input of our mapping algorithm as shown in Fig. 4.2.

4.1.2 Architecture model

The specification of the heterogeneous platform is based on an architecture model that can include different types of processors with or without hardware accelerators. We consider an architecture model, where each processor has its local memory (LM) and communicates with other processors through several buses and shared memories (SM). In this chapter, we consider a heterogeneous platform where processors communicate via one bus and one shared memory. This architecture includes one ARM processor and multiple Microblaze processors, which have different hardware accelerators and frequencies as well. Fig. 4.1b gives an example of a simple architecture platform with one ARM processor (that can run the mapping algorithm), two Microblaze (MB) with one hardware accelerator and all processors communicate through one bus and one shared memory (SM).

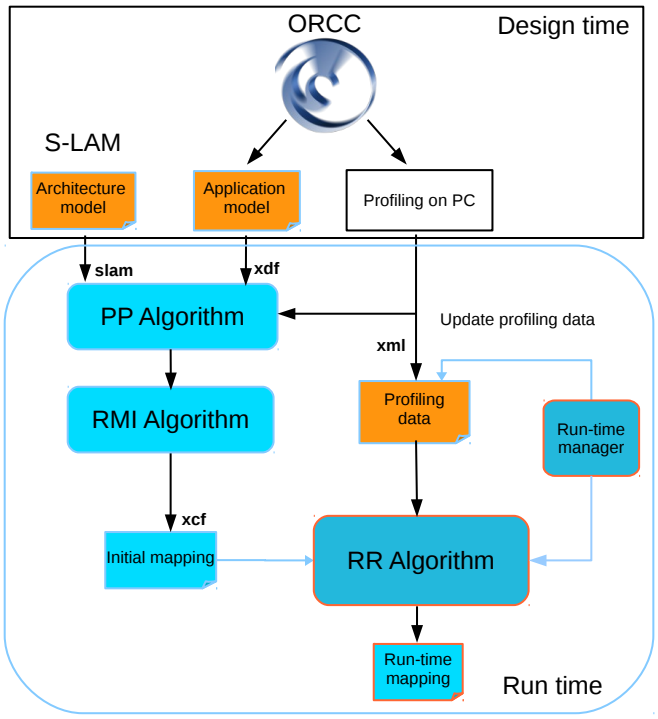


Figure 4.2: Flow overview

4.1.3 Communication model

The latency of a communication medium (bus, NoC) evolves with contention occurrences. But access conflicts cannot be estimated with simulations when fast partitioning is required. That is why we propose to use analytical models that can be updated online if necessary. These models are based on what is observed in the domain of NoC, where the latency is usually measured for different injection rates.

In our approach, the communication model gives the relationship between use-rate and latency since the use-rate can be estimated at runtime according to mapping decisions or estimates. We propose a generic and parametric communication model as in Fig. 4.3. The aim is to have a unique model that can fit with different communication standards, which are supported in the platform. A model is based on two linear functions $y_1 = a_1x + b_1$ and $y_2 = a_2x + b_2$, the intersection between the two curves correspond to the saturation threshold (U_T). These functions can express the communications of a NoC, i.e. the dashed curve in Fig. 4.3, or a bus, i.e. the dash-dot line where $y_1 = y_2$. In practice the parameters of y_1 (a_1, b_1) and y_2 (a_2, b_2) can be adapted online according to real profiling as depicted in Fig. 4.2.

The communication latency is so defined as a function of a use-rate on the

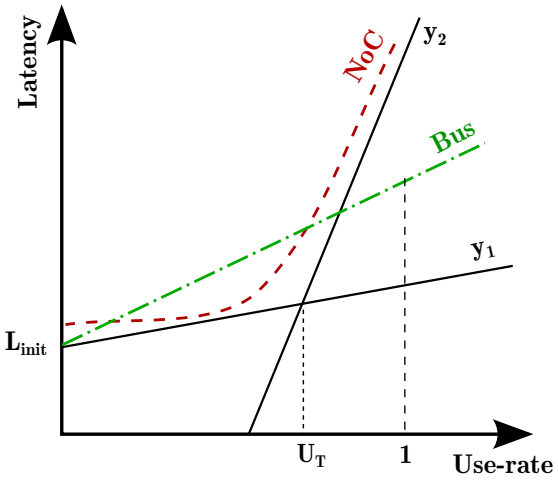


Figure 4.3: An analytical communication model

bus, equ 4.1.

$$f_{cl}(x) = \begin{cases} y_1 & \text{if } x \leq threshold(U_T) \\ y_2 & \text{otherwise} \end{cases} \tag{4.1}$$

4.2 Move based mapping algorithm

The move based mapping algorithm can be divided into three steps: *pre-processing* (PP), *runtime mapping initialization* (RMI) and *runtime remapping* (RR). In the pre-processing step, the goal is to quickly find an initial average period. Therefore, PP ignores communication cost and memory capacity. It computes the initial average period based on a minimum execution time of each actor on the heterogeneous platform from the profiling data. Then, the RMI uses this initial average period as an input to specify a processing budget for all processors. The objective of this process is to maximize the system throughput while considering both computation cost and communication cost at RMI step. In order to further improve the system performance at runtime, the RR step tries to do a remapping and makes a trade-off between the cost of remapping and the mapping performance improvement. The details of three steps are explained in the following subsections.

4.2.1 Parameters and evaluation metrics

In this section, Table 4.1 presents all the parameters and variables used to formalize our move based algorithm. Then, some metrics, which are necessary to

evaluate the mapping quality of dataflow actors, are defined.

Table 4.1: Parameters and variables used for the mapping algorithm

Parameter	**Definition**
DPN application graph (DPNapp)	
$\|\mathbb{A}\|$	Number (Nb) of actors
$\|\mathbb{F}\|$	Nb of FIFO channels
Architecture graph (arch)	
$\|\mathbb{P}\|$	Nb of processors
$MCode_j$	Code memory size of processor j
$MData_j$	Data memory size of processor j
$Freq_j$	Frequency of processor j
Profiling data (profile)	
R_i	Nb of firings of actor i per iteration
W^i	Total computation time of actor i
Cs^i	Instruction code size of actor i
W^e_k	Amount of data on channel k
T_{ij}	Execution time of actor i on processor j
T_c	Communication time per token
$Prod_k, Cons_k$	Nb of token produced & consumed on k

The total computation cost of actor i is the sum of computation times of this actor on all processors on the heterogeneous multiprocessor platform. It means that all processors can execute all actors, otherwise the average computation cost can be used instead. This parameter is computed as the metric for the first sorting in our heuristic algorithm:

$$W^i = \sum_{j \in \mathbb{P}} R_i * T_{ij} \qquad \forall i \in \mathbb{A} \tag{4.2}$$

The amount of data on channel k stands for the number of tokens, which transfer between the source actor and the destination actor of this channel:

$$W^e_k(src, dst) = R_{src} * Prod_k + R_{dst} * Cons_k \;\; \forall k \in \mathbb{F} \tag{4.3}$$

Instruction memory usage of processor j is the total code size of all actors, which are mapped on this processor:

$$memUsage_j = \sum_{i:\mathbb{P}[i]=j} Cs^i \qquad \forall j \in \mathbb{P} \tag{4.4}$$

Computation time of processor j is the sum of the execution time of all actors which are mapped on this processor:

$$compT_j = \sum_{i:\mathbb{P}[i]=j} R_i * T_{ij} \qquad \forall j \in \mathbb{P} \tag{4.5}$$

Communication time of each processor is the total communication time of all actors, which are mapped on this processor. Our analytical communication model is employed to compute $commT_j$:

$$commT_j = \sum_{i:\mathbb{P}[i]=j} 0.5 * f_{cl}(x) * comm(i) * T_c \tag{4.6}$$

where $comm(i)$ is defined as the total number of tokens transferred on the communication media of actor i.

Period of each processor is the sum of total computation time and total communication time:

$$Period_j = compT_j + commT_j \qquad \forall j \in \mathbb{P} \tag{4.7}$$

Throughput is defined as the inverse of the maximum period that takes the longest time to execute one iteration of all actors mapped to its processor.

$$Th = \frac{1}{\max\limits_{j\in\mathbb{P}}(Period_j)} \tag{4.8}$$

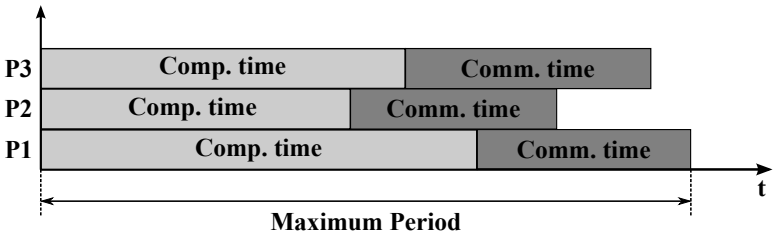


Figure 4.4: An example of maximum period

Fig. 4.4 presents an example of the measured periods during one iteration on 3 different processors in a heterogeneous platform after mapping a dataflow application. Given a partition, regardless of the scheduling, the period with the longest duration imposes on the lower bound of the throughput. In order to maximize the system throughput, we turn this objective to **minimize the maximum period**.

4.2.2 Pre-processing - PP

Since the dynamic behaviors of modern video applications make them difficult to predict at design-time, the pre-processing step, which is outlined in algorithm 2, is based on low-cost profiling analysis of the execution time. Consequently, we assume that profiling mechanisms are available on the target platform. The

algorithm 2 gets data from a default profiling (line 1 in Algo. 2) that gathers the execution time of each actor on different processors. Then, for each actor, the function `SelectMinExe` returns the value of the minimum execution time of the given actor among the processors (line 3 in Algo. 2). In other words, each actor has the best performance in terms of execution time on the platform. This step helps to compute the initial period of each processor without considering the communication time. However, this step may lead to a solution where all actors will be mapped on their best processors. It means that some processors are not used while others have a lot of actors. We expect a load balancing among processors, so we calculate an initial average period (line 5 in Algo. 2) and use this value as a processing budget for all processors.

Algorithm 2 Pre-processing - PP

Input: DPNapp, arch, profile
Output: initial average period

1: getInitExe($\mathbb{A}$,$\mathbb{P}$,profile)
2: **for all** $i \in \mathbb{A}$ **do**
3: initSum+=SelectMinExe(i,$\mathbb{P}$,profile)
4: **end for**
5: initPeriod=initSum/$|\mathbb{P}|$
6: **return** initPeriod

4.2.3 Runtime mapping initialization - RMI

4.2.3.1 Algorithm principle

In this subsection, we inherit GB4M2 from our work in [46]. We propose a modified greedy algorithm combined with a speculative approach to find a good initial mapping. This choice is motivated by the objective of being fast to produce a solution while considering all the constraints of the problem. In contrast to conventional existing mapping strategies, our method differs in following aspects: i) it relies on a factor called alpha (α), which represents a ratio of processor cycle budget considered during the first step (computing-oriented) mapping; ii) it takes into account both computation and communication workloads; iii) it performs both actor and data mappings. Our RMI algorithm, which is outlined in Algorithm 3, can be divided into a computation phase and a communication phase.

Algorithm 3 Runtime mapping initialization - RMI

Input: DPNapp, arch, profile, α
Output: mapping of actors and data

1: {computation phase}
2: $LA1 \leftarrow Actor\_sorting\_Tcomp(\mathbb{A}, \mathbb{P})$
3: **while** $procUse \leq \alpha$ **do**
4: $k \leftarrow$ index_first(LA1)
5: map_actor_Tcomp(k, arch)
6: $procUse \leftarrow \min_{j \in \mathbb{P}}(procUse_j)$
7: $k \leftarrow k + 1$
8: **end while**
9: {communication phase}
10: period_estimate $\leftarrow procUse$
11: **if** $k < |\mathbb{A}|$ **then**
12: LA2 $\leftarrow$ actor_sorting_Tcomm(k,in_out_comm(k))
13: $tcommG \leftarrow \sum_{k \in index(LA2)} (in\_token(i_k) + out\_token(i_k))$
14: $latency \leftarrow Linit$
15: $Tcomm \leftarrow 0.5 * latency * \beta * tcommG$
16: $Cuse \leftarrow Tcomm/period\_estimate$
17: **end if**
18: **for all** $i \in$ LA2 **do**
19: update $latency$
20: **for all** $j \in |\mathbb{P}|$ **do**
21: $tcommGuess_j(i, arch)$
22: $periodGuess_j(i, arch)$
23: **end for**
24: $map\_actor\_Tcomm(min(periodGuess_j), arch)$
25: update $procUse$
26: update $Tcomm$
27: update $Cuse$
28: **end for**

4.2.3.2 Computation phase

We introduce a factor α, which represents a ratio of the processor budget. This factor can vary in the range of $]0, 1]$. In this step, the algorithm performs a sorting of all actors in a descending order with respect to the value of their total computation cost (W^i) Eq. 4.2. A fast sorting algorithm (a bubble sort in the current version) is used. Then, the algorithm picks up the first actor i in the sorted list LA1, i.e. the actor with the largest computation cost. Next, the algorithm selects the best processor, in the list of available processors, according to the minimum T_{ij}. When the current actor is mapped (line 5 in Algo. 3), the algorithm updates the processing use (line 6 in Algo. 3) of the selected processor. At the beginning, all processors are available. As long as the actors are mapped, the processors are removed from the available list if the memory capacity is exceeded or if the processing use ($ProcUse$) is greater that the factor α. Then the algorithm selects the next minimum execution time of this actor on the remaining available processors. This approach is a modified first-fit decreasing bin-packing heuristic, which is a straightforward greedy approximation algorithm. The algorithm applies the same procedure for the next candidate in the first sorted list until the processing use of all processors is greater than α, which is a percentage of the processor budget.

4.2.3.3 Communication phase

The communication phase starts with the remaining actors, which are not mapped. Then, it applies the second sorting for all unmapped actors. This sorting is based on total incoming and outgoing data for each actor. The idea is to take the communication cost into account in this phase. Hence, the algorithm takes the first actor in the second sorting list and considers all the connections with other actors in the network. If this actor and its adjacent actor are mapped on the same processor, the communication time is zero. In order to deal with unknown information when we estimate the use-rate of the bus (% usage of available bandwidth), we introduce a factor β that represents the ratio of remaining data transfers associated to unmapped actors that will use the communication media (e.g. bus). Then the latency is estimated with the communication model (Fig. 4.3) and the algorithm makes a decision of mapping. Then, the algorithm applies the same procedure for the next unmapped actor in the second sorting list.

Fig. 4.5 gives an example of a graph application, which includes 7 actors $\{i_1, i_2, \ldots, i_7\}$. The target platform has three processors $\{P_1, P_2, P_3\}$. In the computation phase, actor i_1, i_3 and i_5 are mapped on P_2, P_3 and P_1 respectively. Then, we have the remained actors with the second sorting list LA2=$\{i_2, i_6, i_4, i_7\}$. In order to map actor i_2, the algorithm takes into account all communication

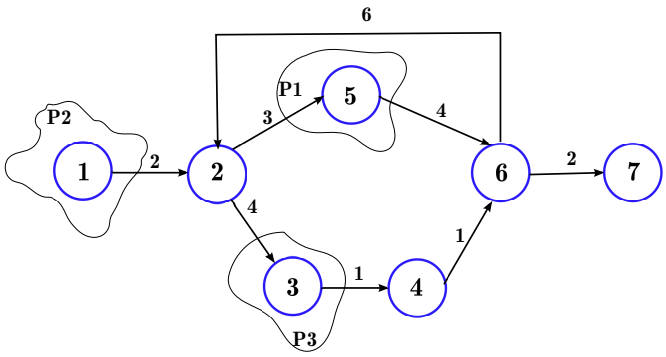


Figure 4.5: Example of a graph application

(line 21 in Algo. 3) between i_2 and its adjacent actors in the graph application. At this step, i_6 is not mapped so $tcommGuess_j(index(i_2), 3) = latency * \{(4 + 2 + \beta * 6), (4 + 3 + \beta * 6), (2 + 3 + \beta * 6)\}$. From this information, the algorithm computes $periodGuess_j$ (line 22 in Algo. 3). Then, the algorithm decides to map actor i_2 on a processor, where i_2 has the minimum value of periodGuess. Next, the algorithm updates the $procUse_j$, $Tcomm$ and $Cuse$ (line 25 to 27 in Algo. 3). All actors in $\{i_6, i_4, i_7\}$ follow the same procedure as actor i_2.

4.2.4 Runtime remapping - RR

After the first mapping is found, the application is launched and the system monitors the workload. The RR is activated to determine whether or not a performance arises and then make a decision of remapping. All procedures of RR step are presented in Algorithm 4. We modify the Fiduccia and Mattheyses algorithm (FM) [114] in terms of actor moves and gain concept to apply in our context. The cost of remapping tries to balance between migration cost and performance improvement.

$$Cost_{remap} = max(gainT(i)) \tag{4.9}$$

Where the gainT is the gain of moving actor i from current processor to another one.

The main steps of our RR algorithm are described below.

4.2.4.1 Finding the possible moves

The RR starts with an initial mapping obtained from RMI solution, while FM uses an initial random mapping. The algorithm inherits the idea of "cell moves" from FM and changes to a concept of "actor moves". This "actor moves" allows a significant time complexity improvement since we do not need all-pair swap gain

computation like in Kernighan and Lin algorithm [114]. Instead of considering all actor moves as in FM [114], we limit the number of actor moves where only actors on the processor with maximum period are moved to other processors. Indeed, the study of migration cost in the context of real-time system is very expensive [115], [116]. Moreover, we need a very fast algorithm, in order not to disrupt operation for long. Therefore, we just need to find the processor which has the maximum period.

Firstly, we get the information of initial actor mapping (line 1 in Algo. 4) based on RMI solution. According to the changing of workload, we update the period of each processor. Then, we find the maximum initial period (line 3 in Algo. 4). At the end of this procedure, we have a list of actor move candidates ($\mathbb{C}$), which belong to the processor with maximum period.

Algorithm 4 Runtime remapping - RR

Input: graph, arch, init_map, profile
Output: a better mapping

1: $get\_mapping\_info(graph, arch, init\_map, profilie)$
2: $ActorMove\_estimation(graph, latency, move)$
3: $\mathbb{C} \leftarrow compute\_maxPerI()$
4: **for** $i \in \mathbb{C}$ **do**
5: $\quad Move\_gain(graph, latency, initL, arch, profile)$
6: $\quad PerG(i)$ {equation 4.11}
7: $\quad Cost_{mig}(i)$ {$lose\_cache\_miss$}
8: **end for**
9: $select \max_{i \in \mathbb{C}}(gainT(i))$

4.2.4.2 Trade-off between migration cost and performance improvement

We perform the following three steps before making the decision of actor move: (i) unlock all actors in the list of actor move candidates, (ii) compute the gain of performance and a migration cost, and (iii) find the maximum gain. We repeat three steps at every move until all actors in the list are locked. Since FM applies for bipartitioning problem, we modified the gain concept of FM, which can handle multi-way partitioning problem as in [117].

In this chapter, the impact of actor migration is considered when doing the remapping of dynamic dataflow application. Some researches in [45], [116] and [107] show the important role of the migration cost of multimedia applications on embedded multiprocessors. Different mapping may incur different migration

cost [45]. In order to simplify, we make the assumption that all the binary code of actors are contained in a shared memory. The cost of actor migration is proportional to the size of the binary code compiled for the processor travelling on a communication media in a platform. Therefore, we compute the migration cost as a function of cache miss (Eq. 4.10) that happens when moving actor from one processor to another one. More sophisticated models can be proposed but we believe that in the context of runtime mapping of dynamic actors/applications a simple model is enough (an accurate model would be very complicated to model all the dynamic aspects).

$$Cost_{mig}(i) = f_{cl}(x) * Cs^{i} \tag{4.10}$$

Our MBA algorithm tries to minimize the maximum period to obtain a maximum performance in terms of throughput. In other words, we can obtain a better mapping when a maximum new period is less than the maximum initial period. We define maxPerI as the maximum initial period. The maxPerN is the maximum new period, which is obtained after moving actor to other processors. Then, we have the performance improvement PerG in terms of period as in Eq. 4.11.

$$PerG(i) = maxPerI - maxPerN(i) \tag{4.11}$$

Finally, we get the gainT, which is a trade-off between migration cost (Eq. 4.10) and performance improvement (Eq. 4.11) as in Eq. 4.12. We select the actor with the maximum gainT to move. The gain of moving actor i from current processor to another one is computed:

$$gainT(i) = PerG(i) - Cost_{mig}(i) \tag{4.12}$$

Table 4.2: Example of actor move candidates - At time t_1, t_2 and t_3

time	$actor_m$	$Cost_{migP1}$	$maxPerN_{P1}$	$\mathbf{gainT}_{P1}$	$Cost_{migP2}$	$maxPerN_{P2}$	$\mathbf{gainT}_{P2}$	$Cost_{migP3}$	$maxPerN_{P3}$	$\mathbf{gainT}_{P3}$	$Cost_{migP4}$	$maxPerN_{P4}$	$\mathbf{gainT}_{P4}$
t_1	i_5	141.01	159251.86	-63779.24	x	x	x	140.93	120064.4	-24591.7	140.99	113228.34	-17755.7
	i_6	96.8	109489.48	-13972.66	x	x	x	96.82	91865.92	**3650.89**	96.86	91865.95	3650.81
t_2	i_5	141.01	162131.86	-66649.84	x	x	x	140.92	122104.77	-26622.67	140.99	114677.75	-19195.71
	i_6	96.79	108280.07	-12753.84	x	x	x	96.81	93065.92	**2460.29**	96.86	93065.95	2460.21
t_3	i_4	x	x	x	491.15	175831.22	-82905.95	490.71	125350.86	-32425.16	490.83	117917.62	-24992.04
	i_1	x	x	x	87.17	115639.17	-22309.93	87.17	89700.34	**3628.9**	87.21	89700.39	3628.82

As an example, let's consider the application graph, composed of the network of actors $\mathbb{A} = [i_1, i_2, \ldots, i_7]$, as shown in Fig. 4.5 on a heterogeneous platform including 4 different processors, $\mathbb{P} = [P1, P2, P3, P4]$. Table 4.2 shows the results of the estimation of $gainT$ at different time steps. We assume an initial mapping as $P1\{i_4, i_1\}, P2\{i_5, i_6\}, P3\{i_3\}, P4\{i_7, i_2\}$. From this result, the $maxPerI$

is 95613.62 time units. The $maxPerN$, $Cost_{mig}$ and $gainT$ in Table 4.2 are expressed in time units. According to the profiling workloads, at time t_1 and t_2, we have the maximum period on processor $P2$. Hence, the RR estimates $gainT$ when moving i_5 and i_6 from $P2$ to the other processors. At time t_1, we have the maximum gain when moving actor i_6 from processor $P2$ to processor $P3$. At time t_2, we get the same result of remapping with different $gainT$ value. As a result, we obtain a new mapping as $P1\{i_4, i_1\}, P2\{i_5\}, P3\{i_3, i_6\}, P4\{i_7, i_2\}$. At time t_3, the maximum period is now on processor $P1$, which has actor i_4 and i_1 and if actor i_1 moves from processor $P1$ to processor $P3$, we can expect the maximum $gainT$. The new mapping is $P1\{i_4\}, P2\{i_5, i_6\}, P3\{i_3, i_1\}, P4\{i_7, i_2\}$. This table shows how the algorithm selects an actor from a list of candidates.

4.2.5 Runtime scenario based simulation - RSS

RSS is used to test and compare the performance of runtime actor mapping algorithms with various scenarios. We provide a mechanism for generating different scenarios that can be used as benchmarks. The goal is to simulate the behavior of dynamic actors in an application graph. We have three graph applications, where each actor in the application graph has different standard deviations. To simulate different input videos, we create 3 scenarios for each application graph. They are different in terms of percentage of dynamic actors. Since we consider the application model based on different MoCs. This also means that each application model may contain both static actors (SDF, CSDF ...) and dynamic actors (KPN, DPN). We define the percentage of dynamic actors as the ratio of number of dynamic actors over number of actors in the application graph. Each scenario has 100 time steps and contents information of execution time of dynamic actors, which varies from time to time for different processor types on the heterogeneous platform.

Table 4.3: Various scenarios in each application graph

Graph	$\|\mathbb{A}\|$	$\|\mathbb{F}\|$	NbScenario	Nbtime-steps/scenario
1	20	38	3	100
2	40	110	3	100
3	60	158	3	100

Table 4.3 shows the various scenarios in each application graph. We generate different scenarios by using Gaussian distribution defined by Eq. 4.13. A normal distribution is

$$f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{4.13}$$

- The parameter μ is the mean or expectation of the distribution
- The parameter σ is its standard deviation

We also have the minimum and maximum parameter for the generation function. In order to specify the number of dynamic actors for each application graph, we introduce the δ parameter, which is defined as the ratio of dynamic actors in each application graph. Then we apply the generation function for these dynamic actors to create different scenarios. The following steps describe how to create different scenarios:

1. Use SDF3 tool [37] to generate XDF file that contains the structure of the graph. This file also provides the profiling information of actors and channels in the graph
2. Create a data file to contain only workload information both in computation and communication
3. Choose a percentage of dynamic actors of the graph and specify the dynamic actors
4. Use our mechanism with the data file (step 2) to generate execution time of the dynamic actors at different time steps
5. Change the percentage of dynamic actors (step 3) and follow step 4 to have another scenario

4.3 Experimental results

In this section, we present the results of our experiments by exploring different aspects of our MBA algorithm of runtime mapping of dataflow actors on the heterogeneous multiprocessor platforms. We apply our MBA algorithm for both generated dynamic dataflow and real dynamic dataflow application. In order to evaluate our heuristic approach more thoroughly, we make the comparison with [46].

4.3.1 Setup environment

For the generated dynamic dataflow applications, we use SDF3 tool [37] combined with our RSS to generate dynamic workload scenarios at different time steps. For the real dataflow applications, we test our methodology with MPEG4-SP Part 2 Simple Profile (MPEG4-SP). Implementation of MPEG4-SP is carried out with ORCC [31] tool as in [71]. The System-Level Architecture Model (S-LAM) [38]

is employed for the specification of different sets of heterogeneous multiprocessor platforms. The number of processors varies from 4 to 8, combined with several hardware accelerators. All experiments were conducted in Linux environment on HP ProBook with core i5-2520M CPU@2.5GHz and 8GB memory.

4.3.2 The need of runtime remapping

In this subsection, we illustrate the variation in terms of workload for different MPEG4 video sequences. We take into account both computation workload and communication workload, which are computed as in Eq. 4.14 and Eq. 4.15, respectively. We formulate Eq. 4.14 and Eq. 4.15 based on these definitions in [14]. In this chapter, we use the term "workload" to represent both computation and communication workload.

The computation workload of an actor is defined as the ratio of its computation time in a given time interval.

$$Comp_w(i) = \frac{W^i}{\sum_{n \in \mathbb{A}} W^n} \qquad \forall i \in \mathbb{A} \tag{4.14}$$

The communication workload of a channel is measured as a ratio of the number of tokens over the total number of tokens in a given time interval. It has to be noticed that computing the workload on the produced or consumed tokens only is enough. Indeed, all produced tokens are also consumed. Computing the workload with the consumed tokens also will just double the number of tokens and the ratio will still be the same.

$$Comm_w(k) = \frac{Prod_k}{\sum_{f \in \mathbb{F}} Prod_f} \qquad \forall k \in \mathbb{F} \tag{4.15}$$

This work is a direct use of the work presented in [35] with a slight modification for the sliding time window profiling used in section 4.3.5.

We use the GB4M2 algorithm [46] to find a mapping solution for each MPEG4 video sequence. According to profiling information, we obtain different mapping results for different input MPEG4 video sequences on the same target platform. This difference can be explained by the fact that we have some dynamic actors in MPEG4-SP decoder. Moreover, different input video sequences have different variation in terms of workload. In order to demonstrate the dynamic behaviors, we keep the same mapping solution at different time step. In this experiment, we observe that the change concerns not only the computation time but also the communication time for each video sequence and every 10 frames (Fig. 4.6). The communication time has fewer variations because we apply the same mapping

solution at every 10 frames. We observe 3 input video sequences from [118]: foreman, stefan and coastguard during the length of each video. There are more complex movements in the foreman video than the coastguard video, so that we can see more dynamic behaviors of the foreman video as shown in Fig. 4.6a, 4.6c. The result shows that the workload varies over the time. This means that we should have an adaptive algorithm for mapping at runtime. In conclusion, these differences clearly demonstrate the need of runtime remapping for dynamic dataflow actors.

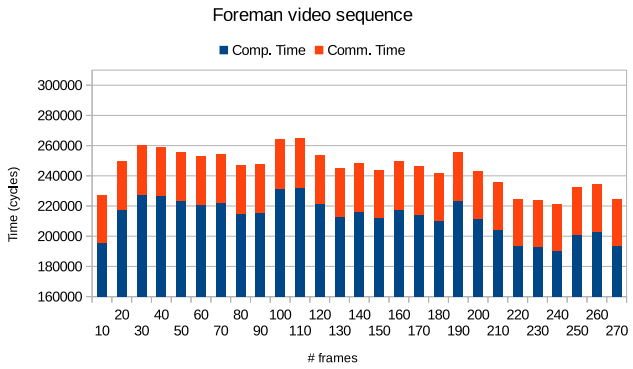
4.3.3 Generated application graphs

We use three generated application graphs which are created by SDF3 tool [37] and our RSS, as explained in subsection 4.2.5. The properties of application graphs were described in Table 4.3. There are 3 dynamic application graphs with the number of actors that varies from 20 to 60. Considering the reference in [36], the MPEG4-SP application contains up to 45% of dynamic actors, so we create 3 scenarios with percentage of dynamic actors varied from 20% to 60% for each application graph. Token transfers on each channel, in/out degrees of actors are randomly created within minimum and maximum bounds and with different average and variance.

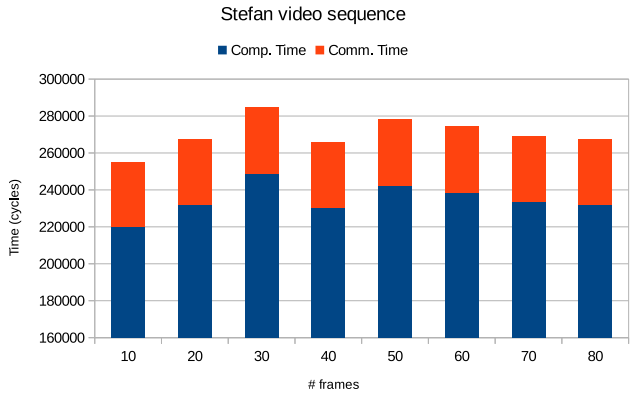
We then apply our mapping flow for each scenario on a heterogeneous platform that includes 4 processors with different hardware accelerators. We assume that the actors can be executed by all different processor types on the heterogeneous platform.

As it can be seen, MBA achieves better throughput at different time steps of all scenarios in Fig. 4.7, Fig. 4.8 and Fig. 4.9. During the experiments, we observe that the variation in workload of the dynamic dataflow applications makes a significant impact on the performance. The more the workload varies, the more dynamic behaviors we have. This is true for all scenarios. This observation also confirms that we can not keep the same mapping for all input video sequences even with the same video decoder.
For instance, in Fig. 4.7c, we have a list of actor move candidates $\{i_5, i_{10}, i_6, i_{18}, i_{17}\}$ on processor 3 after applying RMI algorithm. Then we use RR algorithm to obtain 2.47% speedup when moving actor i_{10} from processor 3 to processor 4 at time step 16. From time step 17 to 30, actor i_{17} is selected to move from processor 3 to processor 1 to achieve the performance improvement. During this time, the maximum speedup is 18.32% at time step 26.

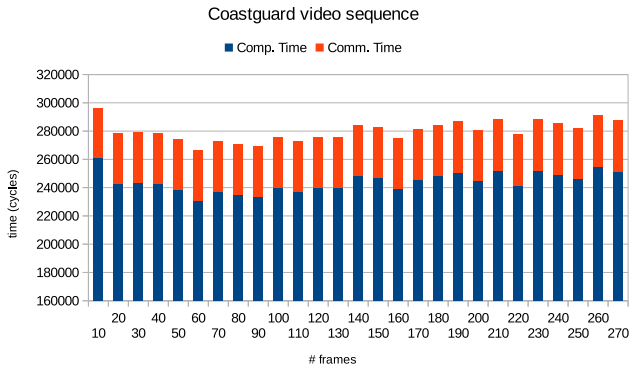
Fig. 4.10 shows the range of speedup, from min value to max value, when applying MBA algorithm for all scenarios of different application graphs. We observe the range of speedup of 3 graph applications as in Table 4.3. As it can be seen, in graph application with 20 actors, when we increase the percentage



(a) Foreman

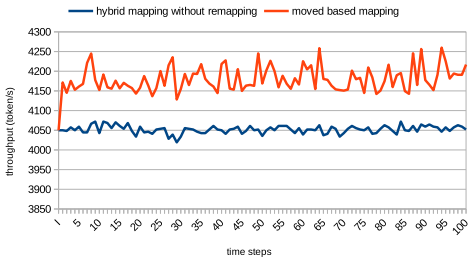


(b) Stefan

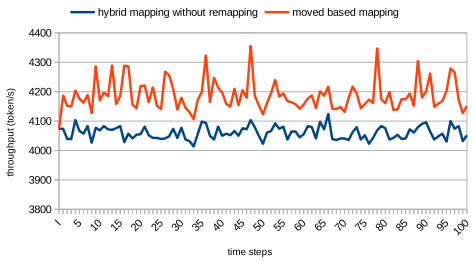


(c) Coastguard

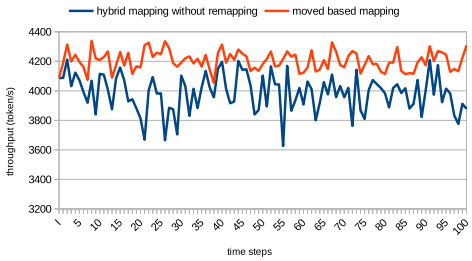
Figure 4.6: Variation in both computation time and communication time of different input video sequences



(a) Scenario 1 - 20% of dynamic actors

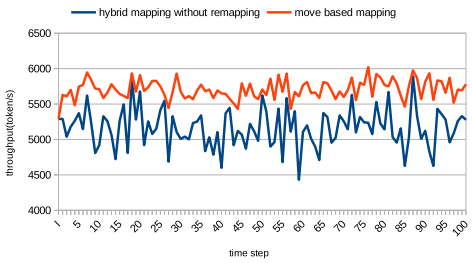


(b) Scenario 2 - 40% of dynamic actors

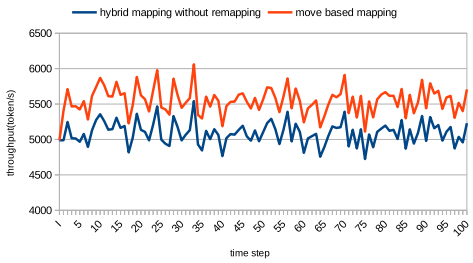


(c) Scenario 3 - 60% of dynamic actors

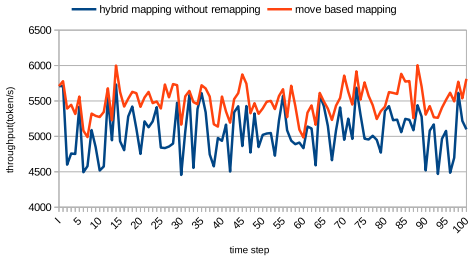
Figure 4.7: Application graph with 20 actors and 38 channels



(a) Scenario 1 - 20% of dynamic actors

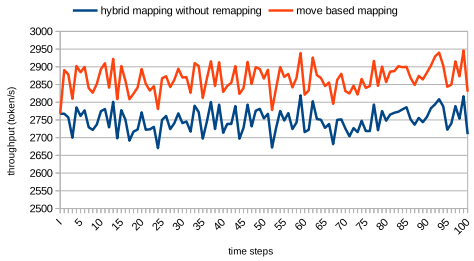


(b) Scenario 2 - 40% of dynamic actors

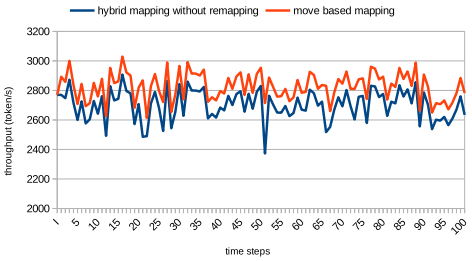


(c) Scenario 3 - 60% of dynamic actors

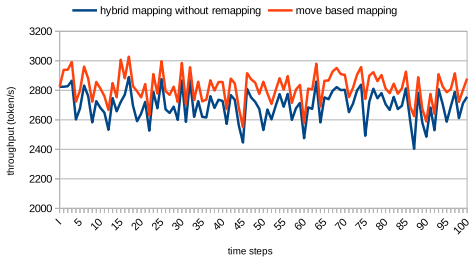
Figure 4.8: Application graph with 40 actors and 110 channels



(a) Scenario 1 - 20% of dynamic actors



(b) Scenario 2 - 40% of dynamic actors



(c) Scenario 3 - 60% of dynamic actors

Figure 4.9: Application graph with 60 actors and 158 channels

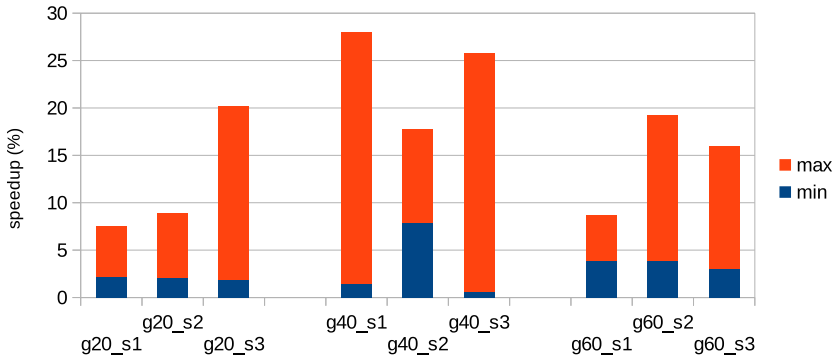


Figure 4.10: Speedup in different scenarios

of dynamic actors in the application, we get more dynamic range of speedup in terms of performance. However, this is not true for other graph applications with 40 actors and 60 actors. This typically happens when less dynamic actors are mapped on the processor with maximum period.

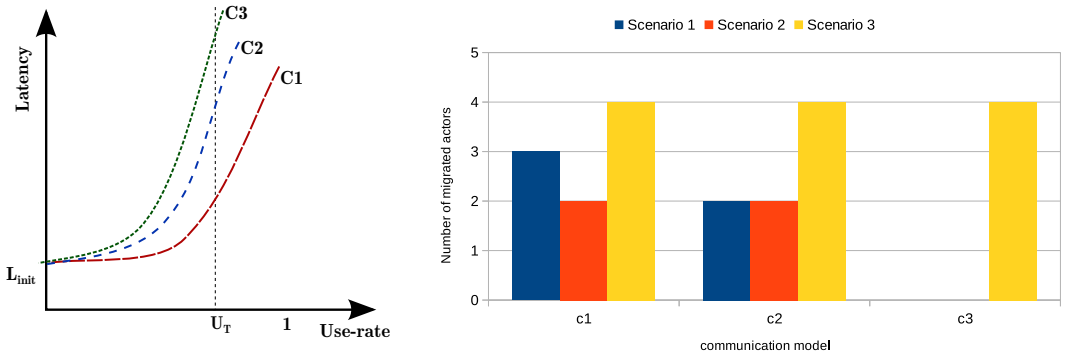
In these experiments, we observe that the frequency of moving does not happen fast. In other words, we can have the same mapping at different time step (time interval) with different performance improvement. This is an interesting result since it leads to an idea to have a parameter in MBA algorithm to make a decision of remapping, move an actor from one processor to another one, at a good time. This parameter is defined as percentage of performance improvement and makes our algorithm more adaptive. For example, after we select the best actor move candidate in the list, if we estimate the performance improvement greater than 5% then the decision of moving is made. Otherwise, we keep the current mapping. In some cases, if we do a remapping actor, we gain a very small speedup. The results also illustrate our MBA algorithm can deal with a wide variety of dataflow application model.

4.3.4 Impact of migration cost at runtime

In order to demonstrate the impact of migration cost, we use three different communication models in Fig. 4.11a for the same scenario. Fig. 4.11b presents the results of number of actors that need to be migrated to improve the performance when applying our MBA algorithm for graph application with 20 actors and 38 channels. This application has three scenarios. As mentioned before, scenario 1 has 20% of dynamic actors, scenario 2 has 40% of dynamic actors and scenario 3 has 60% of dynamic actors. Different colors indicate the number of migrated actors from different scenarios. We run 100 time steps for each scenario. We explore the behavior of all three runtime scenarios for changing the communication models (c1, c2 and c3). Fig. 4.11b shows that the number of migrated actors decreases when the available bandwidth decreases. With communication c3, we cannot enhance the performance of scenario 1 as well as scenario 2 by remapping actor because we have a limited available bandwidth. For scenario 3, although we have the same number of migrated actors for different communication models, we can not have the same actor move because of the different migration costs. We also see that different workload scenarios may have different number of migrated actors on the same platform at the same time interval.

4.3.5 Real application graphs

We use MPEG4-SP video decoder, which is developed by the RVC group. Table 4.4 describes the properties of MPEG4-SP, as well as the number of actors and



(a) Different communication models

(b) Migration cost

Figure 4.11: Impact of migration cost at runtime

FIFO channels. The C code of these decoders are generated with Orcc tool [31]. Some options in the tool can allow for generating additional code for intrusive profiling. In these experiments, we obtain the runtime profiling workload by using two ways: one bases on the average time (AT10), i.e. updating profiling workload after 10 frames, another one bases on sliding time window (STW10), i.e. updating profiling workload every 10 frames. All actors are classified [36, 27] as different MoCs such as SDF, CSDF, KPN and DPN. All experiments made from different video sequences. In particular, *Foreman*, *Stefan* and *Coastguard* videos are used as input videos of MPEG4-SP decoder. The resolution of all MPEG4-SP videos is 352x288, which corresponds to CIF format.

The first profiling is performed on a desktop computer. For these experiments, we specify the size of the FIFO channels used for communication between 512 and 8192 bytes. We introduce some usual hardware accelerators as in Table 4.5. These hardware accelerators can provide significant speed-up when associated with Microblaze (MB) softcore on Xilinx FPGAs. We consider in this example a heterogeneous platform based on 1 ARM and 7 MB processors. By using S-LAM modelling, we specify various hardware accelerators in the heterogeneous platform detailed in Table 4.5.

Table 4.4: Properties of MPEG4-SP video decoder

Decoder	Profile	YUV	#Actors	#FIFOs
MPEG4 Part 2	SP	yes	41	104

Fig. 4.12, 4.13 and Fig. 4.14 present the results in terms of throughput (frames per second). We have better throughput in comparison with the hybrid approach without runtime remapping in [46]. We enhance the throughput by

Table 4.5: Accelerators used in platform 7.1

Platform	MB1	MB2	MB3	MB4	MB5	MB6	MB7
7.1	Merger	IDCT	Parser	Inter	IQ+IAP	Add	IDCT

taking into account the dynamicity to do the remapping of an actor at runtime. In addition, we realize that the way using to obtain runtime profiling workload also affects the system performance at runtime. In case we update runtime profiling workload every 10 frames (STW10), we observe more variable in terms of runtime system throughput than the input runtime profiling workload with the average time manner. At the first 10 frames, we should have the same runtime profiling workload for each video sequence but the results show a slightly different throughput as in Fig. 4.12, 4.13 and 4.14. This can be explained that the runtime profiling mechanism with average time or STW10 manner use Eq. 4.14 and Eq. 4.15 to compute the workload. As a definition of workload in [14], the workload of a processor is the sum of the workloads of all its actors plus a small overhead introduced by their scheduling. As a result, the scheduling makes a slightly different throughput at the first 10 frames of each video sequence.

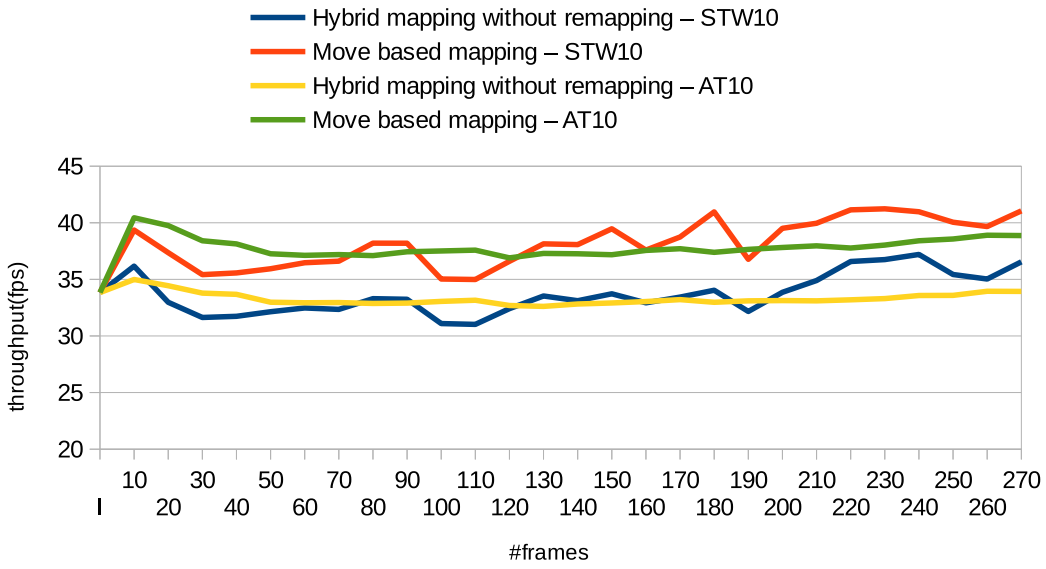


Figure 4.12: Foreman - Platform 7.1

Our runtime solution improves the performances by adapting the mapping at runtime according to observations. However the observation window has an impact on the dynamicity and the performance results. The best choice for the runtime profiling method actually depends on the video sequence. For instance,

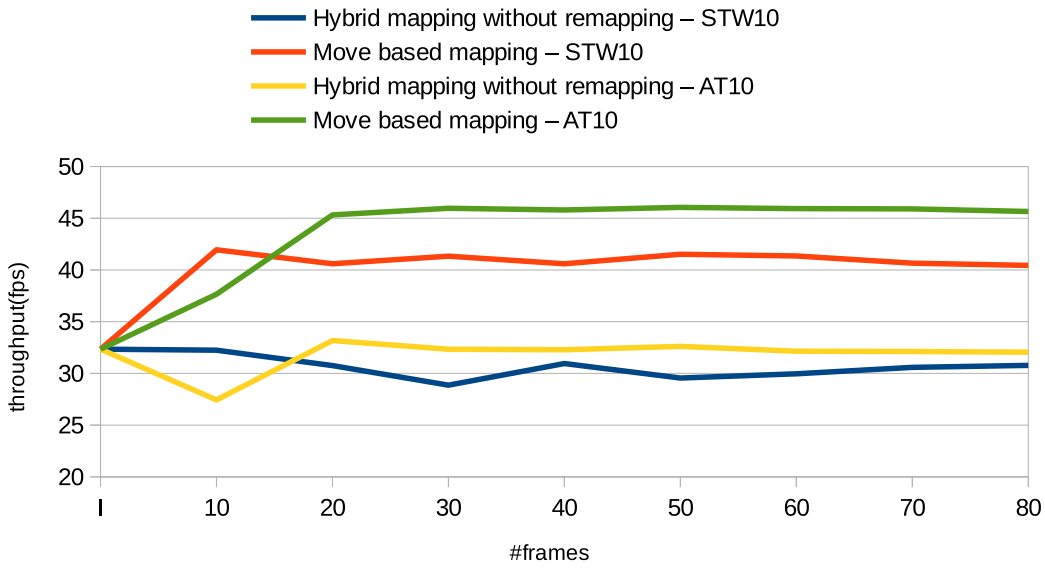


Figure 4.13: Stefan - Platform 7.1

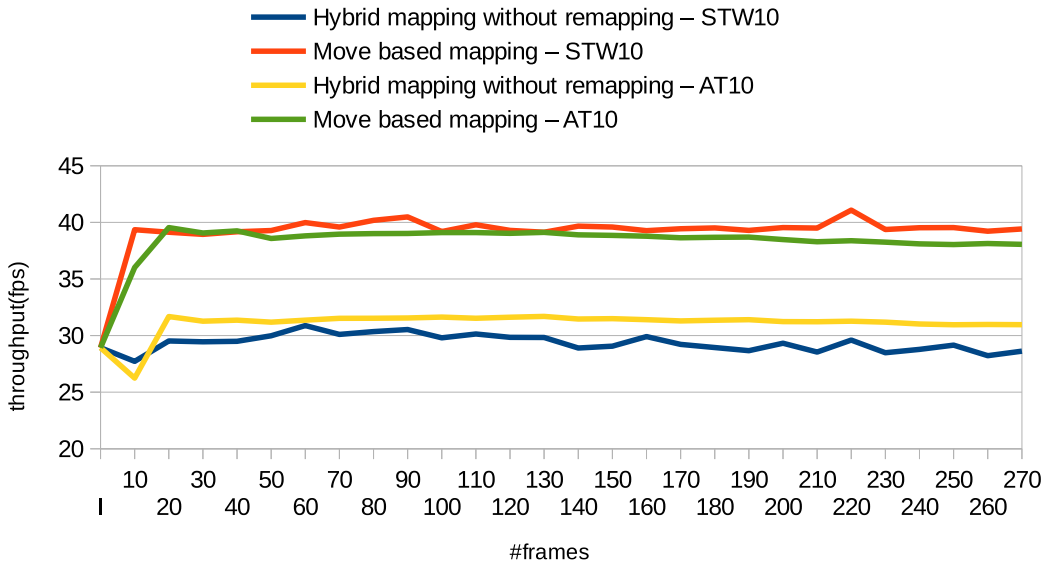


Figure 4.14: Coastguard - Platform 7.1

the result in Fig.4.13 displays the best improvement with the input runtime profiling workload using a full average computation from frames 20 to frame 80. This behavior also happens with foreman video sequence from frame 10 to frame 70, but the STW10 method offers better results from frame 130 to frame 270. In Fig. 4.14, we also see a slight improvement due to STW10 method compared with the one based on AT10. To investigate the dynamicity of workload, we compute a standard deviation of a period, the sum of total computation time and total communication time, at every 10 frames during their whole video lengths. We observe that different input video sequences have different standard deviations. Foreman video has the value of standard deviation of 12795 while it is 9020 and 7171 for stefan and coastguard videos respectively. As can be seen in Fig. 4.6, foreman video sequence is more dynamic than the others. Our conclusion is that a sliding window is a promising solution for runtime adaptation, the choice of the window length can be fixed to default value (e.g. 20) but it may be refined according to the variation rate of the video sequence. This second-order adaptation is a part of our future work.

Table 4.6: Solving time of Runtime remapping algorithm on Zynq platform

Input video	Platform	Runtime remapping
Foreman	7.1	43.55 ms
Stefan	7.1	54.07 ms
Coastguard	7.1	54.01 ms

As indicated in Table 4.6, we measure the solving time of runtime remapping algorithm executed on ARM processor with different input video sequences. RR algorithm took 43.55 ms on average for finding the better mapping while considering the tradeoff between performance improvement and migration cost. The results show that our RR algorithm is fast enough to do a remapping at runtime.

4.4 Conclusion

This chapter is based on two observations. First dataflow applications will use an increasing number of dynamic actors with variable execution and communication times. Secondly the energy efficiency and performances of embedded system can be significantly improved by means of heterogeneous architectures. As a response to the question of dynamicity and the need for performances, we introduce a novel algorithm for runtime actor mapping on heterogeneous multiprocessor architectures.

The originality of the proposed solution is a move-based algorithm adapted to the context of data flow applications. Our approach is fast and designed to handle runtime variations. It deliberately relies on simple analytical models for communication latency and migration costs. These models can be simply adapted to different architecture templates and tuned at runtime according to real observations. We investigate different aspects of our MBA algorithm to improve system performance by offering the remapping solution at runtime. Our approach has been firstly validated with large sets of synthetic graphs and execution scenarios. Secondly it has been validated with real video sequences and architecture models based on Xilinx Zynq devices. Our results demonstrate that significant improvements of throughput can be obtained with a highly flexible MBA algorithm. The proposed solution fits with the Reconfigurable Video Coding paradigm, and is compliant with the ORCC framework. According to the evolution of video standard, it is designed to start with profiling data performed offline, that can be delivered in the header of a video file. Then it adapts the mapping choices according to runtime observations based on a profiling window that can be tailored to application dynamicity.

5 Conclusion

5.1 Conclusion

There are widespread standards of multimedia applications with increasing complexity and dynamicity, such as video codecs, that impact the evolution in design methodologies of multimedia embedded systems. Actually, today embedded systems are becoming increasingly more complex to support a large number of applications or functions in a single device, making traditional analysis techniques unsuitable. More and more processors are integrated into a single chip namely MPSoCs. The trend moves toward heterogeneous MPSoCs to achieve higher performance while saving energy. In the application domain, dataflow MoC shows a powerful perspective on parallel computations at high level descriptions. As a result, the designers have to provide an efficient mapping methodology for such modern multimedia application on heterogeneous multiprocessor platforms with respect to time to market and cost demand requirements. Additionally, a reusable and flexible mapping method is also required in order to deal with the evolution of current and future multimedia embedded systems.

To provide re-usability and flexibility in such embedded systems, we rely on the Y-chart combined with platform based design approach, which facilitates rapid system modifications concerning the application, architecture and mapping. Since MPEG RVC allows reconfiguration and re-usability and also provides platform-independent dataflow models, we consider the real world RVC-CAL applications, which are based upon the general DPN model. The fact is that these dataflow applications are complex and dynamic. Fortunately, they are far from being entirely dynamic, which means that parts of dataflow application can be modeled as static. For example, MPEG4 SP has up to 45% of dynamic actors. Because of the variation workload in both computation and communication, the complete analysis at design time is not feasible. Existing techniques of runtime mapping are still nascent development and lack of complete environment to bridge the gap between hardware efficiency and software flexibility. Thus an efficient runtime mechanism is required to handle this dynamism.

In Chapter 3, a novel runtime mapping algorithm for heterogeneous MPSoCs is presented. This approach relies on hybrid method, which combines DSE at design time and the system status at runtime to select the best mapping of newly arriving actors. This algorithm addresses computation and communication jointly with a simple analytical communication model. We use an abstract model, which allows to easy model MPSoC interconnects. This communication model can capture either Bus-based or NoC-based architectures. By introducing processing budget for each processor, we also avoid memory overhead, which may occur in existing hybrid approaches as they need to store analysis results found at DSE. Our solution is extremely fast and can be used at runtime for the mapping of actors of dynamic dataflow applications for which, by definition, no optimal solution can be found offline. Reconfigurable video coder is a typical domain where such a solution is required. Our algorithm outperforms the fast METIS partitioning algorithm. We demonstrate on a large set of representative synthetic graphs as well as on MPEG4-SP and HEVC real life examples that our algorithm is at least one order of magnitude faster (hundred of μs) on an ARM embedded processor, while producing better performance when the alpha parameter is larger than 0.4. Our approach is between two extreme partitioning techniques: solutions that consider bandwidth as available but do not take it into account and methods like METIS that minimize communication without taking advantage of the available bandwidth to improve the throughput.

Further, we present a move-based algorithm for runtime remapping dataflow actors in Chapter 4. This solution is based on a pragmatic approach with two observations. Firstly dataflow applications will use an increasing number of dynamic actors with variable execution and communication times. Secondly the energy efficiency and performances of embedded system can be significantly improved by means of heterogeneous architectures. As a response to the question of dynamicity and the need for performances, we introduce a complete algorithm for runtime actor mapping on heterogeneous multiprocessor architectures.

The originality of the proposed solution is a move-based algorithm adapted to the context of data flow applications. Our approach is fast and designed to handle runtime variations. It deliberately relies on simple analytical models for communication latency and migration costs. These models can be simply adapted to different architecture templates and tuned at runtime according to real observations. We investigate different aspects of our move-based algorithm to improve system performance by offering the remapping solution at runtime. Our approach has been firstly validated with large sets of synthetic graphs and execution scenarios. Secondly it has been validated with real video sequences and architecture models based on Xilinx Zynq devices. Our results demonstrate that significant improvements of throughput can be obtained with a highly flexible move-based algorithm. The proposed solution fits with the Reconfigurable Video

Coding paradigm, and is compliant with the Orcc framework. According to the evolution of video standard, this solution is designed to start with profiling data performed offline, that can be delivered in the header of a video file. Online, it is possible to execute a runtime mapping before playing a movie if the header contains the require information such as network of actors and profiling data. Then it adapts the mapping choices according to runtime observations based on a profiling window that can be tailored to application dynamicity.

5.2 Perspectives

This thesis presents the solutions for different aspects in mapping methodologies of dataflow applications on heterogeneous platforms. However, this work can serve as basis for future researches. Some promising issues in the methodologies remain to improvement for future implementation on real heterogeneous platforms.

5.2.1 Short term

- **Applications**: The trend moves toward more expressive model required by today's and upcoming applications while the designers want to know how the applications would perform with the given resources. Thus, we need to have benchmarks, which would help to further understand dynamic behaviors. For instance, if we have the information of the standard deviation of each actor in an application at every n frames, we can reduce the solving time of the remapping phase because we can reduce the number of actor move candidates. This understanding could enrich the heuristics developed in this work.

- **Profiling mechanism**: Currently, we assume that the profiling mechanism is available on the platform. The mapping algorithm makes decisions based on profiling information at runtime. This imposes a challenge to have an efficient profiling tool embedded in the architecture to monitor accurate information without degrading the overall system performance.

- **Reliability**: In order to reduce occurrence of faults in the systems, it requires to have preventive measures, which are available on a platform. The current version of our algorithm does not take this information into account. With this kind of information, the mapping algorithm can avoid faulty processors on the platform. For example, when a system detects a faulty processor, the mapping solution will do a remapping of application on the remaining processors.

5.2.2 Long term

- **Adaptive runtime manager**: In this study, we propose some parameters, which make our solution more flexible and adaptive at runtime. The heuristics in this study aim at performance improvement and efficient implementations. However, we do not have a real runtime manager, which would allow for measuring and gathering the current system status. This would help to tune the analytical model proposed in this study to the application. Additionally, future work should focus on making the algorithms energy-aware in order to exploit for energy efficiency. Nonetheless, this will need energy measurement techniques at the electronic system level.

- **Memory mapping**: Mapping of dataflow application has to assign not only actors to processors but also FIFO channels to memory components. In our approach, the shared memory contains the communication channels that connect two actors mapped onto two different processors. The local memory contains the communication channels that connect two actors mapped on the same processor. However, the implementation of FIFOs channels of RVC-CAL applications require the ability to check the number of tokens available during the execution due to the firing rule of DPN model. This makes the implementation of FIFO channels quite challenging. Additionally, memory architectures may affect an efficient implementation of dynamic dataflow application. Thus we need to have a smart memory architecture to deal with non-deterministic of DPN model concerning the checking of incoming tokens without consuming them, to evaluate which action will be fired.

Bibliography

[1] "IEEE Standard for Information Technology - Portable Operating System Interface (POSIX). Shell and Utilities," *IEEE Std 1003.1, 2004 Edition The Open Group Technical Standard. Base Specifications, Issue 6. Includes IEEE Std 1003.1-2001, IEEE Std 1003.1-2001/Cor 1-2002 and IEEE Std 1003.1-2001/Cor 2-2004. Shell*, 2004.

[2] T. O. A. R. Board. (2009) The openmp specification for parallel programming. [Online]. Available: http://www.openmp.org

[3] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI-The Complete Reference, Volume 1: The MPI Core*, 2nd ed. Cambridge, MA, USA: MIT Press, 1998.

[4] J. Diaz, C. Munoz-Caro, and A. Nino, "A survey of parallel programming models and tools in the multi and many-core era," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 8, pp. 1369–1386, Aug 2012.

[5] K. Group. (2013) Opencl - the open standard for parallel programming of heterogeneous systems. [Online]. Available: http://www.khronos.org/opencl/

[6] NVIDIA. (2013) Cuda: Parallel programming made easey. [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html

[7] M. Wolf, *High performance embedded computing : architectures, applications, and methodologies*, 1st ed. Amsterdam : Morgan Kaufmann, 2007.

[8] Gartner. (2014). [Online]. Available: http://www.gartner.com/newsroom/id/2791017

[9] F. Pereira, "Video compression: An evolving technology for better user experiences," in *Telecommunications (CONATEL), 2011 2nd National Conference on*, May 2011, pp. 1–6.

[10] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, ser. AFIPS '67 (Spring). New York, NY, USA: ACM, 1967, pp. 483–485. [Online]. Available: http://doi.acm.org/10.1145/1465482.1465560

[11] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, Jul. 2008. [Online]. Available: http://dx.doi.org/10.1109/MC.2008.209

[12] A. Kumar, H. Corporaal, B. Mesman, and Y. Ha, *Multimedia Multiprocessor Systems : Analysis, Design and Management*. Springer, 2010.

[13] B. Barney. (2014) Introduction to parallel computing. [Online]. Available: https://computing.llnl.gov/tutorials/parallel_comp

[14] H. Yviquel, "From dataflow-based video coding tools to dedicated embedded multi-core platforms," Ph.D. dissertation, University of Rennes 1, 2013.

[15] S. Stuijk, M. Geilen, B. Theelen, and T. Basten, "Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications," in *Int. Conf on Embedded Computer Systems (SAMOS)*, 2011.

[16] A. Ferrari and A. Sangiovanni-Vincentelli, "System design: traditional concepts and new paradigms," in *Computer Design, 1999. (ICCD '99) International Conference on*, 1999, pp. 2–12.

[17] K. Keutzer, A. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, "System-level design: orthogonalization of concerns and platform-based design," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 19, no. 12, pp. 1523–1543, Dec 2000.

[18] B. Kienhuis, E. Deprettere, K. Vissers, and P. van der Wolf, "An approach for quantitative analysis of application-specific dataflow architectures," in *Application-Specific Systems, Architectures and Processors, 1997. Proceedings., IEEE International Conference on*, July 1997, pp. 338–349.

[19] I. Bacivarov, W. Haid, K. Huang, and L. Thiele, "Methods and tools for mapping process networks onto multi-processor systems-on-chip," in *Handbook of Signal Processing Systems*, S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, Eds. Springer US, 2010, pp. 1007–1040. [Online]. Available: http://dx.doi.org/10.1007/978-1-4419-6345-1_35

[20] N. Siret, I. Sabry, J. Nezan, and M. Raulet, "A codesign synthesis from an mpeg-4 decoder dataflow description," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, May 2010, pp. 1995–1998.

[21] G. Kahn, "The semantics of a simple language for parallel programming," in *Information processing*, J. L. Rosenfeld, Ed. Stockholm, Sweden: North Holland, Amsterdam, Aug 1974, pp. 471–475.

[22] E. Lee and T. Parks, "Dataflow process networks," *Proceedings of the IEEE*, vol. 83, no. 5, pp. 773–801, May 1995.

[23] E. Lee and D. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, Sept 1987.

[24] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete, "Cycle-static dataflow," *Signal Processing, IEEE Transactions on*, vol. 44, no. 2, pp. 397–408, Feb 1996.

[25] J. Buck and E. Lee, "Scheduling dynamic dataflow graphs with bounded memory using the token flow model," in *Acoustics, Speech, and Signal Processing, 1993. ICASSP-93., 1993 IEEE International Conference on*, vol. 1, April 1993, pp. 429–432 vol.1.

[26] B. Bhattacharya and S. Bhattacharyya, "Parameterized dataflow modeling for dsp systems," *Signal Processing, IEEE Transactions on*, vol. 49, no. 10, pp. 2408–2421, Oct 2001.

[27] H. Yviquel, E. Casseau, M. Wipliez, J. Gorin, and M. Raulet, "Classification-based optimization of dynamic dataflow programs," in *Advancing Embedded Systems and Real-Time Communications with Emerging Technologies*. IGI Global, 2014, pp. 282–301. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01068648

[28] ISO/IEC, "Part 4: Codec configuration representation," Information Technology – MPEG Systems Technologies, ISO/IEC 23001-4, 2009.

[29] S. S. Bhattacharyya, J. Eker, J. W. Janneck, C. Lucarz, M. Mattavelli, and M. Raulet, "Overview of the mpeg reconfigurable video coding framework," *J. Signal Process. Syst.*, vol. 63, no. 2, pp. 251–263, May 2011. [Online]. Available: http://dx.doi.org/10.1007/s11265-009-0399-3

[30] J. Eker and J. W. Janneck, "Cal language report: Specification of the cal actor language," University of California, Berkeley, Tech. Rep., 2003.

[31] Orcc. The open rvc-cal compiler : A development framework for dataflow programs. [Online]. Available: http://orcc.sourceforge.net

[32] H. Yviquel, A. Lorence, K. Jerbi, G. Cocherel, A. Sanchez, and M. Raulet, "Orcc: Multimedia development made easy," in *Proceedings of the 21st ACM International Conference on Multimedia*, ser. MM '13. ACM, 2013, pp. 863–866.

[33] M. Wipliez, "Compilation infrastructure for dataflow programs," Ph.D. dissertation, INSA of Rennes / IETR, 2010.

[34] J. Gorin, H. Yviquel, F. Prêteux, and M. Raulet, "Just-in-time adaptive decoder engine: A universal video decoder based on mpeg rvc," in *Proceedings of the 19th ACM International Conference on Multimedia*, ser. MM '11. New York, NY, USA: ACM, 2011, pp. 711–714. [Online]. Available: http://doi.acm.org/10.1145/2072298.2072426

[35] H. Yviquel, A. Sanchez, P. Jääskeläinen, J. Takala, M. Raulet, and E. Casseau, "Embedded multi-core systems dedicated to dynamic dataflow programs," *Journal of Signal Processing Systems*, vol. 80, no. 1, pp. 121–136, 2015. [Online]. Available: http://dx.doi.org/10.1007/s11265-014-0953-5

[36] M. Wipliez and M. Raulet, "Classification of dataflow actors with satisfiability and abstract interpretation," *IJERTCS*, vol. 3, no. 1, pp. 49–69, 2012.

[37] S. Stuijk, M. Geilen, and T. Basten, "Sdf3: Sdf for free," in *6th Int. Conf. on Application of Concurrency to System Design,*, 2006.

[38] M. Pelcat, J. F. Nezan, J. Piat, J. Croizer, and S. Aridhi, "A system-level architecture model for rapid prototyping of heterogeneous multicore embedded systems," in *Conference on Design and Architectures for Signal and Image Processing (DASIP) 2009*, nice, France, Sep. 2009, p. 8 pages. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00429397

[39] J. W. Janneck, M. Mattavelli, M. Raulet, and M. Wipliez, "Reconfigurable video coding: A stream programming approach to the specification of new video coding standards," in *Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems*, ser. MMSys '10. New York, NY, USA: ACM, 2010, pp. 223–234. [Online]. Available: http://doi.acm.org/10.1145/1730836.1730864

[40] G. Sullivan, J. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1649–1668, Dec 2012.

[41] R. Marculescu, U. Ogras, L.-S. Peh, N. Jerger, and Y. Hoskote, "Outstanding research problems in noc design: System, microarchitecture, and circuit perspectives," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 1, pp. 3–21, Jan 2009.

[42] P. Marwedel, J. Teich, G. Kouveli, I. Bacivarov, L. Thiele, S. Ha, C. Lee, Q. Xu, and L. Huang, "Mapping of applications to mpsocs," in *Proceedings of the Seventh IEEE/ACM/IFIP International Conference*

on Hardware/Software Codesign and System Synthesis, ser. CODES+ISSS '11. New York, NY, USA: ACM, 2011, pp. 109–118. [Online]. Available: http://doi.acm.org/10.1145/2039370.2039390

[43] A. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: Survey of current and emerging trends," in *2013 50th ACM / EDAC / IEEE Design Automation Conference (DAC)*, May 2013, pp. 1–10.

[44] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[45] C. Lee, H. Kim, H.-w. Park, S. Kim, H. Oh, and S. Ha, "A task remapping technique for reliable multi-core embedded systems," in *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES/ISSS '10. New York, NY, USA: ACM, 2010, pp. 307–316. [Online]. Available: http://doi.acm.org/10.1145/1878961.1879014

[46] D.-T. Ngo, J.-P. Diguet, K. Martin, and D. Sepulveda, "Communication-model based embedded mapping of dataflow actors on heterogeneous mpsoc," in *Proceedings of the Conference on Design and Architectures for Signal and Image Processing (DASIP '14)*, 2014.

[47] MathWorks. (2013) Simulink – simulation and model-based design. [Online]. Available: http://www.mathworks.com/products/simulink/

[48] N. Instruments. (2013) Labview system design software. [Online]. Available: http://www.ni.com/labview/

[49] Synopsys. (2012) Signal processing worksystem (spw). [Online]. Available: http://www.synopsys.com/systems/blockdesign/digitalsignalprocessing/pages/signal-processing.aspx

[50] ——. (2012) System studio. [Online]. Available: http://www.synopsys.com/Systems/BlockDesign/DigitalSignalProcessing/Pages/SystemStudio.aspx

[51] E. Lee and D. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *Computers, IEEE Transactions on*, vol. C-36, no. 1, pp. 24–35, Jan 1987.

[52] Ptolemy. Heterogeneous modeling and design. [Online]. Available: http://ptolemy.eecs.berkeley.edu/index.htm

[53] S. Stuijk, T. Basten, M. Geilen, and H. Corporaal, "Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs," in *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, June 2007, pp. 777–782.

[54] A. Ghamarian, M. Geilen, T. Basten, and S. Stuijk, "Parametric throughput analysis of synchronous data flow graphs," in *Design, Automation and Test in Europe, 2008. DATE '08*, March 2008, pp. 116–121.

[55] A. Bonfietti, L. Benini, M. Lombardi, and M. Milano, "An efficient and complete approach for throughput-maximal sdf allocation and scheduling on multi-core platforms," in *DATE*, March 2010.

[56] S. Stuijk, M. Geilen, and T. Basten, "Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs," in *43rd ACM/IEEE DAC*, 2006, pp. 899–904.

[57] A. Stulova, R. Leupers, and G. Ascheid, "Throughput driven transformations of synchronous data flows for mapping to heterogeneous mpsocs," in *Embedded Computer Systems (SAMOS), 2012 International Conference on*, July 2012, pp. 144–151.

[58] J. Lin, A. Gerstlauer, and B. L. Evans, "Communication-aware heterogeneous multiprocessor mapping for real-time streaming systems," *Journal of Signal Processing Systems*, vol. 69, no. 3, pp. 279–291, Dec. 2012.

[59] A. Ghamarian, M. Geilen, S. Stuijk, T. Basten, A. Moonen, M. Bekooij, B. Theelen, and M. Mousavi, "Throughput analysis of synchronous data flow graphs," in *Application of Concurrency to System Design, 2006. ACSD 2006. Sixth International Conference on*, June 2006, pp. 25–36.

[60] W. Liu, M. Yuan, X. He, Z. Gu, and X. Liu, "Efficient sat-based mapping and scheduling of homogeneous synchronous dataflow graphs for throughput optimization," in *Real-Time Systems Symposium*, 2008.

[61] A. K. Singh, A. Kumar, and T. Srikanthan, "Accelerating throughput-aware runtime mapping for heterogeneous mpsocs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, no. 1, 2013.

[62] R. Jordans, F. Siyoum, S. Stuijk, A. Kumar, and H. Corporaal, "An automated flow to map throughput constrained applications to a mpsoc," in *Bringing Theory to Practice: Predictability and Performance in Embedded Systems*, ser. OpenAccess Series in Informatics (OASIcs), P. Lucas, L. Thiele, B. Triquet, T. Ungerer, and

R. Wilhelm, Eds., vol. 18. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2011, pp. 47–58. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2011/3081

[63] A. Moonen, M. Bekooij, R. van den Berg, and J. van Meerbergen, "Practical and accurate throughput analysis with the cyclo static dataflow model," in *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2007. MASCOTS '07. 15th International Symposium on*, Oct 2007, pp. 238–245.

[64] T. Parks, J. Pino, and E. Lee, "A comparison of synchronous and cyclestatic dataflow," in *Signals, Systems and Computers, 1995. 1995 Conference Record of the Twenty-Ninth Asilomar Conference on*, vol. 1, Oct 1995, pp. 204–210 vol.1.

[65] K. Desnos, M. Pelcat, J.-F. Nezan, S. Bhattacharyya, and S. Aridhi, "Pimm: Parameterized and interfaced dataflow meta-model for mpsocs runtime reconfiguration," in *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII), 2013 International Conference on*, July 2013, pp. 41–48.

[66] V. Bebelis, P. Fradet, A. Girault, and B. Lavigueur, "Bpdf: A statically analyzable dataflow model with integer and boolean parameters," in *Embedded Software (EMSOFT), 2013 Proceedings of the International Conference on*, Sept 2013, pp. 1–10.

[67] B. Theelen, M. Geilen, T. Basten, J. Voeten, S. Gheorghita, and S. Stuijk, "A scenario-aware data flow model for combined long-run average and worst-case performance analysis," in *Formal Methods and Models for Co-Design, 2006. MEMOCODE '06. Proceedings. Fourth ACM and IEEE International Conference on*, July 2006, pp. 185–194.

[68] L. Schor, I. Bacivarov, D. Rai, H. Yang, S.-H. Kang, and L. Thiele, "Scenario-based design flow for mapping streaming applications onto on-chip many-core systems," in *Proceedings of the 2012 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, ser. CASES '12. New York, NY, USA: ACM, 2012. [Online]. Available: http://doi.acm.org/10.1145/2380403.2380422

[69] J. Castrillon, R. Leupers, and G. Ascheid, "Maps: Mapping concurrent dataflow applications to heterogeneous mpsocs," *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 1, pp. 527–545, Feb 2013.

[70] W. Quan and A. D. Pimentel, "A scenario-based run-time task mapping algorithm for mpsocs," in *Proceedings of the 50th Annual Design Automation Conference*, ser. DAC '13. New York, NY, USA: ACM, 2013, pp. 131:1–131:6. [Online]. Available: http://doi.acm.org/10.1145/2463209.2488895

[71] H. Yviquel, E. Casseau, M. Raulet, P. Jääskeläinen, and J. Takala, "Towards run-time actor mapping of dynamic dataflow programs onto multi-core platforms," in *Int. Symp. on Image and Signal Processing and Analysis (ISPA)*, France, 2013.

[72] S. S. Bhattacharyya, G. Brebner, J. W. Janneck, J. Eker, C. von Platen, M. Mattavelli, and M. Raulet, "Opendf: A dataflow toolset for reconfigurable hardware and multicore systems," *SIGARCH Comput. Archit. News*, vol. 36, no. 5, pp. 29–35, Jun. 2009. [Online]. Available: http://doi.acm.org/10.1145/1556444.1556449

[73] J. Janneck, I. Miller, D. Parlour, G. Roquier, M. Wipliez, and M. Raulet, "Synthesizing hardware from dataflow programs," *Journal of Signal Processing Systems*, vol. 63, no. 2, pp. 241–249, 2011. [Online]. Available: http://dx.doi.org/10.1007/s11265-009-0397-5

[74] G. Roquier, M. Wipliez, M. Raulet, J. Janneck, I. Miller, and D. Parlour, "Automatic software synthesis of dataflow program: An mpeg-4 simple profile decoder case study," in *Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on*, Oct 2008, pp. 281–286.

[75] C. von Platen and J. Eker, "Efficient realization of a cal video decoder on a mobile terminal (position paper)," in *Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on*, Oct 2008, pp. 176–181.

[76] M. Wipliez, G. Roquier, and J.-F. Nezan, "Software code generation for the rvc-cal language," *Journal of Signal Processing Systems*, vol. 63, no. 2, pp. 203–213, 2011. [Online]. Available: http://dx.doi.org/10.1007/s11265-009-0390-z

[77] C. Lucarz, M. Mattavelli, M. Wipliez, G. Roquier, M. Raulet, J. Janneck, I. Miller, and D. Parlour, "Dataflow/Actor-Oriented language for the design of complex signal processing systems," in *Proceedings of the 2008 Conference on Design and Architectures for Signal and Image Processing, DASIP 2008*, 2008, pp. 168–175.

[78] C. Lucarz, G. Roquier, and M. Mattavelli, "High level design space exploration of rvc codec specifications for multi-core heterogeneous platforms," in

Design and Architectures for Signal and Image Processing (DASIP), 2010 Conference on, Oct 2010, pp. 191–198.

[79] G. Roquier, E. Bezati, and M. Mattavelli, "Hardware and software synthesis of heterogeneous systems from dataflow programs," *JECE*, vol. 2012, pp. 2:2–2:2, Jan. 2012. [Online]. Available: http://dx.doi.org/10.1155/2012/484962

[80] H. Yviquel, J. Boutellier, M. Raulet, and E. Casseau, "Automated design of networks of transport-triggered architecture processors using dynamic dataflow programs," *Signal Processing: Image Communication*, vol. 28, no. 10, pp. 1295 – 1302, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0923596513001306

[81] J. Castrillon-Mazo, "Programming heterogeneous mpsocs: Tool flows to close the software productivity gap," Ph.D. dissertation, Faculty of Electrical Engineering and Information Technology, RWTH Aachen University, 2013.

[82] T. Instruments. (2014) Omap mobile processors. [Online]. Available: http://www.ti.com/lsds/ti/omap-applications-processors/features.page

[83] Qualcomm. (2011) Snapdragon s4 processors: System on chip solutions for a new mobile age. [Online]. Available: https://www.qualcomm.com/documents/snapdragon-s4-processors-system-chip-solutions-new-mobile-age

[84] Y.-S. Cho, E.-J. Choi, and K.-R. Cho, "Modeling and analysis of the system bus latency on the SoC platform," in *Proceedings of the 2006 International Workshop on System-level Interconnect Prediction*, ser. SLIP '06. New York, NY, USA: ACM, 2006, pp. 67–74.

[85] A. K. Singh, T. Srikanthan, A. Kumar, and W. Jigang, "Communication-aware heuristics for run-time task mapping on noc-based mpsoc platforms," *J. Syst. Archit.*, vol. 56, no. 7, pp. 242–255, Jul. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.sysarc.2010.04.007

[86] J. Castrillon, A. Tretter, R. Leupers, and G. Ascheid, "Communication-aware mapping of kpn applications onto heterogeneous mpsocs," in *Proceedings of the 49th Annual Design Automation Conference*, ser. DAC '12. New York, NY, USA: ACM, 2012, pp. 1266–1271. [Online]. Available: http://doi.acm.org/10.1145/2228360.2228597

[87] S. Foroutan, Y. Thonnart, and F. Petrot, "An iterative computational technique for performance evaluation of networks-on-chip," *Computers, IEEE Transactions on*, vol. 62, no. 8, pp. 1641–1655, Aug 2013.

[88] E. Carvalho and F. Moraes, "Congestion-aware task mapping in heterogeneous mpsocs," in *System-on-Chip, 2008. SOC 2008. International Symposium on*, Nov 2008, pp. 1–4.

[89] S. Kaushik, A. Singh, and T. Srikanthan, "Computation and communication aware run-time mapping for noc-based mpsoc platforms," in *SOC Conference (SOCC), 2011 IEEE International*, Sept 2011, pp. 185–190.

[90] C. Lee, S. Kim, and S. Ha, "Efficient run-time resource management of a manycore accelerator for stream-based applications," in *Embedded Systems for Real-time Multimedia (ESTIMedia), 2013 IEEE 11th Symposium on*, Oct 2013, pp. 51–60.

[91] T. Instruments. Keystone device architecture. [Online]. Available: http://processors.wiki.ti.com/index.php/Keystone_Device_Architecture

[92] Synopsys. Processor designer. [Online]. Available: http://www.synopsys.com/Tools/SLD/ProcessorDev/Pages/default.aspx

[93] ——. Platform architect. [Online]. Available: http://www.synopsys.com/Tools/SLD/VirtualPrototyping/Pages/PlatformArchitect.aspx

[94] Xilinx. (2015) Zynq-7000 all programmable soc. [Online]. Available: http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

[95] H. Orsila, T. Kangas, E. Salminen, T. D. Hämäläinen, and M. Hännikäinen, "Automated memory-aware application distribution for multi-processor system-on-chips," *J. Syst. Archit.*, vol. 53, no. 11, pp. 795–815, Nov. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.sysarc.2007.01.013

[96] Y. Yi, W. Han, X. Zhao, A. Erdogan, and T. Arslan, "An ilp formulation for task mapping and scheduling on multi-core architectures," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, April 2009, pp. 33–38.

[97] J. Falk, C. Zebelein, C. Haubelt, and J. Teich, "A rule-based static dataflow clustering algorithm for efficient embedded software synthesis," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, March 2011, pp. 1–6.

[98] P. K. Sahu and S. Chattopadhyay, "A survey on application mapping strategies for network-on-chip design," *J. Syst. Archit.*, vol. 59, no. 1, pp. 60–76, Jan. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.sysarc.2012.10.004

[99] A. H. Hormati, Y. Choi, M. Kudlur, R. Rabbah, T. Mudge, and S. Mahlke, "Flextream: Adaptive compilation of streaming applications for heterogeneous architectures," in *Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 214–223. [Online]. Available: http://dx.doi.org/10.1109/PACT.2009.39

[100] N. Satish, K. Ravindran, and K. Keutzer, "A decomposition-based constraint optimization approach for statically scheduling task graphs with communication delays to multiprocessors," in *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, April 2007, pp. 1–6.

[101] A. Kumar, B. Mesman, H. Corporaal, and Y. Ha, "Iterative probabilistic performance prediction for multi-application multiprocessor systems," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 29, no. 4, pp. 538–551, April 2010.

[102] S. Stuijk, M. Geilen, and T. Basten, "A predictable multiprocessor design flow for streaming applications with dynamic behaviour," in *Euromicro Conference on Digital System Design: Architectures, Methods and Tools (DSD)*. Washington, DC, USA: IEEE Computer Society, 2010. [Online]. Available: http://dx.doi.org/10.1109/DSD.2010.31

[103] E. de Souza Carvalho, N. Calazans, and F. Moraes, "Dynamic task mapping for mpsocs," *Design Test of Computers, IEEE*, vol. 27, no. 5, pp. 26–35, Sept 2010.

[104] V. Nollet, P. Avasare, H. Eeckhaut, D. Verkest, and H. Corporaal, "Run-time management of a mpsoc containing fpga fabric tiles," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 1, pp. 24–33, Jan 2008.

[105] J. Huang, A. Raabe, C. Buckl, and A. Knoll, "A workflow for runtime adaptive task allocation on heterogeneous mpsocs," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, March 2011, pp. 1–6.

[106] A. Schranzhofer, J.-J. Chen, L. Santinelli, and L. Thiele, "Dynamic and adaptive allocation of applications on mpsoc platforms," in *Design Au-*

tomation Conference (ASP-DAC), 2010 15th Asia and South Pacific, Jan 2010, pp. 885–890.

[107] W. Quan and A. D. Pimentel, "A hybrid task mapping algorithm for heterogeneous mpsocs," *ACM Trans. Embed. Comput. Syst.*, vol. 14, no. 1, pp. 14:1–14:25, Jan. 2015. [Online]. Available: http://doi.acm.org/10.1145/2680542

[108] METIS. Serial graph partitioning and fill-reducing matrix ordering. [Online]. Available: http://glaros.dtc.umn.edu/gkhome/views/metis

[109] O. Derin, D. Kabakci, and L. Fiorin, "Online task remapping strategies for fault-tolerant network-on-chip multiprocessors," in *Networks on Chip (NoCS), 2011 Fifth IEEE/ACM International Symposium on*, May 2011, pp. 129–136.

[110] VSP. Cadence virtual system platform for the xilinx zynq-7000 all programmable soc. [Online]. Available: http://www.cadence.com/products/sd/virtual_system

[111] OVP. Open virtual platform. [Online]. Available: http://www.ovpworld.org

[112] G. Karypis and V. Kumar, "Multilevel algorithms for multi-constraint graph partitioning," in *ACM/IEEE Conf. on Supercomputing*, Washington, DC, USA, 1998.

[113] ——, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, Dec. 1998.

[114] S. K. Lim, *Practical Problems in VLSI Physical Design Automation*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[115] A. Agarwal, J. Hennessy, and M. Horowitz, "An analytical cache model," *ACM Trans. Comput. Syst.*, vol. 7, no. 2, pp. 184–215, May 1989. [Online]. Available: http://doi.acm.org/10.1145/63404.63407

[116] M. Pittau, A. Alimonda, S. Carta, and A. Acquaviva, "Impact of task migration on streaming multimedia for embedded multiprocessors: A quantitative evaluation," in *Embedded Systems for Real-Time Multimedia, 2007. ESTIMedia 2007. IEEE/ACM/IFIP Workshop on*, Oct 2007, pp. 59–64.

[117] L. A. Sanchis, "Multiple-way network partitioning," *IEEE Trans. Comput.*, vol. 38, no. 1, pp. 62–81, Jan. 1989. [Online]. Available: http://dx.doi.org/10.1109/12.8730

[118] Xiph. (2015) Xiph.org video test media. [Online]. Available: https://media.xiph.org/video/derf/