UNIUNEA EUROPEANĂ
GUVERNUL ROMÂNIEI MINISTERUL MUNCII, FAMILIEI ŞI PROTECŢIEI SOCIALE AMPOSDRU
Fondul Social European POS DRU 2007-2013
Instrumente Structurale 2007-2013
MINISTERUL EDUCAŢIEI CERCETĂRII TINERETULUI ŞI SPORTULUI
OIPOSDRU
Universitatea Lucian Blaga Sibiu

Investeşte în oameni!

PROIECT FINANŢAT DIN FONDUL SOCIAL EUROPEAN

ID proiect: 7706

Titlul proiectului: „Creşterea rolului studiilor doctorale şi a competitivităţii doctoranzilor într-o Europă unită”

Universitatea”Lucian Blaga” din Sibiu

B-dul Victoriei, nr. 10. Sibiu

Facultatea de Inginerie “Hermann Oberth”

Domeniul de doctorat Calculatoare şi Tehnologia Informaţiei

Optimized Algorithms for Network-on-Chip Application Mapping

PhD Thesis
Abstract

Author:
Ciprian RADU, MSc

PhD Supervisor:
Professor Lucian N. Vinţan, PhD

SIBIU, September 2011

UNIUNEA EUROPEANĂ

GUVERNUL ROMÂNIEI
MINISTERUL MUNCII, FAMILIEI ŞI PROTECŢIEI SOCIALE
AMPOSDRU

Fondul Social European
POS DRU 2007-2013

Instrumente Structurale
2007-2013

MINISTERUL EDUCAŢIEI CERCETĂRII TINERETULUI ŞI SPORTULUI
OIPOSDRU

Universitatea
Lucian Blaga
Sibiu

Investeşte în oameni!

PROIECT FINANŢAT DIN FONDUL SOCIAL EUROPEAN

ID proiect: 7706

Titlul proiectului: „Creşterea rolului studiilor doctorale şi a competitivităţii doctoranzilor într-o Europă unită”

Universitatea”Lucian Blaga” din Sibiu

B-dul Victoriei, nr. 10. Sibiu

Facultatea de Inginerie “Hermann Oberth”

Domeniul de doctorat Calculatoare şi Tehnologia Informaţiei

Algoritmi optimizați pentru maparea aplicațiilor pe arhitecturi de tipul Network-on-Chip

Teză de doctorat
Rezumat

Autor:
Ing. Ciprian RADU

Conducător ştiinţific:
Prof. univ. dr. ing. Lucian N. Vinţan

SIBIU, Septembrie 2011

Dedicată fratelui meu... Dedicated to my brother...

http://alexraduland.wordpress.com/

Mulţumiri

Munca prezentată în această teză de doctorat a fost efectuată în Centrul de Cercetare pentru Arhitecturi Avansate de Procesare a Informației (Advanced Computer Architecture and Processing Systems – ACAPS – http://acaps.ulbsibiu.ro) al Universităţii "Lucian Blaga" din Sibiu, România, în perioada 2008 – 2011.

Mulţumesc coordonatorului meu ştiinţific, domnul profesor *Lucian Vinţan*, pentru încurajarea şi ghidarea mea către cariera doctorală. Coordonarea ştiinţifică, sfaturile, corectările riguroase, comentariile constructive şi suportul său necondiţionat au fost esenţiale pentru succesul meu, încă din perioada când eram doar student.

Mulţumiri, de asemenea, domnului profesor *Theo Ungerer* de la Universitatea din Augsburg din Germania pentru că mi-a permis să fac parte din echipa sa de cercetare timp de cinci luni, ca stagiu de pregătire în străinătate a doctoratului meu. Perioada din Augsburg a fost plină de inspiraţie. Am câştigat multă experienţă şi am obţinut sfaturi folositoare.

În timpul doctoratului am avut plăcerea să lucrez cu prietenul şi colegul meu, *Horia Calborean*. Aş dori să îi mulţumesc pentru buna colaborare şi pentru observaţiile sale valoroase.

Aş dori să mulţumesc, de asemenea, tuturor membrilor Catedrei de Calculatoare, în special domnului conferenţiar univ. dr. ing. *Remus Brad*, domnului conferenţiar univ. dr. ing. *Adrian Florea* şi domnului asistent univ. dr. ing. *Árpád Gellért*. A fost o plăcere să lucrăm împreună.

Le sunt profund recunoscător şi domnului profesor *Nicolae Ţăpuş* şi echipei de cercetare de la Universitatea Politehnica din Bucureşti, România. Aş dori să mulţumesc în special domnului conferenţiar univ. dr. ing. *Emil Slusanschi* şi domnului asistent univ. *Alexandru Herişanu* pentru ajutorul oferit în exploatarea sistemului nostru HPC şi pentru că mi-au permis şi m-au ajutat să obţin o parte din rezultatele experimentale pe supercalculatorul de la universitatea dumnealor.

În plus, mulţumesc celorlalţi co-autori şi colegi, *Shahriar Mahbub* (masterat) şi *Andreea Gancea* (licenţă), cu care de asemenea am lucrat.

Vreau să îmi exprim sincera şi profunda recunoştinţă familiei mele şi *Nicoletei*, pentru sprijinul şi înţelegerea oferite.

Această lucrare a fost susţinută de contractul financiar POSDRU 7706: *Creşterea rolului studiilor doctorale şi a competitivităţii doctoranzilor într-o Europă unită cofinanţat din Fondul Social European prin Programul Operaţional Sectorial Dezvoltarea Resurselor Umane 2007 - 2013.*

Sibiu, Septembrie 2011
Ciprian Radu
http://webspace.ulbsibiu.ro/ciprian.radu/

Rezumat

În zilele noastre, tendinţele tehnologice au determinat arhitecturile de calculatoare să ajungă la aşa-numitul *power wall*. Datorită continuei micşorări a tranzistorilor, densitatea de putere pe centimetru pătrat a ajuns la limita superioară. Din această cauză, arhitecţii de calculatoare au hotărât să înceteze îmbunătăţirea performanţei *design*-urilor acestora prin intermediul scalării frecvenţei. În loc de aceasta, mai multe procesoare sunt plasate pe acelaşi *chip*. Sistemele *multicore* şi *manycore* oferă o performanţă crescută faţă de arhitecturile cu un singur *core* (nucleu de procesare), prin efectuarea de procesare paralelă. De asemenea, arhitecturile de calculator specifice pentru aplicaţii îmbunătăţesc performanţa prin utilizarea de procesoare eterogene în locul celor omogene. Evident, astfel de arhitecturi trebuie să fie interconectate pentru a comunica. Potrivit viziunii HiPEAC [1], în momentul de faţă comunicarea defineşte performanţa. Reţelele de interconectare au o foarte mare importanţă. Cele bazate pe magistrală de transmisie (bus) nu sunt potrivite pentru sistemele *multicore* şi *manycore* pentru că ele nu scalează [2].

După anul 2000, reţele interconectate pe *chip*, numite arhitecturi *Network-on-Chip* (NoC), au fost propuse drept o alternativă fezabilă pentru reţelele bus. Reţelele NoC au avantaje importante cum ar fi modularitatea şi scalabilitatea, dar sunt şi extrem de limitate în resurse. Ca urmare, există multe probleme de cercetare în domeniul NoC [3].

Maparea aplicaţiilor pe arhitecturi de tipul *Network-on-Chip* este una dintre cele mai oneroase probleme (NP completă), în această zonă de cercetare. De vreme ce o abordare exhaustivă este nefezabilă, pentru această problemă sunt folosiţi algoritmi euristici. *Scopul acestei teze este să evalueze şi să optimizeze algoritmi (mono-obiectiv şi multi-obiectiv) pentru maparea aplicaţiilor pe arhitecturi de tipul Network-on-Chip*.

Primul obiectiv al acestei teze este să se prezinte stadiul actual al algoritmilor proiectați pentru problema mapării aplicațiilor pe arhitecturi Network-on-Chip. Apoi, propunem de asemenea o taxonomie pentru aceşti algoritmi.

Zona de cercetare a arhitecturilor *Network-on-Chip* este relativ nouă. Ca atare, unelte puternice şi mature sunt încă aşteptate. Din câte ştim, la această dată nu există un cadru unitar *open source* (gratuit) pentru evaluarea şi optimizarea algoritmilor pentru maparea aplicațiilor pe arhitecturi de tipul *Network-on-Chip*. Cel de al doilea obiectiv al nostru este să proiectăm un cadru comun pentru evaluarea şi optimizarea algoritmilor pentru diferite mapări pe arhitecturi multiple de tipul *Network-on-Chip*.

Al treilea obiectiv este să optimizăm şi să adaptăm un algoritm de tipul *Simulated Annealing* pentru maparea aplicațiilor pe NoCuri, folosind cunoştinţe de domeniu.

Al patrulea obiectiv constă în evaluarea şi optimizarea (folosind cunoştinţe de domeniu) algoritmilor evolutivi pentru maparea multi-obiectiv a aplicațiilor pe NoCuri.

În cele din urmă, ne propunem să efectuăm o explorare automată, ghidată de aplicație, a spațiului arhitectural pentru Sisteme *on Chip*. Aceasta implică sisteme specifice aplicaţiilor, cu procesoare eterogene, utilizând o reţea NoC parametrizabilă.

Această teză aduce contribuţii originale în optimizarea sistemelor de tipul *Network-on-Chip*. Contribuim cu unelte pentru simulare şi *benchmarking*. Optimizăm algoritmi pentru problema mapării aplicaţiilor pe arhitecturi NoC. De asemenea, propunem o metodă de explorare automată, ghidată de aplicație, a spațiului arhitectural pentru Sisteme *on Chip*.

“Lucian Blaga” University of Sibiu
“Hermann Oberth” Engineering Faculty
Computer Engineering Department

Optimized Algorithms for Network-on-Chip Application Mapping

PhD Thesis

Abstract

Author:
Ciprian RADU, MSc

PhD Supervisor:
Professor Lucian N. Vinţan, PhD

SIBIU, September 2011

Acknowledgments

The work presented in this PhD Thesis has been carried out in the Advanced Computer Architecture and Processing Systems (ACAPS) research lab (http://acaps.ulbsibiu.ro) at "Lucian Blaga" University of Sibiu, Romania, during the years 2008 – 2011.

I thank my PhD supervisor, Professor *Lucian Vinţan*, for encouraging and guiding me towards the Doctoral degree. His scientific coordination, his advices, his thorough reviews, his constructive comments and his generous support were essential for my success, starting from the period when I was just an undergraduate student.

Grateful acknowledgements go also to Professor *Theo Ungerer* from University of Augsburg, Germany, for kindly allowing me to be part of his research team for five months, as my PhD external research stage. My research stage in Augsburg was inspiring. I gained a lot of experience and obtained good pieces of advice.

During my PhD work, I had the pleasure to work with my friend and colleague, *Horia Calborean*. I would like to thank him for the good collaboration we had and for his valuable observations.

I would also like to thank to all the members from the Computer Engineering Department, especially to Associate Professor Dr. Ing. *Remus Brad*, Associate Professor Dr. Ing. *Adrian Florea* and to Assistant Professor Dr. Ing. *Árpád Gellért*. It has been my pleasure working with them.

My deep gratitude goes as well to Professor *Nicolae Ţăpuş* and to his research staff from Politehnica University of Bucharest, Romania. I would like to thank especially to Associate Professor Dr. Ing. *Emil Slusanschi* and to Assistant Professor *Alexandru Herişanu* for helping me with our HPC system and for allowing and helping me do part of my experimental results on their university supercomputer.

In addition, I thank to all the other co-authors and colleagues, *Shahriar Mahbub*, MSc and *Andreea Gancea*, BSc, with whom I have also worked.

I want to express my sincere and deep gratitude to my family and to *Nicoleta*, for their support and understanding.

This work was supported by POSDRU financing contract POSDRU 7706.

Sibiu, September 2011
Ciprian Radu
http://webspace.ulbsibiu.ro/ciprian.radu/

Author's Papers

Ciprian Radu, Lucian Vinţan, *Domain-Knowledge Optimized Simulated Annealing for Network-on-Chip Application Mapping*, Submitted to an Elsevier journal, September 2011.

Ciprian Radu, Shahriar Mahbub, Lucian Vinţan, *Developing Domain-Knowledge Evolutionary Algorithms for Network-on-Chip Application Mapping*, in review (since July 25th, 2011) at Journal of Systems Architecture (**Impact Factor: 0.667; 5-Year Impact Factor: 0.768**), July 2011.

Ciprian Radu, Lucian Vinţan, *UNIMAP: UNIFIED FRAMEWORK FOR NETWORK-ON-CHIP APPLICATION MAPPING RESEARCH*, Acta Universitatis Cibiniensis – Technical Series, "Lucian Blaga" University of Sibiu, Romania, ISSN 1583-7149, May 2011, Sibiu, Romania.

Ciprian Radu, Lucian Vinţan, *Optimized Simulated Annealing for Network-on-Chip Application Mapping*, Proceedings of the 18th International Conference on Control Systems and Computer Science (CSCS-18), Politehnica Press, pp. 452-459, ISSN 2066-4451, 24 - 27 May 2011, Bucharest, Romania.

Ciprian Radu, Lucian Vinţan, *Towards a Unified Framework for the Evaluation and Optimization of NoC Application Mapping Algorithms*, Sixth International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems (ACACES), Academic Press, Ghent, Belgium, pp. 163-166, ISBN 978-90-382-1631-7, July 14, 2010, Terrassa (Barcelona), Spain.

Ciprian Radu and Lucian Vinţan, *Optimizing application mapping algorithms for NoCs through a unified framework*, In Proceedings of the 9-th IEEE RoEduNet International Conference, Sibiu, Romania, pp. 259 - 264, ISBN 978-1-4244-7335-9, 24-26 June 2010. IEEE Computer Society. Indexed IEEE, ISI , Scopus

Ciprian Radu, Horia Calborean, Adrian Florea, Arpad Gellert, Lucian Vinţan, *Exploring some multicore research opportunities. A first attempt*, Fifth International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems (ACACES), Academic Press, Ghent, Belgium, pp. 151-154, ISBN 978-90-382-1467-2, July 2009, Terrassa (Barcelona), Spain.

Adrian Florea, **Ciprian Radu**, Horia Calborean, Adrian Crapciu, Arpad Gellert, Lucian Vinţan, *Understanding and Predicting Unbiased Branches in General-Purpose Applications*, Buletinul Institutului Politehnic Iasi, Tome LIII (LVII), fasc. 1-4, Section IV, Automation Control and Computer Science Section, pp. 97-112, ISSN 1220-2169, "Gh. Asachi" Technical University, 2007, Iaşi, Romania. Indexed Zentralblatt MATH

Adrian Florea, **Ciprian Radu**, Horia Calborean, Adrian Crapciu, Arpad Gellert, Lucian Vinţan, *Designing an Advanced Simulator for Unbiased Branches' Prediction*, Proceedings of 9th International Symposium on Automatic Control and Computer Science, ISSN 1843-665X, November 2007, Iaşi, Romania.

Ciprian Radu, Horia Calborean, Adrian Crapciu, Arpad Gellert, Adrian Florea, *An Interactive Graphical Trace-Driven Simulator for Teaching Branch Prediction in Computer Architecture*, The 6th EUROSIM Congress on Modelling and Simulation, (EUROSIM 2007), ISBN 978-3-901608-32-2, 9-13 September 2007, Ljubljana, Slovenia (special session: *Education in Simulation / Simulation in Education I*).

Contents

1 Introduction

In the current days, the technology trends determined computer architectures to reach the so called power wall. Due to continuously shrinking transistors, the power density per square centimeter reached the upper limit. Because of this, computer architects decided to stop improving the performance of their designs by means of frequency scaling. Rather than this, more processors are placed on the same chip. Multicore and manycore systems provide better performance than single core architectures, by performing parallel processing. Also, application specific computer architectures yield increased performance by employing heterogeneous processors instead of homogenous processors. Obviously, such architectures must be interconnected in order to communicate. According to HiPEAC's vision [1], nowadays communication defines performance. Interconnection networks are of high importance. Traditional bus-based networks are not suitable for multicores and manycores, because they do not scale [2].

After year 2000 on chip interconnection networks, called Network-on-Chip (NoC) architectures have been proposed as a feasible alternative to bus networks. NoCs have important advantages like modularity and scalability but, they are also extremely resource limited. As such, there are many outstanding research problems in the NoC field [3].

Network-on-Chip application mapping is one of the most onerous, NP-hard, problems in this area of research. Since an exhaustive approach is infeasible, heuristic algorithms are used to address this problem. *The scope of this thesis is to evaluate and optimize Network-on-Chip application mapping algorithms (using single-objective and multi-objective approaches).*

The first objective of this thesis is to realize a state of the art regarding the algorithms designed for the Network-on-Chip application mapping problem. Then we also propose a taxonomy for these algorithms.

The Network-on-Chip research field is relatively new. Therefore powerful and mature tools are still expected. To the best of our knowledge, there is not currently an open source unified framework for the evaluation and optimization of Network-on-Chip application mapping algorithms. Therefore, our second objective is to design a framework that uses a common frame for evaluating and optimizing different state of the art mapping algorithms on multiple NoC architectures.

The third objective of this work is to adapt and optimize a general Simulated Annealing technique, for NoC application mapping, using domain-knowledge.

Our forth objective is to evaluate and optimize (using domain-knowledge) evolutionary algorithms, for Network-on-Chip application mapping, through a multi-objective approach.

Finally, we aim to perform an application driven automatic design space exploration of System-on-Chip designs. This involves entire application specific systems, with heterogeneous processors, using a NoC as interconnection.

This thesis brings original contributions in the Network-on-Chip research field. We contribute with tools for simulating and benchmarking NoC designs. We optimize algorithms for the NoC application mapping problem. We also propose an application driven automatic design space exploration method for System-on-Chip architectures.

2 Network-on-Chip Architectures

Since the invention of the integrated circuit in 1958, Moore's law [4] describes a trend in Computer Engineering that is still nowadays. For more than half a century, the number of transistors that can be placed onto a single chip doubles approximately every two years (initially it was one year, than Moore readapted its law) [5]. In the early beginnings, a computer system occupied an entire room. As technology evolved, in the 70s the Large Scale Integration (LSI) era began and the computers were rack-level systems. In the 80s, Very Large Scale Integration (VLSI) era began. This meant a system can be placed on a single board. Ten years later, in the 90s, we went to chip-level systems (ULSI – Ultra Large Scale Integration). Nowadays, billion transistors can be integrated on a single die. A chip is an entire system and so, the term System-on-Chip (SoC) was coined. Systems-on-Chip make use of parallel processing at all levels: Instruction Level Parallelism (ILP), Memory Level Parallelism (MLP) and Thread Level Parallelism (TLP) [6], [7], [8], [9]. We researched these levels of parallelism previously by focusing on branch prediction [10], [11], [12] and multicore architectures [13], [14]. SoCs are feasible for a wide range of applications. However, they determine the architects to focus on the complex aspects of the communication architecture.

The continuously growing number of transistors per chip leads to a bigger and bigger gap between logic gate delays and wire delays [15]. As compared with the gate delays, the global interconnection wires used by a typical bus interconnection network determine significantly higher delays.

Systems-on-Chip also incur problems related to complexity, design flexibility and productivity and system synchronization. Achieving global synchronization is getting harder and harder as technology advances and chip speed increases.

Currently, computer architects face with the difficult problem called Power Wall. The Power Wall is what determined the appearance of multicore and manycore architectures [16]. Parallel programming is needed to exploit multicores. Obviously, such architectures require scalable interconnection networks. It is well-known that the bus is not a scalable interconnection network [2].

The gap between on-chip and off-chip communication is increasing. On-chip, we have greater bandwidth and shorter latencies but, the power budget is smaller. Besides *scalability*, on-chip communication also means *flexibility*, *simplicity* and *efficiency*. Flexibility is achieved by no longer using application-specific wiring (like buses do). Simplicity refers to modular, structured and regular design. Efficiency means the interconnection's ability to share global wires between different communication flows. Communication is a performance bottleneck. Because of this, the design shifts from a processing-centric to a communication-centric approach.

Simply stated, a **Network-on-Chip (NoC)** is a communication network that is used on a single chip. A Network-on-Chip consists of a number of interconnected heterogeneous devices (e.g. general or special purpose processors, embedded memories, application specific components, mixed-signal I/O cores) where communication is achieved by sending packets over a scalable interconnection network. No global wiring is used by a NoC. Wiring resources are shared by the communicating devices. The

idea appeared in the 90s but it started to be researched only from year 2000. Some of the first papers introducing the NoC concept are [17], [18], [19], [20], [21], [22] and [23].

The Network-on-Chip research field is relatively new and of high importance. In HiPEAC's vision [24], nowadays *communication defines performance*. Communication is essential at three levels: (1) between a processor and its memory, (2) between a multicore's different processors and (3) between processing systems and input/output devices. At the processor – memory level, the impact of communication on performance is basically controlled through cache hierarchies. At the other two levels, it is the role of the interconnection network to deal with the communication cost so that performance is less affected. More precisely, more and more processors are integrated on the same chip. Since *power defines performance*, multicores are now the solution for achieving higher performance. In this context, traditional buses, which allow the processors to communicate, no longer suffice. Networks-on-Chip provide the scalability that buses lack. Therefore, NoCs will have an increasing importance in the following years. The growing interest in this area of research is stressed out in HiPEAC's vision [24]: interconnects is one of the clusters on which HiPEAC's roadmap is built.

A component-based hardware design methodology is envisioned in the future [24]. This means that systems will be built from standard reusable components like memories, cores and interconnection networks. This design technique applies however at multiple levels. The level of abstraction increases progressively. Basic blocks (gates, registers, ALUs etc.) make components (processors, NoCs etc.). Components are then used to create different kinds of chips (CPUs, GPUs and so on), which in turn are used to obtain systems that also are interconnected, leading to systems of systems.

Obviously, the importance of interconnection networks increases as the number of communicating components raises. For intra-chip communication, the NoC is the solution and this is due to at least one factor: scalability. As the number of cores increases, the impact of memory bandwidth and memory latency becomes more and more stringent. Networks-on-Chip help at controlling the problems of memory bandwidth and latency. However, *NoCs have a lot of issues that need solving*. For example, they still require a lot of power and occupy large areas of the chip.

More precisely, research in the field of interconnection networks is required by all of HiPEAC's current research objectives: Design Space Exploration (DSE), concurrent programming models and auto-parallelization, design of optimized components, self-adaptive systems and virtualization.

Performing *Design Space Exploration (DSE)* for entire systems is currently a challenge. Unified DSE frameworks, that include the interconnection networks, are estimated to be available only between years 2016 and 2020 [24]. HiPEAC Consortium also estimates that the design space of interconnects will be feasible for exploration only around the year 2015. Only then, network traffic models, benchmarks and realistic performance/power models will be available for on-chip interconnection networks.

Developing *concurrent programming models* requires network interface mechanisms which efficiently support the cache coherence protocols and the communication between processors.

Electronic Design Automation (EDA) refers to a set of methods and tools that help at improving the system's design efficiency. EDA includes (among others) hardware/software modeling and *partitioning and mapping applications* to Multi-

Processor System-on-Chip (MPSoC) architectures (related to this is the mapping problem for Network-on-Chip architectures). EDA has several challenges related to interconnection networks:

- full system simulation, including the interconnections;
- designing application-specific networks;
- designing reusable interconnection modules through interface standards.

Creating interconnection network architectures which reduce power, latency and integration area is a challenge of *designing optimized components*. The interconnection network may also be optimized by using dynamic power management techniques. Another goal is to design on-chip memory hierarchies.

A challenge of *self-adapting systems* is to design *fault tolerant network* architectures and protocols. The network traffic may also be monitored and controlled. Such data may be used by the run time system for self-adaptation.

Network interconnection is important for *virtualization* as well, from the point of view of system security and quality of service. The network may be physically or logically partitioned. A research challenge is to identify how network topologies and routing algorithms can help at system partitioning and isolation.

3 Network-on-Chip Application Mapping

The Network-on-Chip research field deals with fifteen major problems [3]. We will focus next only on one of them, namely Network-on-Chip application mapping.

We begin by defining the Network-on-Chip application problem and by showing that it is an NP-hard problem. Then we show this problem is directly connected to other two NoC research problems: scheduling and routing.

We then propose a taxonomy for Network-on-Chip application mapping algorithms and we describe some of the state of the art algorithms for NoC mapping.

3.1 The Network-on-Chip Application Mapping Problem

The design flow of a Network-on-Chip architecture for a specific application implies the following three major steps [25]:

1. dividing the application into a graph of concurrent tasks (threads);
2. assigning and scheduling the application tasks to the available IP cores;
3. mapping each IP to a NoC tile, so that the metrics of interest are optimized.

The Network-on-Chip *application mapping problem* was formulated in [25] as the *topological placement of the IPs onto the on-chip tiles*. It is an instance of the *quadratic assignment problem*, which is proven to be an NP-hard problem [26]. The search space increases factorially with the system size. For example, a NoC with 8x8 tiles theoretically allows 64! mappings. Theoretically, mapping N IP cores onto M network nodes ($N \le M$) implies $\frac{M!}{(M-N)!}$ possible core arrangements on the NoC nodes. When the number of IP cores is identical to the number of network nodes ($N = M$), the number of possible mappings becomes $M!$. This is therefore a permutation, combinatorial, problem. It directly affects NoC's performance in terms of latency, throughput, power consumption, energy etc. This is because typical network metrics like latency and power are directly proportional to distance.

A typical mapping cost function [27] is:

$Cost(\pi \in P) = \sum_{l \in L} BW_l = \sum_{1 \le i,j \le N} [bw_{i->j} \cdot Dist(i,j)]$, where π is a particular mapping from P, the set of all possible mappings. L is the set of NoC links which are used by the application. BW_l is the bandwidth delivered over link l. $Dist(i, j)$ is the distance between nodes i and j (hop count) and $bw_{i->j}$ is the bandwidth required by node i for communicating its data to node j.

Consider for example the following two mappings π_1 and π_2. They consist of six processing elements placed onto a 2D mesh NoC. PE2 communicates 30 bits/s to PE6 and PE4 100 bits/s to PE3. We are interested to evaluate the two mappings using the above cost function.

Fig. 1 Example of two mappings π1 and π2

For the first mapping, we have:

$$Cost(\pi_1) = bw(PE_2 \rightarrow PE_6) \cdot Dist(PE_2 \rightarrow PE_6) + bw(PE_4 \rightarrow PE_3) \cdot Dist(PE_4 \rightarrow PE_3)$$

$$Cost(\pi_1) = 30 \cdot Dist(PE_2 \rightarrow PE_6) + 100 \cdot Dist(PE_4 \rightarrow PE_3)$$

$$Cost(\pi_1) = 30 \cdot 2 + 100 \cdot 3 = 360$$

Similarly, for the second mapping we get: $Cost(\pi_2) = 30 \cdot 2 + 100 \cdot 2 = 260$. Notice that the only difference between the two mappings is the placement of PE4 and PE5. In the second mapping, PE4 is closer to PE3. Because of this, given the above conditions, mapping π_2 is better than mapping π_1.

In the field of embedded systems, an application is typically described through a **Communication Task Graph (CTG)**. A CTG is defined in [28] as a directed *acyclic* graph, $G' = G'(T, D)$, where each vertex, $t_i \in T$, is a an application task (a computational module in the application). A task typically has assigned to it information like: *execution time* on every type of Processing Element (PE) available for the NoC, *energy consumption* (when assigned to a certain PE), *task deadline* (the time until the task associated with the CTG node must complete its execution [29]), etc. A directed arc between t_i and t_j, is noted as $d_{i,j} \in D$ and has a value associated to it, which represents the communication volume ($v(d_{i,j})$, usually expressed in bits) exchanged between tasks t_i and t_j. Each arc shows both data and control dependencies. A *data dependency* marks that there is a communication between the two tasks (t_i and t_j) [30]. A *control dependency* indicates that a task cannot be executed before its predecessor tasks are not completely executed [30]. Thus, a data dependency is basically an undirected arc between two tasks. When such an arc is present between two tasks, it means that the two tasks are communicating. When the arc is directed, the arc's arrow shows a control dependency between the two tasks.

Note that a CTG is defined as an acyclic directed graph. However, in reality, the tasks of an application may exhibit a communication pattern which creates loops. Loops are not usually modeled with a CTG because of real-time considerations. For hard real-time applications, unbounded loops are avoided because they do not allow bounds on graph execution times. It is not possible to guarantee that the worst-case communication volume path can be executed under the specified deadline. It is preferred that deadlines can be assigned to tasks and a CTG typically has a period attached to it. The CTG can therefore be reiterated after a certain amount of time [31].

The Directed Acyclic Graph (DAG) model of a parallel program is used in [32] to address the scheduling problem. In our humble opinion, the Network-on-Chip research

community adopted the DAG model, from the scheduling research area, with the name of Communication Task Graph.

A task is defined in [32] as a set of instructions that are executed sequentially, on the same processor, without preemption. The task is a node in the DAG. It may have a weight attached to it, which represents the computational cost. However, a CTG does not weight the nodes because it is only communication oriented.

The DAG arcs model the communication messages and the precedence constraints between tasks. The arcs are weighted with communication costs. If two communicating tasks are assigned to the same processor, their communication cost will be neglected. The precedence constraints are what make the graph to be directed. They show how communication flows among tasks. A node is not allowed to start its execution until it receives all the messages from its parent nodes.

Program loops cannot be explicitly represented using the DAG model. Conditional branches are not included as well. According to [32], including loops and branches in the DAG model is an implicitly difficult problem. Additionally, many numerical applications (e.g.: Fast Fourier Transform) contain loops with a number of iterations known at compile time. For such programs, techniques like loop unrolling [6] can be applied. This way, one or more loop iterations can form a task. Also, large classes of numerical applications and data-flow programs have very few conditional branches.

Scheduling a DAG with probabilistic branches and loops was addressed in [33]. Each graph arc has a probability that the child node will be executed immediately after the parent node. Scheduling DAGs with conditional branches is made in [34] by using, beside the precedence graph, a branch graph, too. Although DAG models that deal with loops and/or conditional branches have been proposed, the Network-on-Chip research community adopted the simple DAG model, without loops and conditional branches. Therefore, a CTG does not model program loops nor branches. It focuses on the communications among the tasks of data-flow programs.

The acyclic property of a Communication Task Graph is dropped at a coarser level, denoted by an **Application Characterization Graph (APCG)**. An APCG models an application at the level of Intellectual Property (IP) cores and it is defined in [28] as: a directed graph, $G = G(C, A)$, where each vertex $c_i \in C$ represents an IP core and each directed arc, $a_{i,j} \in A$, characterizes the communication between cores c_i and c_j. This may be application specific information like communication volume. It can also be design constraints, like communication bandwidth, area of IP cores, etc. As in the case of a CTG, a directed arc of an APCG shows data and control dependencies. But, compared to a CTG, an APCG allows cycles. For example, we can have a bidirectional communication between two cores. Note that loops are still not desired in APCGs because of real-time constraints. It is often preferred to transform a directed graph into a Directed Acyclic Graph (DAG) [35]. This allows worst-case execution time analysis, which makes the APCG usable in hard real-time systems as well.

An Application Characterization Graph is obtained from a Communication Task Graph by scheduling the tasks on available IP cores.

Having the definitions for a CTG and an APCG, we can now illustrate the application mapping problem for NoCs using the following figure.

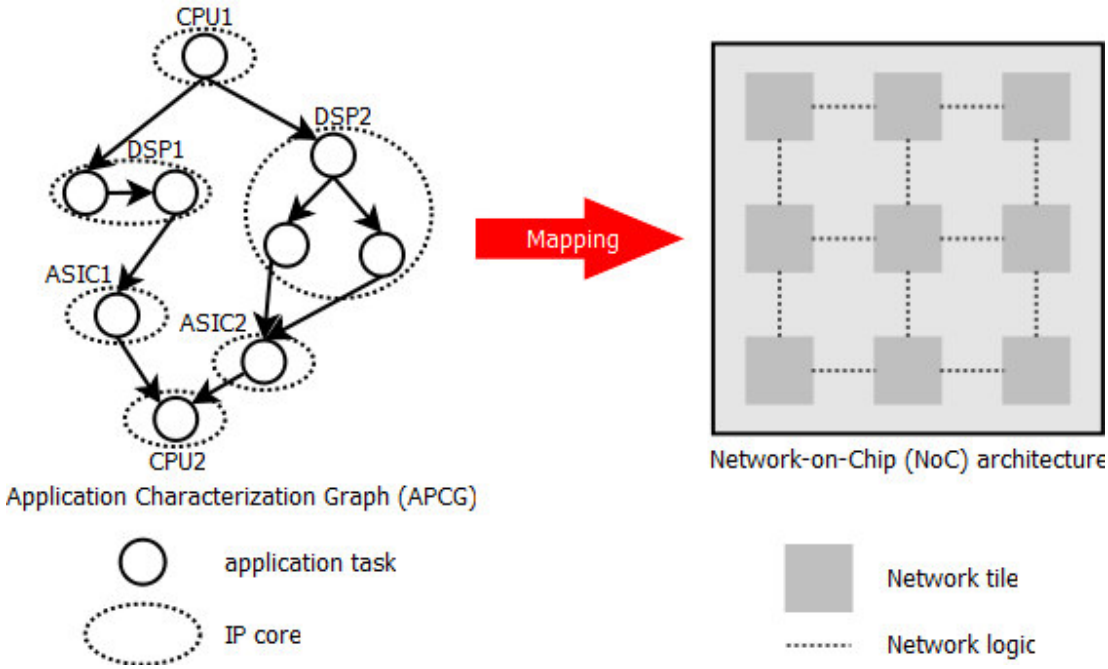

Fig. 2 The Network-on-Chip application mapping problem

Obviously, the NP-hard problem cannot be solved by means of exhaustive search. Heuristic algorithms [36] are employed with the purpose of finding the best topological placement of cores onto network nodes. The objective is to optimize network latency, its energy consumption, etc. Multiple objectives may be followed at the same time, too.

We show next that Network-on-Chip application mapping interacts directly with other two NoC research problems: routing and application scheduling.

3.1.1 Application Mapping and Routing Problems

While a good mapping of cores onto network nodes can lead to energy savings, the routes used by the cores to communicate can have a great impact on the NoC's performance. The best topological placement of cores onto nodes is not enough to account for the performance of the network. The next figure shows an example where two minimal routes are available between the top-left and bottom-right tiles of a 2D mesh NoC. Choosing the proper route can increase the performance of the network.

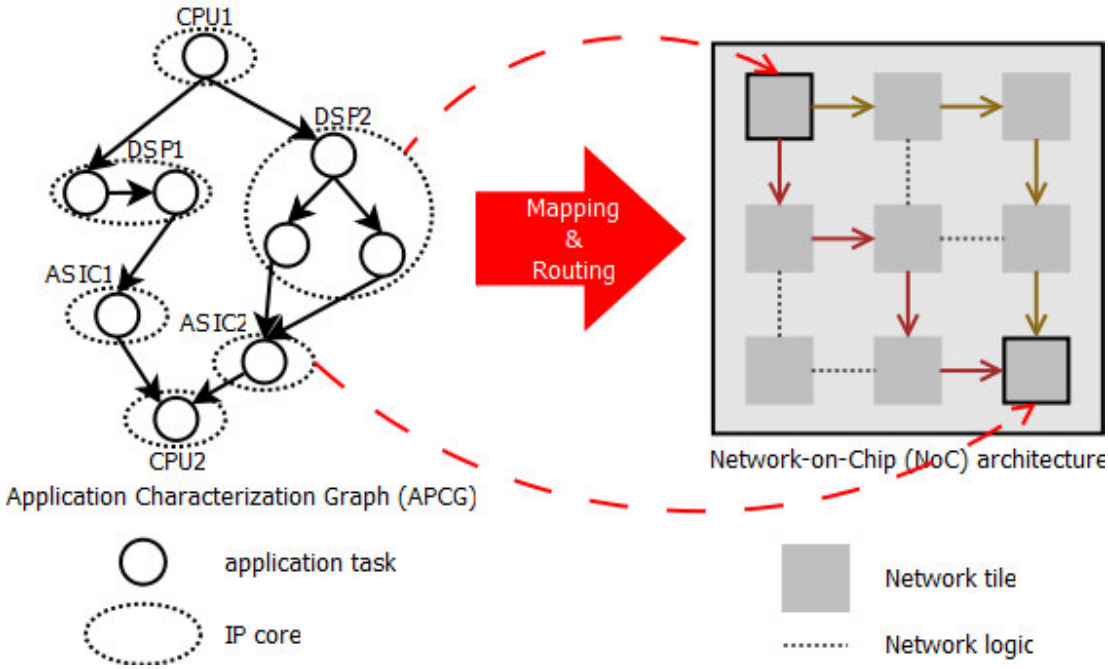

Fig. 3 The application mapping and routing problems

This shows that the **application mapping problem is tightly connected to the routing problem**. Usually it is not necessarily to generate routing paths when placing IP cores onto NoC tiles. A mapping algorithm may simply consider that the NoC architecture is using a particular routing protocol (like XY routing in [25]). However, routing information can help at obtaining a better mapping [37].

3.1.2 Application Mapping and Scheduling Problems

Before mapping the IP cores onto the Network-on-Chip tiles, the application's tasks and communication transactions must be assigned to the NoC resources. Additionally, the tasks' execution order must be established. This is called the **scheduling problem** for NoC architectures [38] and is an NP-hard problem as well. It has a considerable influence on the energy consumed by the IP cores when computing, due to their heterogeneity. For example, a DSP core may consume less energy than a general purpose processor when computing a Fast Fourier Transform. Also, the communication energy consumption of the NoC architecture is affected by the task assignment (because of the routing paths).

Therefore, the application mapping problem is connected to the scheduling and routing problems. The following figure illustrates this fact.

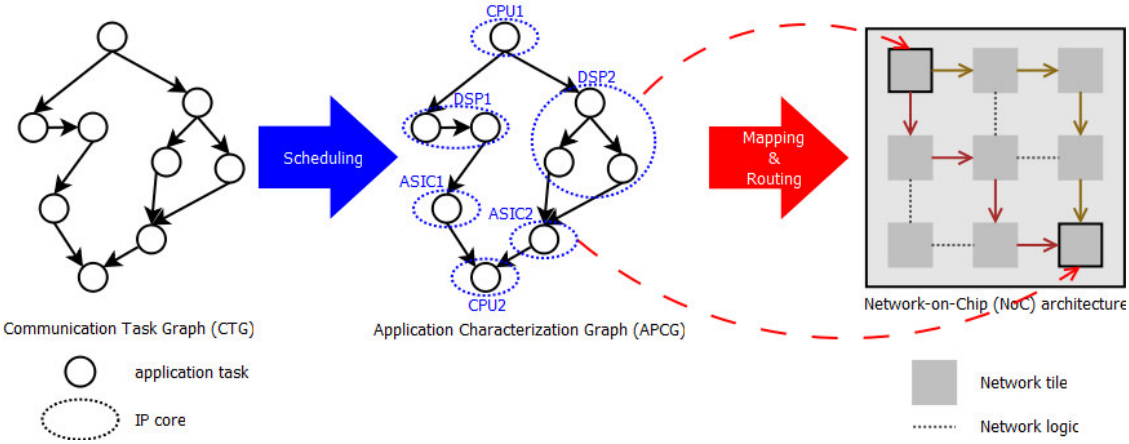

Fig. 4 The scheduling, application mapping and routing problems

An application is described through its Communication Task Graph. A scheduling algorithm is then used to assign application tasks (threads) to available IP cores and to specify their order of execution. After the scheduling step, the Application Characterization Graph is obtained. Then, using a mapping algorithm (which may generate the routing function as well), the IP cores are topologically placed onto the NoC tiles.

We observe that both scheduling and mapping algorithms for Networks-on-Chip have similar objectives. Increasing the performance and decreasing the energy consumption of a NoC, for a particular application, are two optimizations typically made by such algorithms.

Ideally, both scheduling and mapping problems should be treated together. In other words "scheduling" means mapping the application's tasks onto the available IP cores, and "mapping" means mapping the IP cores onto the available NoC nodes. Therefore, both scheduling and mapping problems deal with application mapping onto a Network-on-Chip.

Nevertheless, because of the NP-hard complexity of the problem, mapping applications onto NoCs is divided in a two-step process: scheduling, followed by mapping.

3.2 Taxonomy for the Application Mapping Algorithms

An application mapping algorithm takes into consideration the characteristics of the application, and it has the purpose of finding the best placement of IP cores, onto the tiles of the Network-on-Chip architecture. Obviously, the application mapping algorithm must be aware of the NoC topology. The placement of the cores onto the network nodes can be made before the application starts to be executed and it cannot be changed afterwards. We call this **type of mapping** a *static* mapping. Obviously, the mapping process is iterative: multiple mappings are generated until the optimum mapping is found but, in case of static mapping, all the mappings are obtained before the application starts running. If the mapping of cores changes while the application runs, we have a *dynamic* mapping. This is typical for NoCs that are fault tolerant or application-adaptive. This kind of mapping could also lead to an increase of network performance and/or to a decrease in power consumption but, it is more difficult to implement (than static mapping is).

The factorial number of possible mappings can be decreased because it is very likely that not every mapping is feasible. This is because of the communication demands of the application and the hardware limitations of the underlying Network-on-Chip architecture. For example, consider that we have two communicating IP cores which require a bandwidth of *B* bytes/s. The NoC architecture may have some links that support such high bandwidth and other links that do not support it. In such a case, mapping the two IP cores so that they would require communicating over links that do not support the required bandwidth would generate an impractical mapping. The bandwidth requirement is an example of a **mapping constraint**. We define the mapping constraint (*MC*) as a restriction, derived from the requirements of the application and the characteristics of the Network-on-Chip architecture, imposed when associating IP cores to network nodes. Any mapping constraint may limit the size of the search space. An application mapping algorithm may or may not use one or more mapping constraints but, usually this should be an obvious thing to do because it would speed up the mapping algorithm. The difficulty of using a mapping constraint consists of having the means to evaluate if a mapping satisfies or not that constraint.

The application mapping algorithm explores the search tree of possible mappings and tries to find the best mapping (for a certain application and NoC architecture). In order to determine the best mapping, at least one **optimization goal** is required. Example of optimization goals can be: network performance, communication energy, power consumption, etc. Thus, a mapping algorithm may search for the best mappings by considering a *single objective* or even *multiple objectives*.

As we showed in Section 3.1.1, the mapping problem is also closely related to the routing problem. Any routing algorithm may be applied after the mapping has been done. However, if the mapping algorithm is not **routing aware**, it is possible that the best mapping does not actually provide the best network performance due to the fact that the routing paths were not considered when applying the optimization goals to the possible mappings. A mapping algorithm can thus, deal with identifying the routing paths for the mapped IP cores as well. The routing function can be *deterministic* or *adaptive*. Also, it should provide freedom from deadlock and livelock, and it may have other characteristics like being minimal.

To summarize, we have established that we have two types of application mapping algorithms: static and dynamic. Any mapping algorithm, whether static or dynamic has at least one optimization goal (single-objective or multi-objective). It may use (one or more) mapping constraints. Also, it may determine the routing function, during mapping. The routing can be deterministic or adaptive and it can have other properties like freedom form deadlock and others. We have thus four classification criteria:

mapping type	static	**optimization goals**	single objective
	dynamic		multiple objective
mapping constraints	with one or more mapping constraints	**routing awareness**	generates routes while mapping
	without any mapping constraint		does not generate routes

An application mapping algorithm can be static or dynamic. Either static or dynamic, the mapping algorithm can have a single objective (SO) or multiple objectives (MO) to optimize. Characteristics like using mapping constraints and being routing aware (RA) are optional and can be applied to any type of mapping algorithm (making it thus more specific).

Finally, we note that in [39], where the scheduling problem is also considered, the algorithms are classified as integrated or separated based on whether they treat NoC mapping and scheduling together or not. We consider this to be good classification criteria when including application scheduling, too. The algorithms presented in the above cited paper are for NoCs and for bus-based multiprocessor embedded systems. The NoC algorithms are classified only by whether they have routing awareness or not. The algorithms for bus-based systems are classified according to their optimization goal (energy minimization, handling soft real time constraints or memory awareness). Issues like mapping type and mapping constraints are also mentioned but they are not used as classification criteria. The single/multi objective (optimization goals) criterion is not included. Therefore, we consider our proposed taxonomy to be in accordance with the one from [39] but, more general and suitable.

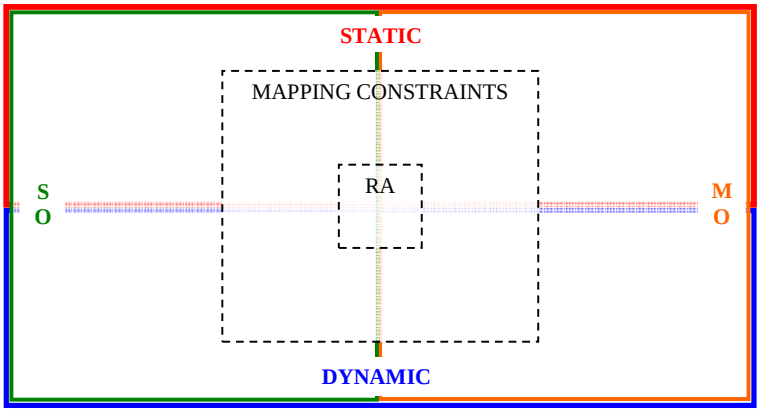

Fig. 5 Taxonomy for application mapping algorithms

4 Designing a Unified Framework for the Evaluation and Optimization of NoC Application Mapping Algorithms

The NoC application mapping problem is addressed by the research community through application mapping (heuristic) algorithms. As we have already shown in Chapter 3, these algorithms consider the characteristics of both the application and NoC architecture. However, currently, the existing application mapping algorithms are basically evaluated only on 2D mesh topologies. But, they can be extended, to work with other network topologies, too. These algorithms are evaluated only on some specific NoC designs and also, their performance cannot be directly compared because a common evaluation methodology is missing.

We propose a unified framework for the evaluation and optimization of Network-on-Chip application mapping algorithms, called **UniMap**. Such a framework will allow a better comparison of their performance. The framework will also be flexible so that many NoC designs (e.g.: different network topologies) can be used for testing the performance of the mapping algorithms. An overview of UniMap was published in [40], [41]. Our framework is an open source project available under GPL v3 license for the research community [42].

We have successfully used UniMap on our **High Performance Computing (HPC) System** [43] **from "Lucian Blaga" University of Sibiu, Romania**. Our HPC currently has 30 Intel Xeon E5405 homogenous quad cores (15 blades, 120 cores), operating at a frequency of 2 GHz. This means a total of **120 Intel cores**. This HPC system also includes 4 **IBM Cell** Broadband Engine (Cell BE) processors (2 blades, 36 cores). The IBM Cell is a heterogeneous multicore, consisting of a 64-bit dual thread PowerPC (master) core plus 8 SIMD processors. These (slave) vectorial processors, called SPU (Synergistic Processor Unit), are specialized for data intensive processing domains like cryptography, media and scientific applications. The HPC allocates 4.84 GB of DRAM memory for each two Intel quad cores and 7.85 GB of DRAM memory for each two IBM Cell cores. This means a total of **88.3 GB** of **DRAM** memory. The total storage capacity is approximately **1.2 TB**. We also performed simulations with UniMap on the HPC system from **Politehnica University of Bucharest, Romania**. UniMap is written in Java (except the NoC simulator, which is written in C++) which makes it highly portable and feasible to be further improved with concurrent programming characteristics.

4.1 The Unified Framework Design

UniMap is composed of the following major modules:

- a model for representing real applications;
- a module for assigning the application tasks to IP cores (Scheduller);
- a module that contains application mapping algorithms (Mapper);
- a model for representing different Network-on-Chip architectures;
- a Network-on-Chip simulator.

This design reflects the interaction between the Network-on-Chip application mapping problem and the other two problems with which it interacts (routing and scheduling – see

sections 3.1.1 and, respectively, 3.1.2). The modules are as decoupled as possible. This approach allows UniMap to be flexible, reusable (and modular).

We use eXtensible Markup Language (XML) schemas to describe real applications and Network-on-Chip architectures. The Scheduler, Mapper and NoC simulator modules do not interact directly. They communicate through XML models. This approach theoretically allows any NoC simulator to be used with UniMap. Similarly, any scheduling or mapping algorithm can be integrated as easy as possible.

The following figure illustrates these components and presents the design flow of the unified framework.

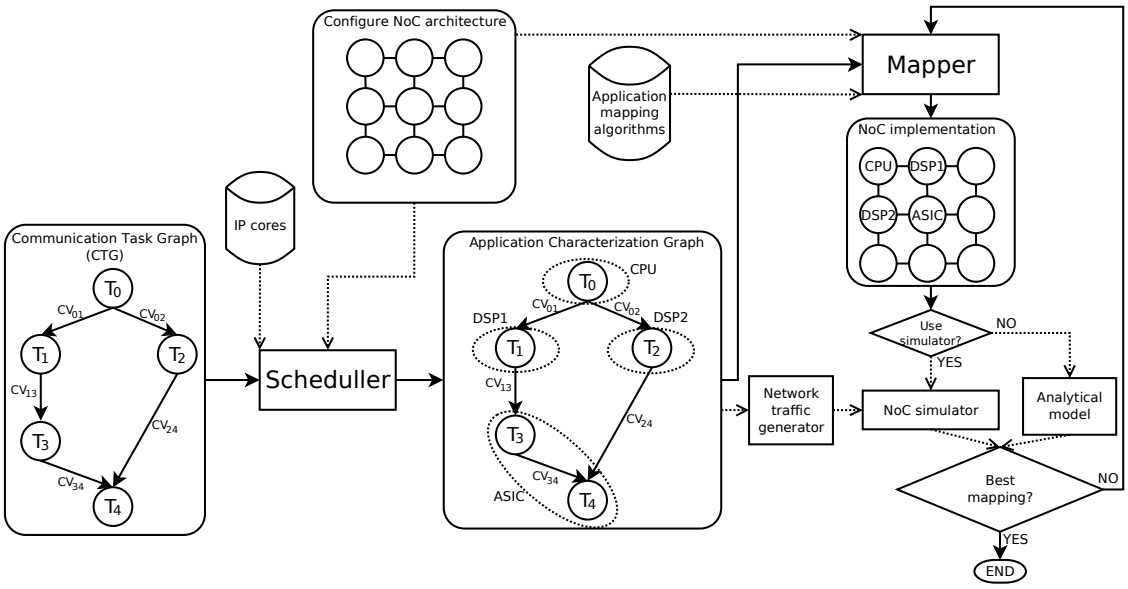

Fig. 6 UniMap design flow

An application running on a NoC architecture is described through its Communication Task Graph (CTG). The CTG presents the application partitioned into tasks (concurrent threads). It shows the communication pattern of the application: which tasks are communicating with which tasks and the communication volume of the data exchanged between tasks (e.g.: CV_{01} denotes the communication volume from task T_0 to task T_1).

We propose obtaining CTGs in three distinct ways:

1. randomly, by using the TGFF [31] tool;
2. from realistic embedded applications, using the E3S benchmarks suite [44];
3. from real-world multithreaded applications, using the CETA [35] tool.

The tasks must be first assigned to the IP cores. This can be done using a scheduling algorithm. For example the EAS algorithm [38] is able to perform scheduling under real-time restrictions, while trying to optimize the energy consumption of the NoC architecture.

The IP cores library from E3S was integrated in UniMap. For each IP core, information like task execution time and power consumption for a given task is known.

The output of the scheduling algorithm is the Application Characterization Graph (APCG). The APCG is the input for the mapping algorithm.

A main component of the framework will consist in a library containing (state of the art) application mapping algorithms' implementations. The performance of every mapping algorithm can be evaluated on multiple NoC designs, through our developed simulator.

The NoC simulator is another important part of the unified framework. An important aspect of the simulator consists in its flexibility. This will impact on the number of possible ways in which the simulated NoC can be configured. The simulator is also responsible with determining the network's performance represented through multiple objectives (performance, energy consumption, etc.). This allows a thorough comparison of the mapping algorithms, in a unified manner. For each selected network design (e.g.: the network topology can be varied), an application mapping algorithm will provide multiple mappings, until the best mapping is determined. The NoC simulator includes a network traffic generator which emulates the communicational behavior of the application (based on CTG and APCG graphs).

4.2 The Developed Network-on-Chip Simulator

ns-3 NoC is a Network-on-Chip simulator that the author of this Thesis started to develop during his five months of PhD external research stage at Augsburg University (Germany), Department of Systems and Networking, led by Professor Theo Ungerer. We decided to develop our own NoC simulator because the current tools for this (new) research field are still immature. According to HiPEAC's vision [24], mature NoC simulators are expected only in 2015.

The simulator is based on the ns-3 simulation framework for Internet systems. It is a modular, flexible and scalable NoC simulator. It has parameters like: flit size, packet size, packet injection probability, packet injection rate, buffer size, switching mechanism (Store and Forward, Virtual Cut Through, Wormhole), routing algorithm (Dimension Order Routing and other two protocols that account for the network load). It supports k-ary d-cube topologies (2D mesh, 3D mesh, 2D torus, 3D torus, hypercube etc.). It contains a network traffic generator based on communication patterns form real applications. Also, using ORION 2.0 [45], it can estimate power consumption and integration area. Our ns-3 NoC is an open source project, which we contribute to the Network-on-Chip research area.

4.2.1 Experimental Results

We present next some preliminary simulation results published in [46], were we evaluated the potential of the NoC Irvine architecture and were we showed the impact of the buffers' size on NoC's performance. The following results express the network performance, through the average latency of the packets, as a function of packet injection probability. The synchronous version of the simulator was used. During the simulation, the first 1000 cycles were considered warm-up cycles. Packets were injected into the network for 10000 cycles. Only the packets injected after the warm-up cycles were collected into the statistics. The Irvine architecture was used, with XY routing, wormhole switching, input channel buffers of 9 flits in size and packets of 8 flits in length. The effects of speeding up the data flits, like it is done in [47], are shown it the following charts.

Fig. 7 shows how, on a 8x8 Irvine NoC, the average packet latency decreases as data flits are sent through the network using a clock frequency which is two or four times higher than the one used for advancing the head flits.

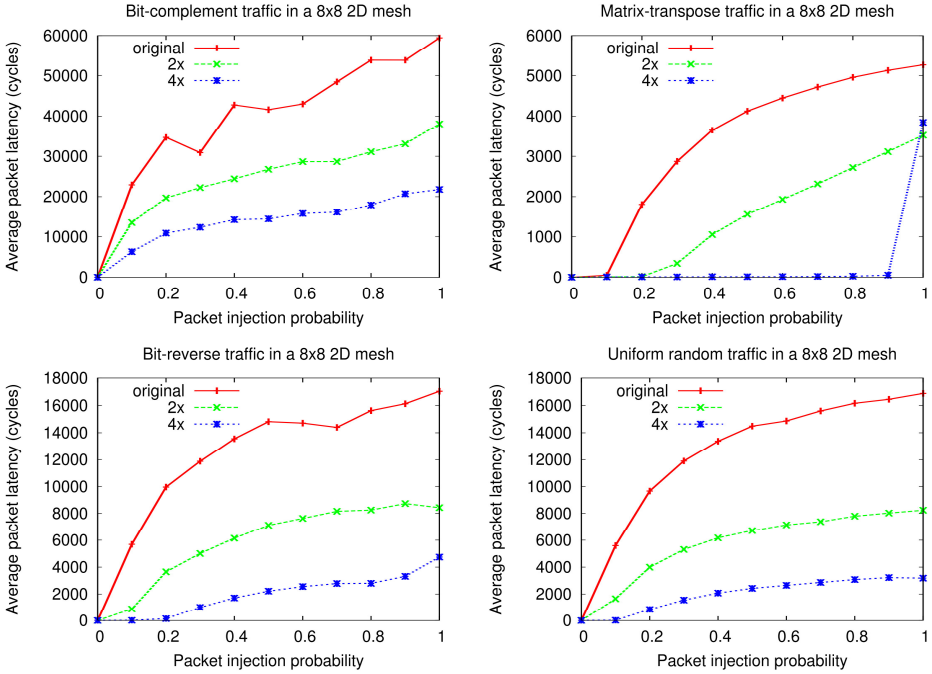

Fig. 7 The average packet latency on a 8x8 Irvine NoC architecture, while the speed with which data flits advance in the network varies for 4 different communication patterns

With the matrix-transpose traffic pattern and using a 4 times higher clock frequency for the data flits, the packet's average latency remains close to the zero-load latency, as long as the injection probability is lower or equal than 0.9. The Irvine architecture helps at decreasing the network congestion. This is also visible for the other three traffic patterns. The network is significantly less congested when data flits are transmitted faster than head flits. For the bit-complement traffic pattern, the average packet latency is fairly higher because each node injects packets. This is not true with the other traffic patterns because they can create traffic from a certain node to exactly the same node, which is not injected into the network. Therefore, we believe that this behavior might contribute to the bit-complement's higher packet latency.

We did similar simulations on a 4x4 Irvine NoC, too. The simulations on an 8x8 Irvine NoC took approximately 10 times more time than the simulations done on the 4x4 network. The longest simulation on a 4x4 network took around 2 minutes and a half (this is approximately 10 times faster than NoCSim [48]).

In [49] we showed how the NoC performance varies on topologies like: 2D mesh, 2D torus, 3D mesh, 3D torus and hypercube. For example, the following figure shows the buffer size influence on the performance of a (2x2x2x2) hypercube. Unless specified otherwise, the simulator's parameters have the same values as before.

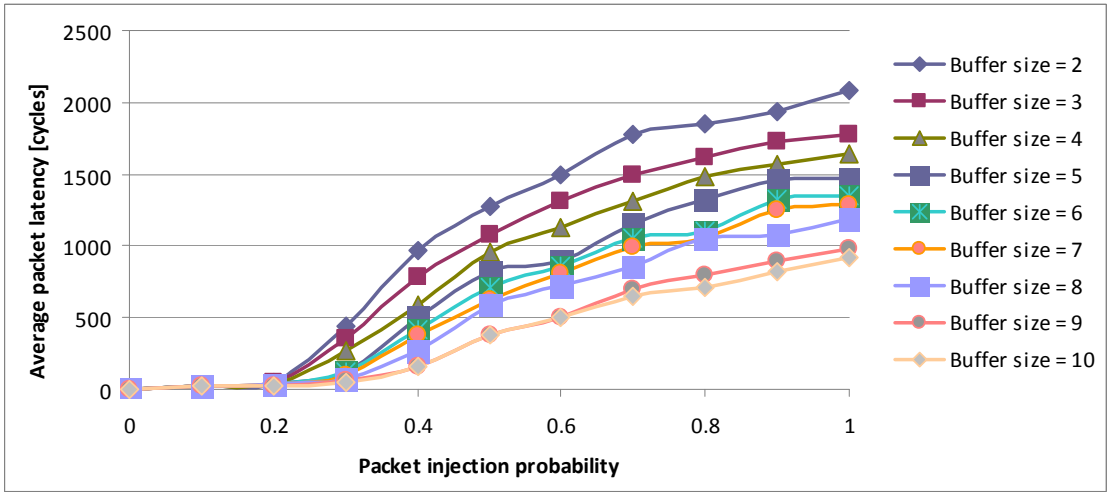

Fig. 8 The average packet latency on a hypercube NoC architecture, while the size of the input buffers varies uniformly

The Network-on-Chip's performance improves as the buffer size increases.

We show next how the NoC's average packet latency decreases as we increase the node degree by switching from a 2D mesh to a 3D mesh and then to a hypercube. The simulations were made using the uniform random traffic pattern.

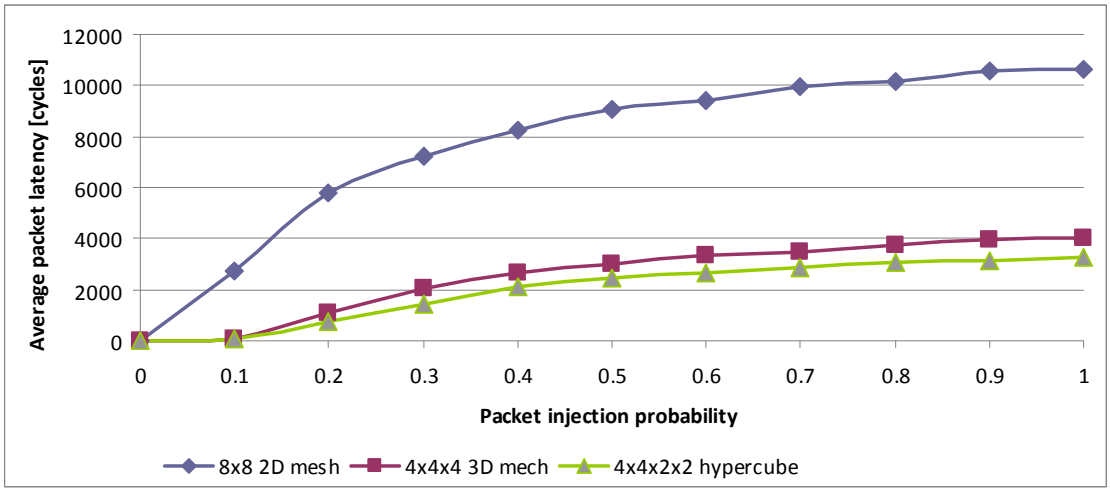

Fig. 9 Average packet latency on 64 node mesh NoCs, with 2, 3 and respectively 4 dimensions

We observe a significant increase in the NoC's performance when using a 3D mesh. The performance increases even further when placing the 64 nodes in a hypercube topology. We observed the same behavior torus topologies.

5 Benchmarks

In the previous chapter we presented our developed unified framework for the evaluation and optimization of Network-on-Chip application mapping algorithm. UniMap uses as input traffic patterns for real applications, described through directed graphs. As stated in [30] Network-on-Chip benchmarking is still an open problem. The Open Core Protocol International Partnership (OCP-IP) is currently working to model real applications for NoC benchmarking [50].

This chapter presents the benchmarks used in this PhD thesis, for studying the Network-on-Chip application mapping problem. All benchmarks describe real applications, designed for Systems-on-Chip (SoCs). These applications are modeled using Communication Task Graphs. We gathered some of the most used CTGs and APCGs by the NoC research community and integrated them in UniMap, through a common XML representation. The communication graphs are taken from the Embedded Systems Synthesis Benchmark Suite (E3S) [44] and from some of the most cited papers from the field of Networks-on-Chip. We also make our contribution to Network-on-Chip benchmarking, by proposing two new Communication Task Graphs for a H.264 video decoder.

In this PhD thesis abstract we present only the first Communication Task Graph for the H.264 video decoder.

CTG 0 presents a H.264 decoding system that uses data partitioning: the video stream is equality divided onto more CPUs, each one of them running a H.264 decoder.

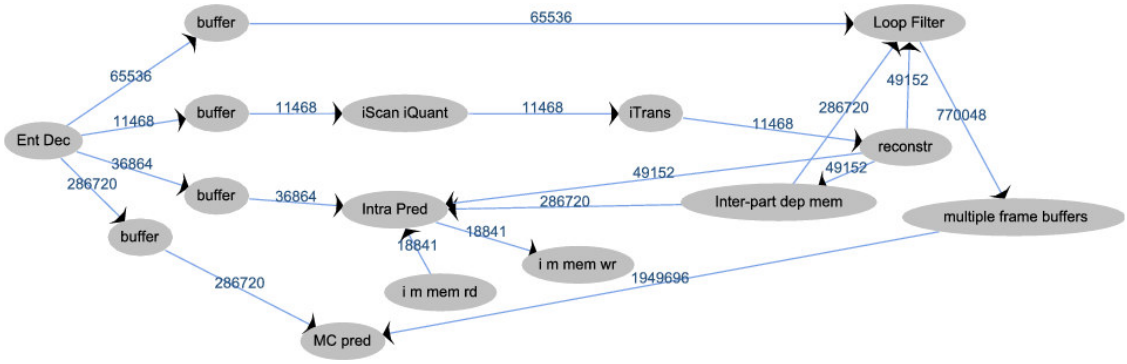

Fig. 10 H.264 CTG 0 (period: 0.0009765625 seconds)

With the functional partitioning approach, the messages between the decoder tasks are communicated. With data partitioning, data dependencies among data partitions are communicated. It is shown in [51] that, with data partitioning, a significant bandwidth reduction is obtained.

From H.264 CTG 0 we created APCG 0, with 14 cores, by grouping the two tasks for accessing the intra mode memory (*i m mem rd* and *i m mem wr*).

6 Optimized Simulated Annealing for Network-on-Chip Application Mapping. A Domain-Knowledge Approach

Simulated Annealing (SA) [52] is one of the first heuristic algorithms used to address the Network-on-Chip application mapping problem.

The advantages of Simulated Annealing are given by its ease of implementation, its applicability to many combinatorial optimization problems and the ability to give reasonably good solutions [53].

However, the parameters of the algorithm must be carefully chosen, since SA can easily run for a very long time until it gives a suitable solution. Because Simulated Annealing is a very general algorithm, several choices must be made in order to implement it for a particular problem.

This chapter presents a domain-knowledge approach to Network-on-Chip application mapping problem. We describe an Optimized Simulated Annealing (OSA) [54] algorithm that we designed for the topological placement of cores onto NoC nodes. OSA uses an application- and network-based exploration of the search space. Using knowledge about communication demands, the IP cores are clustered implicitly and dynamically. We compare OSA with the above mentioned simulated annealing technique and with a branch and bound algorithm, too. We focus on algorithm speed, memory consumption and solution quality.

6.1 The Algorithm

OSA was created by continuing the work of Hu and Marculescu. Their Simulated Annealing and Branch and Bound algorithms are available through the NoCmap project [55]. We have ported their two algorithms, written in C++, into UniMap (written in Java). OSA also uses some of the best practices for Simulated Annealing applied for assigning tasks to processors [56]. We justify our approach by the fact that NoC application mapping problem is closely related to the NoC scheduling problem [28].

We present next the Optimized Simulated Annealing pseudocode, which is derived from the general Simulated Annealing from [56].

Require: $M_i \neq 0$
Ensure: $T_0 \geq 1$
  $M \leftarrow M_i$
  $C \leftarrow BitEnergyCost(M_i)$
  $M_{best} = M$
  $C_{best} = C$
  $T_f = 0.001$
  $R = 0$
  for $i = 0$ *to* ∞ **do**
    if $i \% L = 0$ **then**
      $R = 0$
    end if
    $T = T_0 \cdot 0.9^{\lfloor \frac{i}{L} \rfloor}$
    $M_{new} = PDFbasedSwapping(M, T)$
    $C_{new} = BitEnergyCost(M_{new})$
    $\Delta C = C_{new} - C$
    if $\Delta C < 0$ *or* $NormInvExpAccept(\Delta C, T)$ **then**
      if $C_{new} < C_{best}$ **then**
        $M_{best} = M_{new}$
        $C_{best} = C_{new}$
        $R = 0$
      else
        $R = R + 1$
      end if
      $M = M_{new}$
      $C = C_{new}$
    else
      $R = R + 1$
    end if
    if $T \leq T_f$ *and* $R = L$ **then**
      break
    end if
  end for
  return M_{best}

Fig. 11 Optimized Simulated Annealing

OSA starts from an initial mapping, M_i, which is randomly generated. Another input parameter can be the initial temperature, T_0, set to 1 by default. The mapping's cost is obtained using the bit energy model from [25]. We use a standard geometric annealing schedule, with

$$L_{OSA} = {}_nC_2 - {}_{n-c}C_2 = \frac{n(n-1)}{2} - \frac{(n-c-1)(n-c)}{2} = \frac{c(2n-c-1)}{2} \quad ,c,n \in N, n \geq c$$

annealing iterations per temperature level. This number corresponds to how many mappings may be obtained from the current mapping, by moving one core. SA has $L_{SA} = 100n^2$ (we noted the number of NoC nodes with n). It is obvious that $L_{OSA} < L_{SA}$. Also, in terms of algorithm complexity, we note that $\left.\begin{matrix} O(L_{OSA}) = O(n^2) \\ O(L_{SA}) = 100O(n^2) \end{matrix}\right| \Rightarrow 1 - \frac{O(L_{OSA})}{O(L_{SA})} = 99\%$.
This speedup is in perfect concordance with our further experimental results.

While other Simulated Annealing approaches (for NoC application mapping) select the core to be swapped randomly, OSA does not use a uniformly random probability when determining the core to be moved. Instead, it adapts the variable grain

single move (based on probability densities and used for task mapping [56]) into a variable grain swapping move, which uses two Probability Density Functions (PDFs). OSA builds a Probability Density Function (PDF) for each core, based on the amount of data it communicates. This leads to better chances for selecting a core that communicates more data than a core which communicates less data. As the annealing temperature decreases, the probabilities uniformly equalize. Therefore, at low temperatures, all cores have an equal chance to get selected for swapping. Through this approach, OSA uses problem knowledge (dynamic characteristics) to explore the search space. The following function is used:

$$P[SelectedCore = i] = \frac{1}{c} + \frac{T}{T_0}\left(\frac{coreToComm_i}{totalToComm} - \frac{1}{c}\right), \text{ where:}$$

- c is the number of cores to be mapped;
- T *and* T_0 are the current and initial temperatures;
- *totalToComm* is the total amount of data communicated by the all cores;
- $coreToComm_i$ is amount of data communicated by core i.

The second core used for swapping is selected by accounting for the communication volumes between the core to be swapped and the rest of the cores. Another Probability Density Function is built for each core. It is similar with the one above but, it does not consider only the data communicated by the core but, also the data received by the core. Also, this second PDF is not temperature dependent. Each core gets such a PDF associated before the annealing starts. This PDF is defined as $P[c_i \leftrightarrow c_j] = \frac{comm_{ij}}{totalComm}$, where:

- $comm_{ij}$ is the communication volume between core i and j (this value is positive if core i sends data to core j, or core j sends data to core i; otherwise, it is zero);
- *totalComm* is the communication volume of the entire application.

According to the PDF described above, the second core is selected for swapping. Then, OSA searches, in a uniformly random way, for a direct neighbor of the second selected core. This one will be swapped with the first selected core. This approach tries to make communicating cores to attract each other, to cluster themselves, in a natural manner. OSA's move function performs an implicit clustering of the communicating cores, using a stochastic approach.

Compared to Cluster-based Simulated Annealing (CSA) [57], our algorithm clusters the cores dynamically, during the annealing phase. OSA does not work with predetermined clusters, and it also does not cluster the NoC nodes. Network-on-Chip node clustering is not needed because OSA looks in the NoC node's neighborhood.

We call this kind of move a *PDF-based swapping move*. At every temperature level, OSA performs exactly L_{OSA} PDF-based swappings.

We use the normalized inverse exponential acceptance function because this is the one recommended by [56]. OSA stops when the final temperature (T_f = 0.001) is reached and the number consecutive rejected moves, R, reaches L. This corresponds to the coupled temperature and rejection threshold stopping condition proposed in [56]. While in [56] R counts how many moves were rejected since the last accepted move, in OSA we use R to count how many moves were rejected, per temperature level, since the last current best mapping was found. This means that while OSA requires no best mapping to be found during an entire temperature level, the general Simulated Annealing from [56]

needs to wait until the number of unaccepted moves, counted from the last one accepted, reaches *L*. OSA's stopping condition is therefore more coupled to T_f than to *R*. This makes OSA's number of iterations to be independent of the NoC topology and its size. Since we consider that the energy variations are small enough when the final temperature is reached, we believe our way of computing *R* is more suitable for a Simulated Annealing applied to NoC application mapping.

Currently, OSA works only with 2D mesh topologies but, it can be adapted to work with other NoC topologies, too. Like Hu and Marculescu's SA, OSA is also capable to generate the routing functions, in a deadlock- and livelock-free manner, and to check if the obtained mapping meets the bandwidth constraints.

Compared to the general SA, OSA determines how many iterations to make per temperature level by considering the mappings' neighborhood size. Using Probability Density Functions, OSA performs an implicit and dynamic core clustering (CSA's clustering is explicit and static).

6.2 Experimental Results

In this section, we evaluate our Optimized Simulated Annealing by comparing it with Simulated Annealing and Branch and Bound. The evaluation is three folded. We account for execution runtime, memory consumption and solution quality. We show next only the most representative results. More detailed results are available in [58].

We begin with a runtime comparison between OSA and SA and respectively OSA and BB. The speedups represent an average of the 1000 runtime speedups obtained for each benchmark.

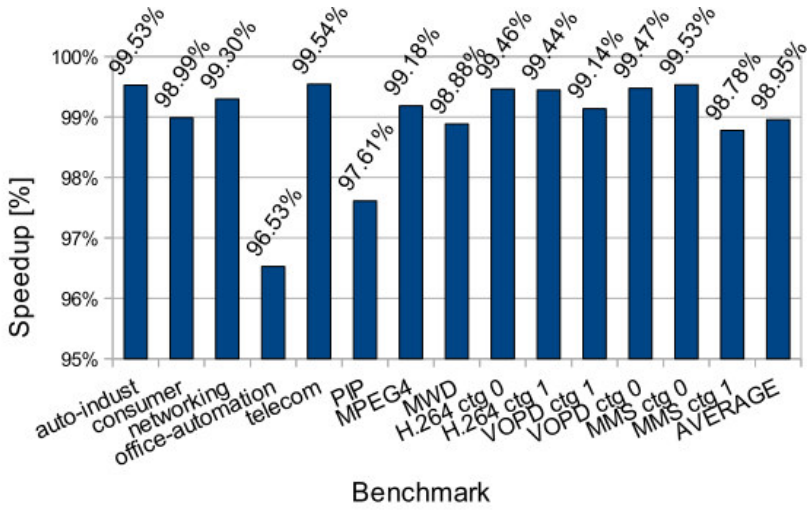

Fig. 12 OSA speedup over SA

The chart above clearly shows OSA is much faster than Hu and Marculescu's Simulated Annealing. We have obtained a 98.95% speedup on average. This is in perfect concordance with our theoretical speedup expectations. The "lowest" speedups are on *office-automation* and PIP, the benchmarks with the smallest number of IP cores. We justify this significant speed gain mainly by the way OSA computes the number of iterations per temperature level. This number takes into consideration the NoC size, the

number of cores to be mapped, and it is much lower than the number used by Hu and Marculescu.

The following chart shows how fast OSA is compared to Branch and Bound.

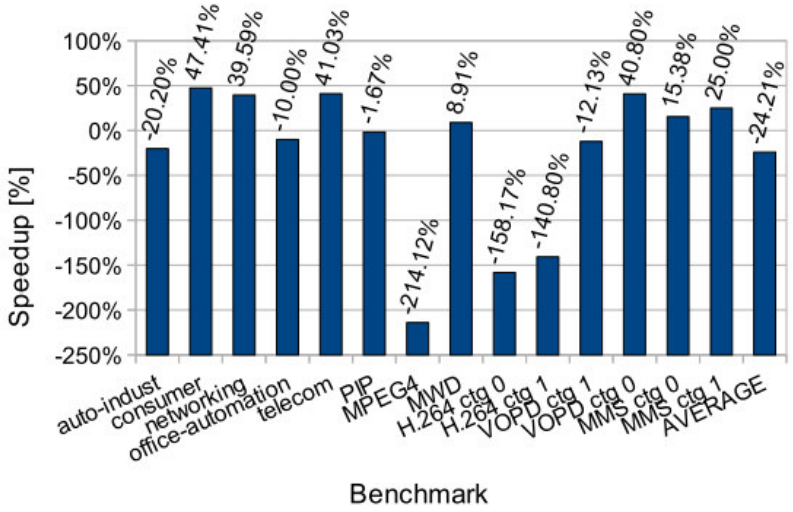

Fig. 13 OSA speedup over BB

It can be seen that OSA is slower than BB by ~ 24%, on average. However, for half of the benchmarks, OSA is faster. Compared to Branch and Bound, our algorithm obtained poor runtimes on MPEG4 (more than twice slower), H.264 (~ 1.5 times slower in both cases) and slower but similar runtimes for PIP, *office-automation*, VOPD (CTG 1) and *auto-indust*. We also observe OSA was faster on the biggest benchmarks: 25% speedup for MMS (with 25 cores) and ~ 41% speedup for *telecom* (30 cores).

Next we show how OSA's memory consumption is, compared to the memory consumed by Simulated Annealing and Branch and Bound.

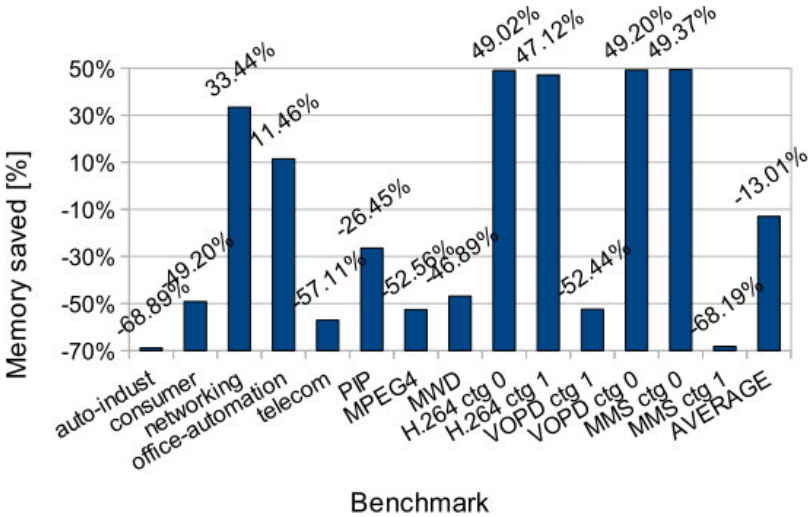

Fig. 14 OSA compared to SA in terms of heap memory consumption (a positive value means OSA consumes less memory)

Simulated Annealing consumes less memory than OSA when mapping the benchmarks with more than 16 cores. OSA manages to beat SA on several benchmarks with 16 cores but, on average, Simulated Annealing consumes with ~13% less memory than our Optimized Simulated Annealing.

However, compared to Branch and Bound, OSA takes a little bit less memory on average. This is shown in the next chart.

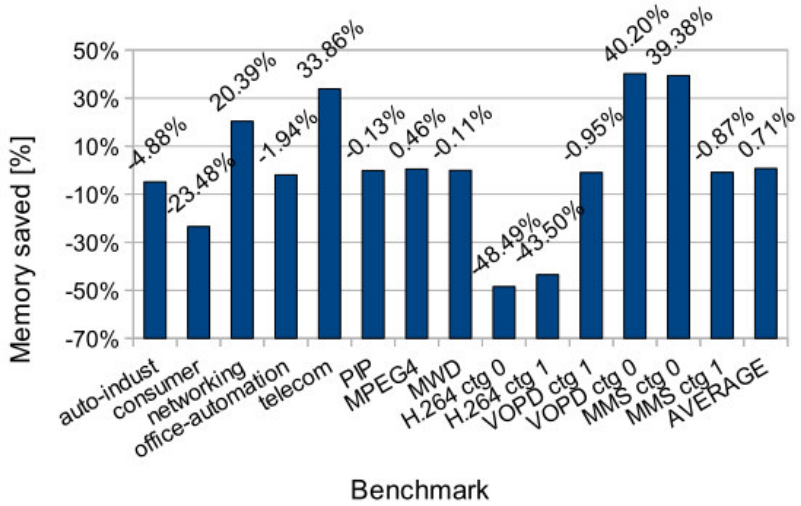

Fig. 15 OSA compared to BB in terms of heap memory consumption (a positive value means OSA consumes less memory)

Actually, this chart points out the tendency of Branch and Bound to grow its memory requirements as the problem size gets higher: OSA consumes with more than 33% less heap memory than BB, on *telecom*.

Now we present the quality of the solutions found by the three algorithms. We are interested in solutions with the smallest cost possible because the cost function we used estimates the energy consumed by the Network-on-Chip.

The following chart compares the mappings found by SA and OSA. For each benchmark, we evaluate the 1000 mappings returned by the two algorithms and count how many times one algorithm retuned mappings better (marked with "<" in the chart's legend) than the other one. Cases when both algorithms returned mappings with exactly the same cost are marked distinctively.

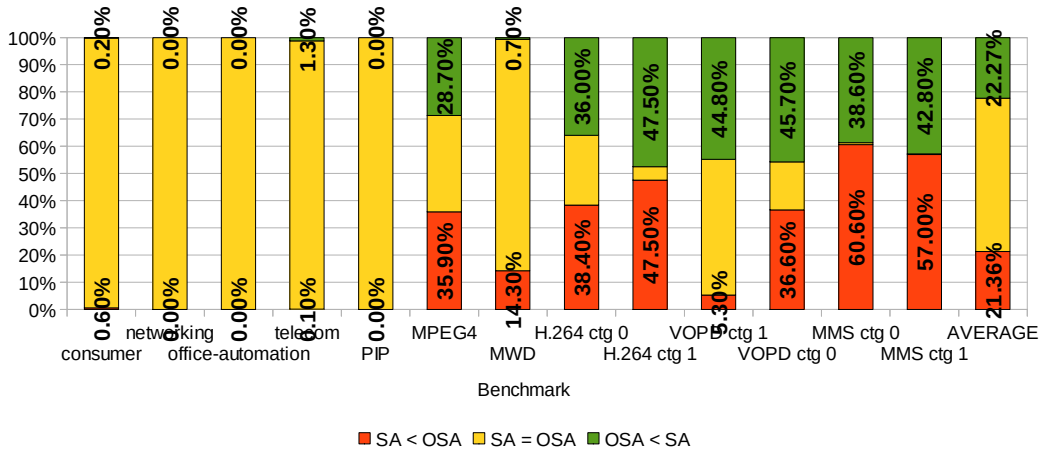

Fig. 16 OSA mapping costs, compared to SA mapping costs

We notice that both algorithms find the same "best solution", after all 1000 runs, for benchmarks: *networking*, *office-automation* and PIP. For the last two of these three benchmarks, we confirm the solution is optimal because we applied an exhaustive search. Overall, OSA finds worse solutions than SA for 6 of the 14 benchmarks used in our simulations: MPEG-4, MWD, H.264 (CTG 0), MMS (CTG 0), MMS (CTG 1) and *consumer*.

We have also found out that SA and OSA always find the same best solution. However, Branch and Bound fails to obtain a mapping that consumes at most like the best mapping found by SA and OSA in two cases: for MMS (CTG 1), the energy lost with BB's mapping would be 0.1 % and for *auto-indust*, the energy loss is ~6%.

We measured the difference between the worst and best mappings found for each benchmark by SA and OSA. With our Optimized Simulated Annealing, the variation between the worst and best mappings was not higher than 8%. However, with SA we obtained the highest variation to be 70% for MMS (CTG 1). For the rest of the benchmarks SA did not varied with more than 6%. Excluding MMS (CTG 1), the SA average variation was 1.51% and the OSA average variation was 2.56%. If we also consider MMS (CTG 1), SA had an average variation of 5.85% while OSA's value was less than half (2.53%). We conclude that the variations between the best and worst mappings are comparable for SA and OSA.

Fig. 17 shows how many times the best solution, given by all three algorithms, was found by each one of them.

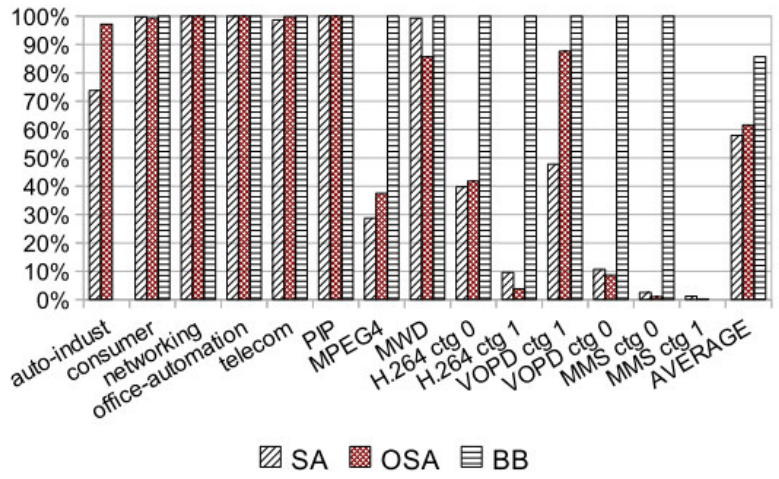

Fig. 17 Best solution percentage

This chart shows that OSA finds the best solution more often than SA for several benchmarks: *auto-indust*, *telecom*, MPEG4, H.264 (CTG 0), VOPD (CTG 1). BB outmatches OSA for the MMS benchmarks, VOPD (CTG 0), H.264 (CTG 1), MWD and *consumer*. Another observation is related to BB: it finds the best solution with probability 1 for all benchmarks, except *auto-indust* and MMS (CTG 1).

We also averaged the quality of the 1000 mappings per benchmark. Branch and Bound is the algorithm that, on average, gives the mapping with the smallest energy consumption. It fails just on *auto-indust* benchmark, where OSA provides the best average mapping cost. Optimized Simulated Annealing achieves for MMS (CTG 1) a far better average cost compared to Simulated Annealing: more than 34% energy gain is obtained with OSA. For the rest of the benchmarks, the differences between OSA and SA are less than one percent. Compared to BB, OSA provides solutions that are worse with no more than 2.5% on each benchmark, except *auto-indust*, where OSA is better with more than 6% than Branch and Bound.

Using 1000 simulations per benchmark, we have previously shown that the percentage of better solutions was lower for OSA than for SA on six benchmarks: MPEG-4, MWD, H.264 (CTG-0), MMS (both CTGs) and *consumer*. We present here our attempt of increasing OSA's quality of solution by increasing the initial temperature. We applied this technique on the benchmarks mentioned above, with the purpose of getting OSA's percentage of better solutions over SA's percentage. Increasing the initial temperature allows OSA evaluate more mappings. Also, the higher the temperature, the bigger is the probability to accept "bad" moves during the annealing process.

Through this technique the quality of solution for our Optimized Simulated Annealing got better, matching SA's quality of solution i.e., OSA's percentage of better mappings overcame the corresponding SA percentage. Still, we had one exception: we were unable to obtain the desired outcome for MMS (CTG 1). We disregard this undesired result due to the fact that in this case, on average, SA consumes with more than 34% more energy than OSA.

Note that we have increased OSA's initial temperature exponentially because, due to the OSA's geometric annealing schedule, an exponential increase of temperature leads to a linear increase of the number of temperature levels.

The following table presents OSA's speedup over SA, in terms of runtime, and the initial temperature required by OSA to beat SA.

Benchmark	Speedup (%)	Initial temperature
MPEG4	97.51	1e10
MWD	96.76	1e10
H.264 (CTG 0)	99.18	1e2
MMS (CTG 0)	97.41	1e17
MMS (CTG 1)	61.40	1e107
consumer	98.91	2

If we ignore MMS (CTG 1), we see that the speedup remained high even with the increase of initial temperature.

In order to illustrate how important OSA's clustering technique is, we present next a comparison between OSA with and without clustering. The single thing that distinguishes OSA without clustering from OSA (with clustering) is that, in the first case, the simple random core swapping is used, without any restrictions.

The following chart shows how frequently the best solution is found.

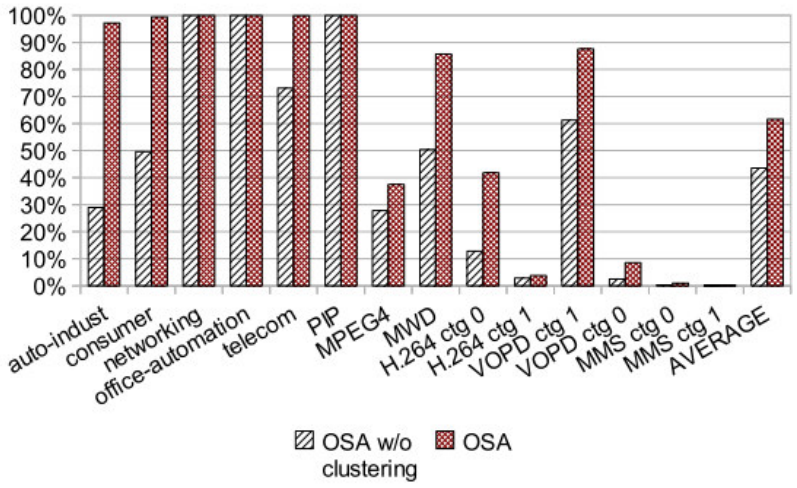

Fig. 18 The influence of OSA's clustering on best solution percentage

For all benchmarks, OSA with clustering finds the best solution more frequently than OSA without clustering. More than this, we observe significant differences for the benchmarks mapped onto the 4x4, 5x5 and 6x5 2D mesh NoCs. It is important to mention that the two OSA variants find the same best solution for all benchmarks, except MMS (CTG 1). In this case, the best solution found by OSA w/o clustering is with 0.02% worse.

The next chart shows how much energy is consumed on average by OSA without clustering (compared with OSA using clustering).

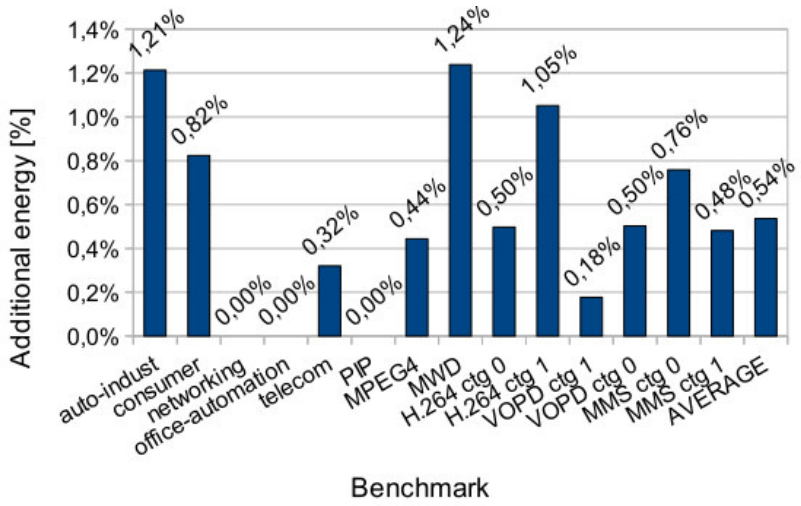

Fig. 19 Average energy consumed by the mappings obtained with OSA without clustering

It may be noticed that OSA without clustering finds mappings that consume additional energy. The clustering technique leads to lower energy consumption with more than 1% in some cases. OSA with clustering always gives better average results than OSA without clustering.

Finally, we present the simulation results on bigger 2D meshes. We used four instances of the VOPD benchmark with 16 cores (like in [57], because applications with a high number of cores are lacking and because we preferred using real applications instead of randomly generating core graphs, like in [25], [37]) and obtained a benchmark with 64 cores. Using SA, OSA and BB, we mapped it on an 8x8 2D mesh. SA was run ten times and OSA and BB run 100 times.

We obtained an average running time of ~ 12.65 hours (per simulation) for SA. OSA ran for approximately 155 seconds, while BB required just ~ 114 seconds. Averaging the results from the 100 runs, OSA was ~36% slower than BB. Still, OSA runtime is significantly lower than CSA's runtime: 4750 seconds [57].

OSA consumes with approximately 39% less memory than Branch and Bound. During the 100 simulations, OSA's peek memory consumption was 37.3 MB, while BB required a maximum memory of 85 MB.

The best mapping was found by Simulated Annealing. However, OSA's best mapping is only ~ 0.7% worse. Branch and Bound finds a best mapping that consumes around 64% more than the best mapping found by SA. Averaging the 100 mappings done by OSA and comparing them with the ones obtained with BB, we have observed that Branch and Bound obtains on average a mapping cost ~70% worse.

We have aggregated all the E3S benchmarks used in our previous simulations and obtained 84 cores that we mapped onto a 10x9 2D mesh. Again, SA run 10 times, while OSA and BB run 100 times.

SA required a very big time to run one simulation: approximately 70 hours. OSA ran for approximately 526 seconds, while Branch and Bound needed only 380 seconds. Averaging the results from the 100 runs, Optimized Simulated Annealing was ~48% slower than BB.

OSA consumed approximately the same of memory Branch and Bound required. During the 100 simulations, OSA's peek memory consumption was 62 MB, while BB

required a maximum memory of 71 MB. We believe Branch and Bound manages to keep the memory consumption not growing exponentially by pruning most of the search space (we observed BB, in several simulations, to prune 85% to 93% of the explored search space).

Averaging the 100 mappings done by OSA and comparing them with the ones obtained with BB, we have observed that Branch and Bound obtains on average a mapping cost ~76% worse. Simulated Annealing found the best solution but, it is better than OSA's best solution by only 0.09%.

Using the H.264 (CTG 1), MMS (CTG 0), MMS (CTG 1), MPEG4, MWD and VOPD (CTG 0) benchmarks, we have obtained 97 cores that we mapped onto a 10x10 NoC. Because of the huge running time SA needed for mapping the previous application, we simulated these application with 97 cores only with OSA and BB (both were run ten times).

Optimized Simulated Annealing run on average approximately 15.9 minutes per simulation. Branch and Bound needed only two thirds of this time: ~15.44 minutes for each mapping simulation (OSA is only 3% slower than BB).

Branch and Bound consumed around 40 MB of memory and Optimized Simulated Annealing required approximately 45 MB.

Once more, OSA found every time mappings better than the ones found by Branch and Bound. Averaging the 100 mappings done by OSA and comparing them with the ones obtained with BB, we have observed that Branch and Bound obtains on average a mapping cost ~76% worse.

By combining all non E3S benchmarks (PIP, H.264, MPEG4, VOPD, MWD, MMS), we get a benchmark with 131 cores, which we mapped onto a 12x11 Network-on-Chip. OSA and BB mapped this benchmark ten times.

OSA required, on average, approximately 51 minutes mapping this application. Branch and Bound was ~15% faster: it needed only around 44 minutes, on average.

In this case, OSA consumed less memory, 36 MB, while BB memory requirements were 14% higher.

Optimized Simulated Annealing found each time a mapping that consumes significantly less memory. On average, OSA's solutions need 79.4% less memory than BB's solutions.

Finally, we combined all of our benchmarks an obtained an application with 215 cores. We used OSA and BB to map it (ten times) onto a 15x15 NoC.

Optimized Simulated Annealing run for 8.4 hours, on average. OSA consumed on average 265 MB of memory, for each mapping.

Branch and Bound run on average 3.77 hours for each mapping. This is more than half OSA's runtime. Memory consumption was also significantly lower: only 158 MB. However, we obtained no solution from BB, after all ten mapping. All mapping attempts will Branch and Bound failed. No suitable solution was found because, each time, the algorithm pruned more than 98.7% of the search space. This severe pruning did not allow BB to finish mapping the application. This leaves us to believe that Branch and Bound's memory consumption does not grow exponentially but, the quality of solution is heavily affected, up to the point where the algorithm does not give any solution.

7 Designing Domain-Knowledge Evolutionary Algorithms for Network-on-Chip Application Mapping

Evolutionary Computing (EC) [59] is a part of Artificial Intelligence (AI) inspired from the evolution process encountered in Biology. The heuristic algorithms from this field of research address NP-hard optimization problems by means of natural selection and evolution mechanisms. The search space is filled with candidate solutions, called individuals.

Evolutionary Algorithms (EAs) are used in many research fields to address single-objective and multi-objective optimization problems, based on the concept of Pareto efficiency [60].

In this chapter, we use UniMap (see Chapter 4) to evaluate and optimize two evolutionary algorithms: an Elitist Genetic Algorithm (EGA) and an Elitist Evolutionary Strategy (EES). After approaching our problem with an Optimized Simulated Annealing technique, we decided to switch to evolutionary algorithms due to their intrinsic parallelism. Evolutionary techniques perform searches starting (in parallel) from multiple points in the search space. Our evaluated algorithms optimize the Network-on-Chip communication energy. We consider multiple crossover and mutation operators, specific for permutation problems, like NoC application mapping is. Using problem specific knowledge, we propose such context-aware operators. We show such operators improve the evolutionary algorithms' performance. We try to find out which crossover and mutation leads to the best solutions. We also research whether crossover or mutation helps more the evolutionary algorithms. These algorithms are compared with our Optimized Simulated Annealing (OSA) technique (see Chapter 6). Finally we approach our problem in a multi-objective way: besides minimizing NoC communication energy, we also try to obtain a mapping that is thermally balanced.

The work presented in this chapter was submitted (on July 21st, 2011) to the Journal of Systems Architecture (JSA - http://ees.elsevier.com/jsa). Since July 25th, 2011, it is under review with manuscript number JSA-D-11-00103.

7.1 Energy- and Performance-Aware Genetic Algorithm

We developed in UniMap an Energy- and performance-aware Genetic Algorithm (EGA). EGA is based on the Generational Genetic Algorithm (GGA) [61]. As compared to GGA, EGA implements an elitist mechanism.

EGA is developed for MxN 2D mesh NoCs but, it may be extended to work with other topologies as well. The algorithm uses a bit-energy analytical model for computing the NoC communication energy. It considers that Dimension Order Routing is employed but, it can also generate a deadlock- and livelock-free routing function using the turn [62] and odd even [63] models. Additionally, network bandwidth constraints may be considered.

7.2 Elitist Evolutionary Strategy

Elitist Evolutionary Strategy (EES) [59] is available in jMetal. We adapted this algorithm to our problem by using the same energy-aware fitness function like in the EGA case.

7.3 Developing Problem Knowledge Crossovers

This section presents the crossover operators used in this research. We work with crossover operators for permutation problems. There are many such operators in literature (order, inversion, cycle etc.) [64]. We used Position Based Crossover and Partially Mapped Crossover. Position Based Crossover (PB) [65] aims keeping absolute position information during the recombination process. Partially Mapped Crossover (PMX) [66] tries to preserve genes' order, adjacency and position as much as possible. PMX is one of the most used crossover operators for permutation problems [59].

Next, we present two new crossover operators that we propose for the Network-on-Chip application mapping problem.

7.3.1 NoC Position Based Crossover (NPB)

NoC Position Based Crossover (NPB) extends PB so that the cores that are kept fixed are not selected randomly. We rather keep fixed the hot spot cores, i.e. the cores which communicate the most data.

Our approach, based on hot spots, is similar to the approach from [67]. The difference is that, we do not simply swap the hot spot core with a randomly chosen core; we rather fix the first half of the most communicating cores. While the crossover from [67] behaves as a swap mutation, NPB acts as a Position Based crossover, with context-awareness.

7.3.2 Mapping Similarity Crossover (MS)

Our developed Mapping Similarity crossover (MS) has the purpose of identifying the topological similarities between two (parent) mappings and replicating them in the offspring. MS has two phases. The first phase tries to identify the mapping similarities between the two mappings. By doing so, the common characteristics of the two mappings are identified. The cores mapped in a similar way in both parents are mapped the same in the two children: child 1 maps the similar cores like parent 1 and child 2, like parent 2. We should point out that the offspring keep the common characteristics of their parents, either good or bad. The goal of the first MS phase is to decide which genes the offspring inherit from their parents. MS attempts to improve the offspring through a secondary phase, which performs a greedy mapping for the rest of the genes. This phase tries to raise the children fitness by rearranging the cores which are not mapped similarly, hoping they will be placed better with respect to the similar cores.

We argue our MS crossover operator does not simply act as a swap mutation operator like in the research of Ascia et al. [64], [68], [67]. MS instead tries to identify mapping similarities, which are inherited from both parents. This emphasizes the crossover character.

7.4 Mutation Operators

This section presents the two mutation operators used in this research. We chose to work with swap mutation, which is a very common genetic operator in permutation problems. It simply interchanges two randomly selected genes.

Using our developed Optimized Simulated Annealing algorithm as a mutation operator we obtain a hybrid algorithm: an Evolutionary Algorithm which incorporates a

Simulated Annealing technique. OSA performs a context-aware mapping and outputs two cores which must be swapped. It performs an iteration each time it gets called by the Evolutionary Algorithm. When the number of iterations reaches OSA's number of iterations per temperature level, the annealing temperature is decreased.

By using OSA as a mutation operator, we propose using hybrid algorithms for NoC application mapping. More precisely, we have a meta-heuristic, with an Evolutionary Algorithm as the main algorithm. The EA encapsulates a NoC specific algorithm, as a mutation operator (OSA). This approach allows us to benefit from the intrinsic parallelism that EAs contain. Also, the exploration has context-awareness, through the proposed mutation. The mutation may be performed by any algorithm for NoC application mapping. Any EA using a mutation operator may be used.

7.5 Multi-objective Optimization

NoC communication energy is minimized by placing the communicating IP cores as close as possible, onto the NoC tiles. Since we are interested to evaluate the performance of the genetic operators used in this research, we do a multi-objective optimization, too. Our second objective is to do a thermal-aware placement of the IP cores. Uniformly distributing the IP cores' temperature across the network leads to the minimization of the hotspot temperature. Two IP cores that consume significant power should be placed at a greater distance from one another. However, this means our thermal balance objective is in contradiction with our energy objective.

7.6 Experimental Results

We present next only the most representative results obtained with our research on domain-knowledge evolutionary algorithms for Network-on-Chip application mapping. Our entire set of results is available in [69].

We start by measuring the mapping cost found for each benchmark. Since (in order to improve the accuracy of our results) we map the same application multiple times, we obtain an average mapping cost (energy). We get such average cost for every evolutionary algorithm, with every crossover operator and for each mutation probability. For EGA with MS crossover and OSA mutation, we also limit the similarity function to the IP cores that are one – EGA-MS-OSA (1) – or two hops away – EGA-MS-OSA (2).

We work with the metric that we call **Normalized Absolute Deviation (NAD) from the minimum** (in this case the minimum average energy). This metric is based on the statistic absolute deviation (*AD*) metric. Because we deal with a minimization problem, we consider the absolute deviation from the minimum average cost from the entire data set (X_b). Then, we normalize *AD* by dividing it to $\max\{X_b\}$ (the maximum average cost from the entire data set). Therefore, the normalized absolute deviation (of data point $x_{b_m} \in X_b$) from the minimum is $NAD_{b_m} = \frac{x_{b_m} - \min\{X_b\}}{\max\{X_b\}}$. The index b_m marks a benchmark evaluated with an algorithm with at a certain mutation probability. For each benchmark, we obtain its NAD, at every mutation rate, using the above formula. The data set X_b contains the average mapping energies obtained by all evolutionary algorithms, for the specified benchmark. At each mutation level, we average the NADs of all our

benchmarks. Therefore, we have $Average\ NAD_m = \frac{NAD_{1_m} + NAD_{2_m} + ... + NAD_{B_m}}{B}$, with B being the number of benchmarks and m the mutation probability ($m \in \{10\%,20\%,...,100\%\}$). We use this metric because directly comparing the average energies obtained for different applications is infeasible since each application has its own energy domain (which usually differs significantly).

Fig. 20 presents how much the average mapping cost deviates from the minimum average cost, found by all algorithms. We show the results obtained only for the big benchmarks (VOPD 4x, *all-mocsyn*, *97-cores*, *131-cores* and *215-cores*) because our evolutionary algorithms perform similarly on the rest of the benchmarks (in terms of average mapping cost). For every algorithm, only the point corresponding to the mutation probability where it performed best is shown.

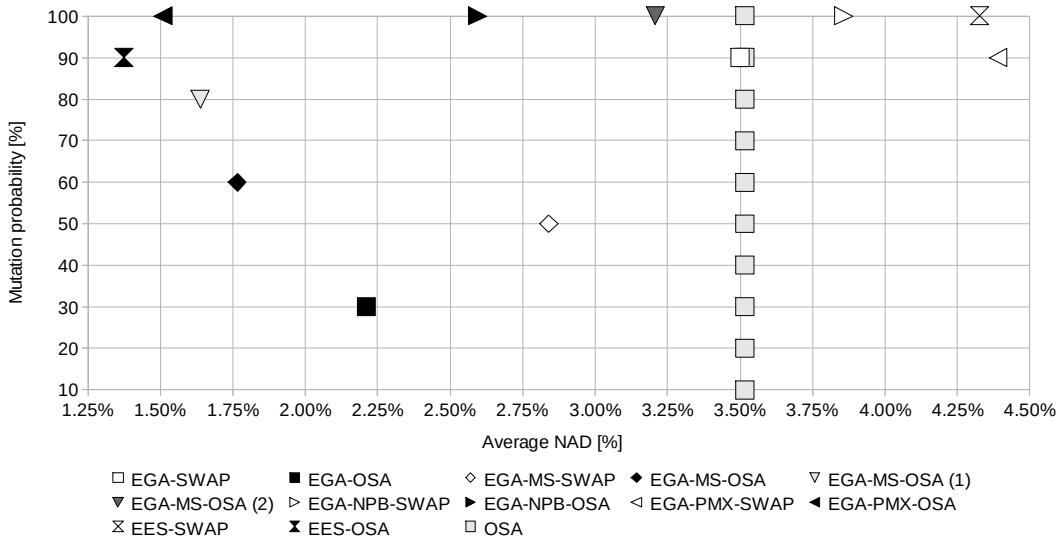

Fig. 20 Algorithms' comparison based on their average normalized absolute deviation, from their common minimum average cost (only big benchmarks)

It may be easily observed that all algorithms perform significantly better with OSA mutation than with swap mutation. EES-OSA has the smallest deviation, among all algorithms, followed by EGA-PMX-OSA. EGA-MS-OSA is the next best performing algorithm in this case. We even notice a slightly better performance for EGA-MS-OSA (1) (1.64% deviation) than for EGA-MS-OSA (1.77% deviation). However, EGA-MS-OSA (2) performs much worse (3.21% deviation, at 100% mutation probability). After EGA-MS-OSA we have EGA-OSA and EGA-NPB-OSA. EGA-OSA is the algorithm that gives the smallest deviation at the lowest mutation rate: 30%. EGA-NPB-OSA is better than EGA-MS-SWAP. Still, EGA-MS-SWAP clearly beats OSA, making MS the only crossover than outmatched OSA with both mutation operators. Mapping Similarity is the only crossover operator that performed well regardless of the mutation operator. PB crossover also does not provide bad results but, MS is clearly better (EGA-SWAP has a 3.52% deviation, with only 0.02% smaller than OSA's deviation). The performance of our other crossover operator, NPB, is not good when we compare it with PB. In both

cases (OSA or swap mutations), NPB performs worse than PB. However, we observed (on all benchmarks) that NPB performed better and better as mutation grew. It performed best at 100% mutation probability (similarly to PMX). PB performed best at 80% mutation (on all benchmarks). Raising the mutation made PB perform worse. EGA with MS performed best at 50% - 60% mutation rate, on all benchmarks. We may conclude that MS is the crossover operator that contributes the most at obtaining a good average mapping cost. The rest of the operators rely significantly more on the mutation operator.

In conclusion, in terms of average mapping cost, the Elitist Evolutionary Strategy with OSA mutation performs the best. The Energy Aware Genetic algorithm has the best behavior with OSA mutation and with PMX crossover. Our developed Mapping Similarity crossover gives similar results: its normalized absolute deviation is with only 0.25% worse than the one of PMX. NPB performs worse on the big benchmarks. Its deviation is with 1% higher that the one of PMX.

Next, we are interested to find how good are the best mappings found by each algorithm. In order to compare the best solutions found by all algorithms, we have identified for each application the best solution found by all algorithms. Then, for each application, with each algorithm and mutation probability, we have computed the additional energy (*AE*) its best mapping consumes, with respect to the best solution found by all algorithms. Using the same notation like for NAD computation, we define the Additional Energy metric as $AE_{b_m} = \frac{x_{b_m} - \min\{X_b^{'}\}}{x_{b_m}}$. In this case however, we work with a different data set. $X_b^{'}$ contains the minimum mapping energies (not the average ones, like in the previous case). Finally, like for average NAD, we averaged the additional energies for all benchmarks. The following chart presents these results. We show for every algorithm the value at the mutation level where it obtained the lowest average additional energy.

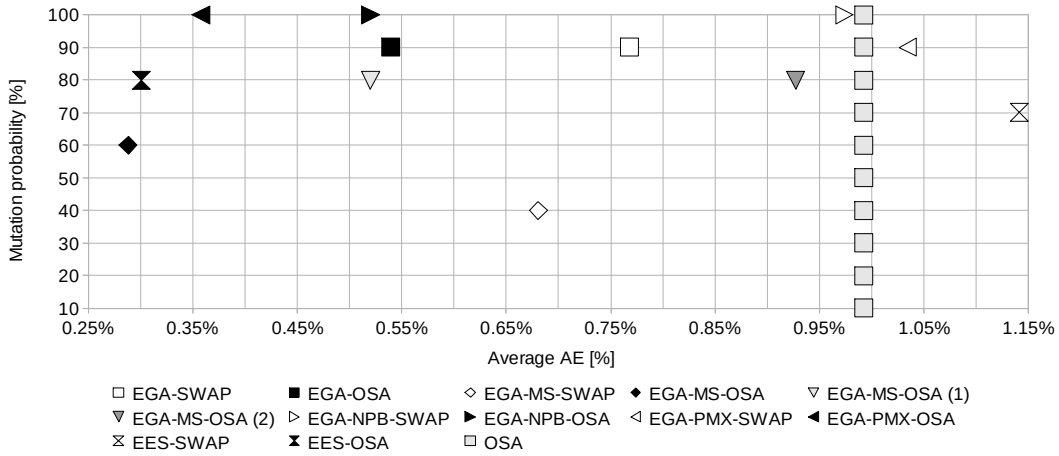

Fig. 21 Average additional energy consumed by the best mappings found by each algorithm, compared to the best mappings found by all algorithms

EGA-MS-OSA is the algorithm that has the most mappings that are the best. On average, the best mappings found with this algorithm introduce just 0.29% additional energy. Very

close to this result is EES-OSA, with 0.3% additional energy. After EGA-PMX-OSA (0.36%), follow EGA-NPB-OSA and EGA-MS-OSA (1), both with 0.52% additional energy. Note that all the algorithms except EES-SWAP and EGA-PMX-SWAP find, on average, better mappings than OSA. This chart also shows that swap mutation produces worse mappings than OSA mutation, regardless the algorithm. However, there is an exception: EGA-MS-OSA (2) does not produce better mappings than all algorithms with swap mutation.

We conclude our solution quality based analysis by showing how often each algorithm manages to reach the best solution. We refer to the best solution found by all algorithms, not to the best solution each algorithm found. Hence, it is possible an algorithm has a zero best solution percentage. We define the Averaged Best Solution percentage at mutation rate m as $Average\,BS_m = \frac{BS_{1_m} + BS_{2_m} + ... + BS_{B_m}}{B}$ [%]. BS_{b_m} is the Best Solution percentage for benchmark b, at mutation level m. It represents how many times an algorithm finds the best mapping, found by all algorithms.

On the big benchmarks, OSA is unable to find the best solution. Also, not all of the evolutionary algorithms manage to reach the best solution. This may be seen in the following figure.

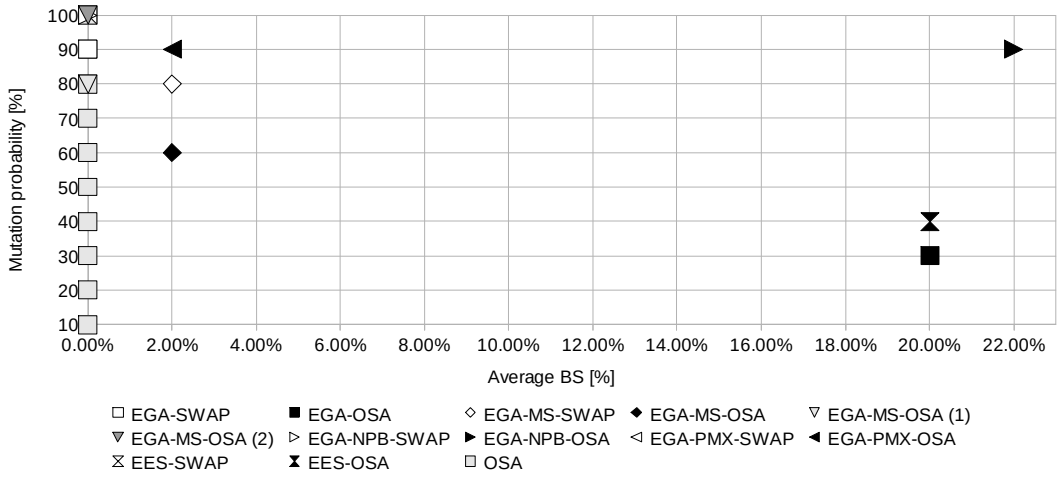

Fig. 22 Average best solution percentage on big benchmarks

EGA-NPB-OSA is the algorithm that has the highest best solution percentage, which is 22% at 90% mutation probability. EES-OSA and EGA-OSA have a value of 20%. Than, with just 2%, we have EGA-PMX-OSA, EGA-MS-OSA and EGA-MS-SWAP. We notice all the algorithms using swap mutation are unable to reach the best solution. The only exception is EGA-MS-SWAP.

Our conclusion is that NPB crossover gives the best solution percentage, on the big benchmarks. Mapping Similarity and PMX crossovers give a similar best solution percentage. OSA mutation is essential for EES because with swap mutation EES performs worse even than EGA with NPB crossover and swap mutation.

Regarding the optimal mutation probability, we observed there are algorithms, like EGA-NPB-SWAP, for which we obtained exactly the same mutation rate. However, in general there is no ideal mutation probability. Our experiments indicate the optimal mutation probability may vary from 20% up to 100%. For EGA-MS-OSA, we got the same optimal mutation probability in terms of average and best mapping cost. We conclude that mutation probability is application and algorithm dependent. The lowest mutation rate is consistently encountered when working with our developed Mapping Similarity crossover. This indicates MS is the crossover operator that relies the least on mutation to find the best NoC application mapping. We tried to limit the similarity function of MS by considering only the cores which are one or two hops apart in the NoC. Overall, we did not obtain significantly better results. EGA-MS-OSA (1) and EGA-MS-OSA (2) require a higher mutation probability to function optimally.

We present next how some of our algorithms converge in time. Since the previous results showed us that OSA mutation gives better results than swap mutation, we focus only on these algorithms: EGA-OSA, EGA-PMX-OSA, EGA-NPB-OSA, EGA-MS-OSA and EES-OSA. We ran each of the five algorithms for 1000 generations per application. To improve the accuracy of our simulations, we ran each application for 100 times (by setting the random number generator seed from 1 to 100). Finally, we averaged the energy cost of all 100 mappings per application and per generation. We worked with the mutation values determined by our average cost analysis.

Fig. 23 shows how the five algorithms converge on our biggest benchmark. We mention that for all the other benchmarks we obtained the same behavior, as we will detail next.

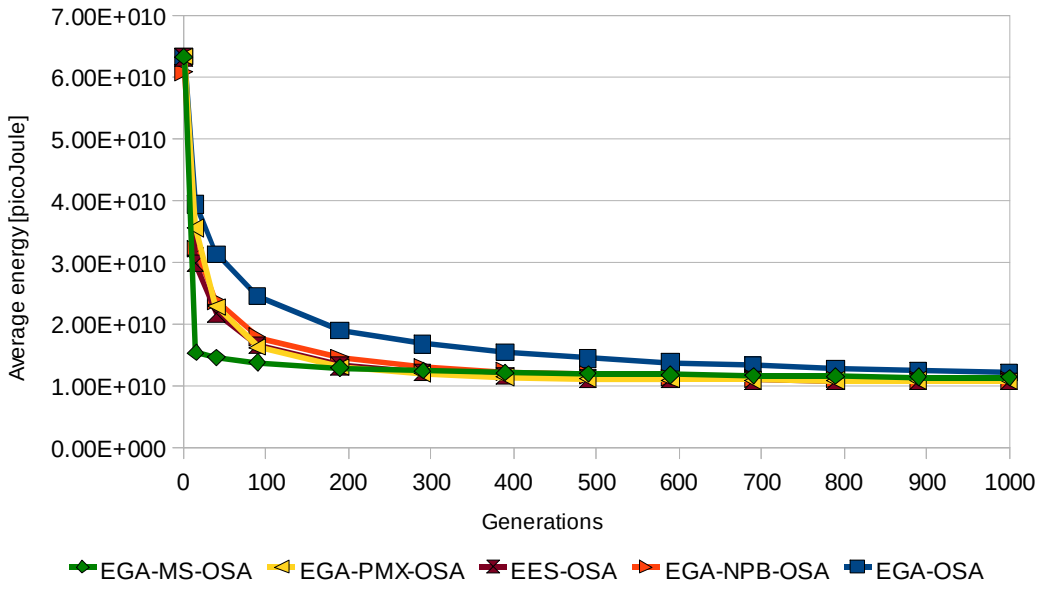

Fig. 23 Algorithms' convergence for 215-cores benchmark

All algorithms manage to reduce the mapping energy significantly, within the first 100 generations. EGA-OSA has the lowest convergence speed. EGA-PMX-OSA, EGA-NPB-OSA and EES-OSA behave approximately the same. EGA-MS-OSA is the algorithm that

converges the fastest during the first generations. After that, its convergence speed decreases and it is outrun by EGA-PMX-OSA, EGA-NPB-OSA and EES-OSA. We believe this is justified by the greedy approach from the second phase of our Mapping Similarity crossover.

We measured when each algorithm reaches its best solution during the 1000 generations, for each benchmark and we averaged the results. EGA-OSA converges in 732 generations. It requires the most number of generations to obtain its mapping with the best communication energy. EGA-MS-OSA converges in 562 generations. Algorithms EGA-PMX-OSA, EGA-NPB-OSA and EES-OSA require 475 generations. EES-OSA is the algorithm that, on average, has the fastest convergence speed (424 generations).

Finally, we switch from a single objective to multi-objective Network-on-Chip application mapping. Besides minimizing communication energy, we are now also interested in obtaining a thermal balanced NoC design. Using NSGA-II and SPEA2 genetic algorithms implemented in the jMetal library, augmented with all our genetic operators, we evaluated NoC mappings for *all-mocsyn*. This is the benchmark that contains all E3S applications. For E3S we know how much power the IP cores consume to execute a particular task. Each algorithm ran once, with each crossover – mutation combination, for 1000 generations. Each time we started from the same initial population. We used the optimal mutation probabilities determined by our average mapping cost analysis.

Fig. 24 shows, for every algorithm, the (normalized) hypervolume [60] obtained at each generation. For a minimization problem (like ours), the hypervolume is defined as the volume enclosed by the Pareto front and a reference point. The coordinates of this point are determined by the maximum values of the objectives. The values are also normalized using the (constant) volume between the coordinate systems' origin and the hypervolume reference point.

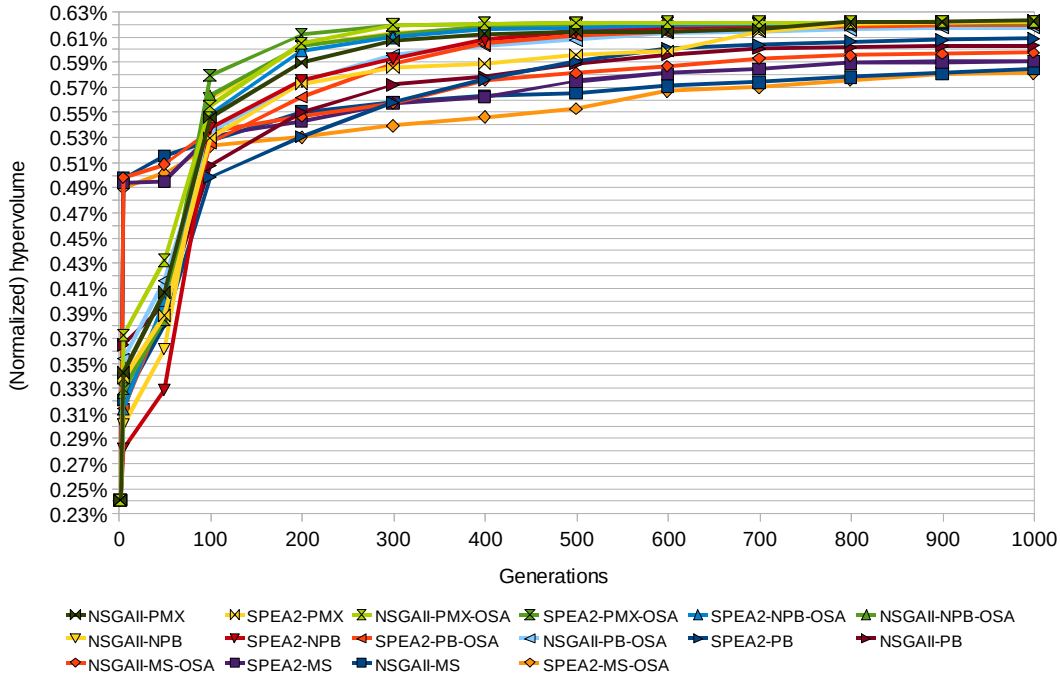

Fig. 24 (Normalized) hypervolumes, for all evaluated algorithms

The hypervolume grows significantly until the first 200 – 300 generations, for all algorithms. Then, it keeps growing slowly until the last generation. This indicates how the algorithms converge. The algorithms using our developed Mapping Similarity crossover have a very fast convergence speed within the first 100 generations. However, in the end, their hypervolumes are the smallest. This indicates MS leads to the worse performance in this multi-objective case. However, if we want fast results, then this will be a suitable crossover. Looking at the hypervolume values within the last generations, we ordered the algorithms. This order may be seen in the chart's legend. It may be observed that PMX performs the best. It is followed by NPB, PB and finally MS. We also observed that both NSGA-II and SPEA2 performed better with PMX and swap mutation. The performance with OSA mutation was worse. These multi-objective results appear to be in contradiction with our previous single-objective results. The explanation resides in the fact that our two objectives are in a mutual contradiction. OSA mutation, MS and NPB crossover work to optimize energy but, this implicitly leads to worsening the NoC mappings in terms of thermal balance. NPB is more suitable than MS (in this case) because it just identifies hot spot cores, in terms of energy. However, they may also be in terms of thermal balance because a highly communicating core might also have a higher temperature.

Fig. 25 shows the Pareto front obtained in the last generation by combining the Pareto fronts of all the evaluated algorithms. This combined Pareto front holds only the non-dominated individuals from all the merged Pareto fronts.

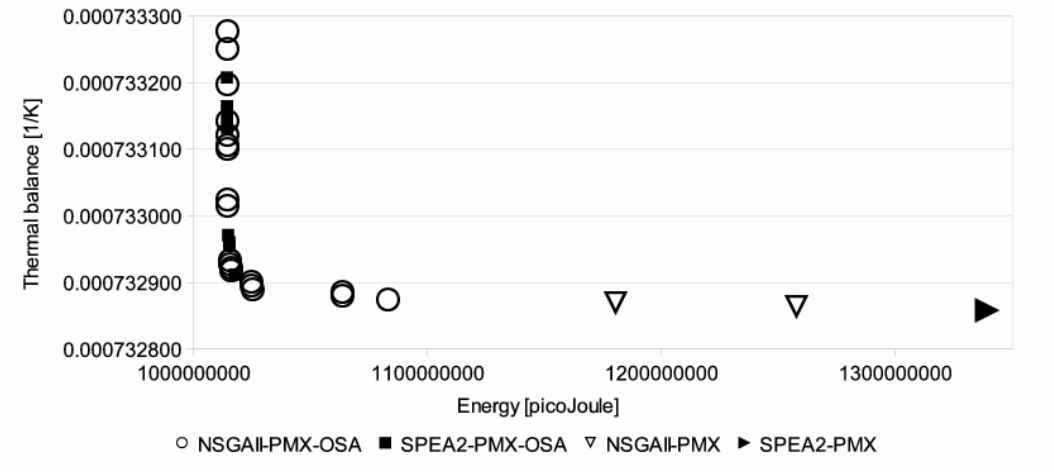

Fig. 25 Combined Pareto front (generation 1000)

It may be seen that PMX is the single crossover that leads to the best solutions, found by either NSGA-II or SPEA2. We also observe there are a lot of good solutions in terms of energy. All these mappings were found using OSA mutation. With swap mutation, we managed to find three good solutions in terms of thermal balance. The significantly higher number of good energy-biased solutions indicates the fact we tried to optimize only energy with NoC application mapping knowledge. Probably using a crossover which also optimized energy was too much bias towards a single objective. This is how we explain PMX was the best performing crossover. Anyway, PMX was one of the best performing crossovers in the single-objective case, too.

8 Application Driven Automatic Design Space Exploration for System-on-Chip Architectures

In this chapter we propose a method for performing an application driven automatic design space exploration for System-on-Chip (SoC) architectures. We integrate UniMap with a Framework for Automatic Design Space Exploration (FADSE) [70] with the purpose of automatically finding the best SoC design for any given application, in a multi-objective way. Our objectives are: SoC energy consumption, SoC area and application runtime.

Using UniMap's features, we simulate an entire computing system, consisting of tens of heterogeneous IP cores that are mapped onto the nodes of a Network-on-Chip.

FADSE automatically configures this System-on-Chip. It then simulates it using UniMap's simulator and gives the simulation results to the DSE algorithm that drives the search process.

We show a feasible DSE workflow that meets our requirements and we identify the most suitable SoC architectures, for a given application, in terms of energy, area and runtime. We also compare four DSE multi-objective algorithms (two genetics and two bio-inspired) with the purpose of identifying the algorithm that performs the best.

8.1 Framework for Automatic Design Space Exploration

The Framework for Automatic Design Space Exploration (FADSE) [71], [72] is developed by Horia Calborean from "Lucian Blaga" University of Sibiu, Romania, as part of his PhD thesis [70]. FADSE is a client-server tool that includes many state of the art algorithms through jMetal [73]. FADSE was successfully used for a multi-objective, hardware-software co-design exploration of the design space for a superscalar system [74], [75].

8.2 Design Space Exploration Workflow

Our DSE workflow starts with mapping applications onto NoC architectures using UniMap's algorithms. The mappings are evaluated by estimating the NoC communication energy with an analytical model. The best solutions found are saved into a database.

For each application, FADSE searches for the best SoC design by considering the first ten best mappings (a higher number of best mappings may be used depending on how many resources are available). Note that we select these mappings from all best mappings found by all UniMap mapping algorithms: Simulated Annealing, Branch and Bound, Optimized Simulated Annealing and Elitist Genetic Algorithm and Elitist Evolutionary Strategy, with all their variants, evaluated in Chapter 7.

Then we configure FADSE to start a DSE process, driven by a multi-objective algorithm. FADSE evaluates different System-on-Chip architectures. Firstly, it selects the type for each IP core. The given mapping already contains information about what IP core will execute what task. However, FADSE will try with other compatible IP cores as well. Any IP core capable of executing a task is considered compatible with that task. Note that the analytical model used for obtaining the best mappings does not account for

IP core types. Secondly, it instantiates a SoC architecture by placing the selected IP cores onto the nodes of a NoC that it configures. Finally, it calls UniMap's ns-3 NoC simulator. We model the tasks' execution using Finite State Machines. The network communications are created using our network traffic generator. ns-3 NoC measures application runtime, SoC energy and SoC area. These are the three objectives of our DSE workflow.

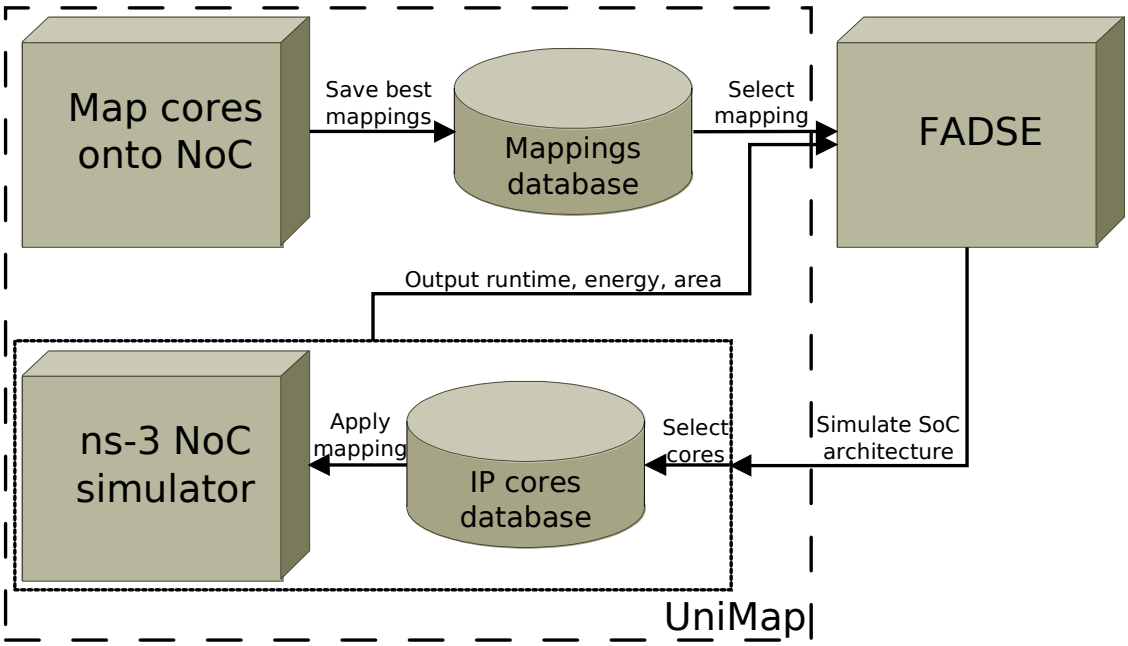

Fig. 26 Application driven DSE workflow for SoC designs

We use the E3S [44] IP core library, which provides data about the power consumed by each core while executing a certain task and while idle and the area occupied by every core.

For our NoC architecture, power and area metrics are measured using ORION 2.0 [45], which is integrated with UniMap's NoC simulator (see section 4.2). We work with the Network-on-Chip total power, which includes leakage and dynamic power for routers and links. Similarly, NoC area is the sum of routers and links area.

We measure application runtime by running the application for a specified number of CTG iterations. We determine the number of CTG iterations empirically, so that the simulations run fast enough so that our DSE process ends in a feasible amount of time.

The output of this workflow is a Pareto front with the "best" (near optimal) SoC configurations, for a particular application.

In the next section we give details about how exactly we performed the simulations, on which benchmarks, what architectural parameters we varied and how UniMap and FADSE were configured. We must point out that, during the workflow, the NoC topology is kept unchanged. This is because the topology is basically the single NoC architectural element used by the mapping algorithms. Changing it would lead to inconsistencies, i.e. doing and comparing mappings for different NoCs. Obviously, our workflow may also be applied for different NoC topologies. By doing so we could also

determine the most suitable NoC topology. However, this would require adapting our application mapping algorithms for these other NoC topologies. Only then we will be able to obtain the best mappings for other NoC topologies.

8.3 Experimental Results

We show next some preliminary results obtained with our previously presented application driven design space exploration technique for System-on-Chip architectures. We managed to explore all ten best mappings just for the *telecom* benchmark. For the rest of benchmarks we explored only the first best mapping.

We start with the *telecom* DSE. In the next figure we use the hypervolume metric to show how our four DSE algorithms progress while searching for the best SoC designs for the *telecom* benchmark. We obtained hypervolumes for each DSE algorithm, on every one of the ten *telecom* mappings. Then we computed the average hypervolume.

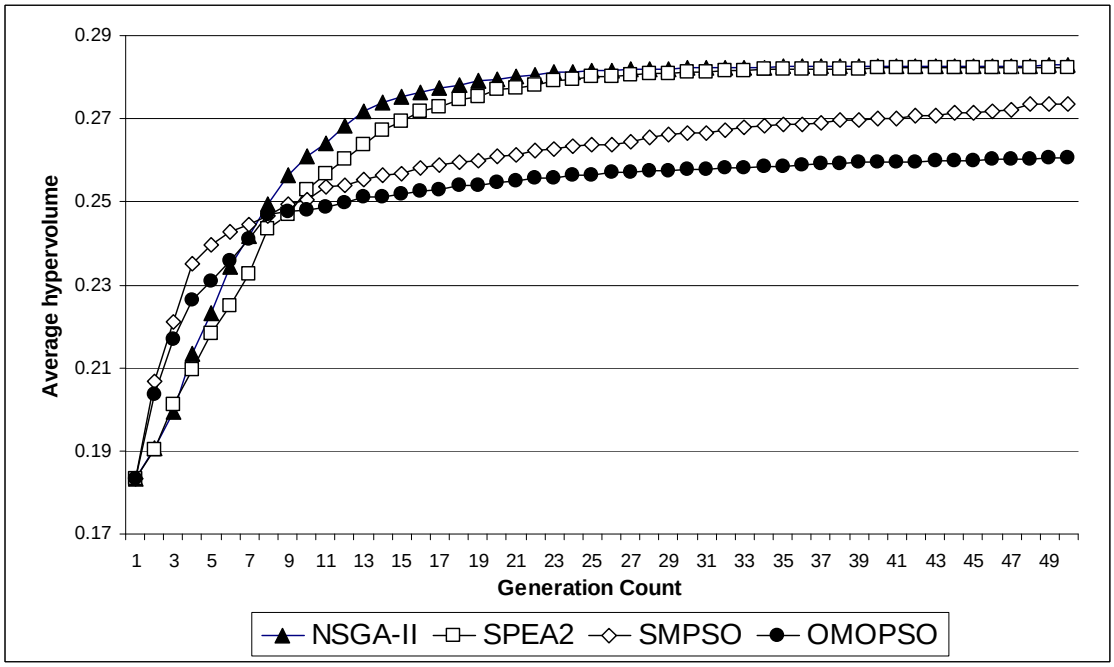

Fig. 27 Average hypervolumes over all ten best telecom mappings

It can be seen that the two genetic algorithms (NSGA-II and SPEA2) obtained the best hypervolumes. NSGA-II has a slightly faster convergence speed than SPEA2. In the last ten generations, both of them saturate; they no longer find significantly better solutions. SMPSO performs better than OMOPSO but, both PSO algorithms perform worse than the genetic algorithms in terms of solution quality (we used the same hypervolume reference point). However, they have the fastest convergence speed. Only after 8-9 generations the genetics recover and surpass the PSO algorithms.

We also compared the four algorithms using the coverage metric (results are omitted due to space constraints). We concluded that SPEA2 has the best overall results. The following figure shows the Pareto front obtained with SPEA2, by combining the Pareto fronts from all ten *telecom* mappings.

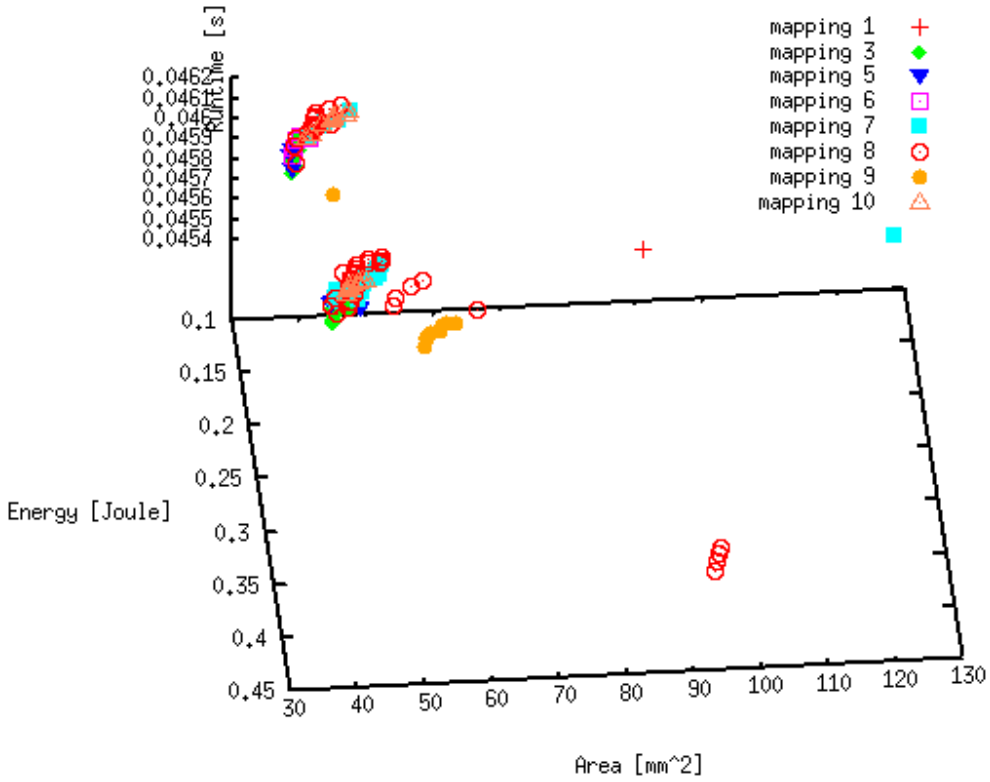

Fig. 28 SPEA2 Pareto front, for telecom

We observe that the Pareto front contains solutions from all eight of the ten best *telecom* mappings. We obtained the best energy consumption with the eight mapping. The smallest area was given by mappings three and five. With exactly the same area, the third mapping has a better energy, while the fifth has a better application runtime. Finally, the lowest application runtime was found on a SoC design corresponding to mapping eight.

It is interesting to see that we did not obtain the best energy with the first best mapping, which analytically gave us the lowest NoC communication energy. This can be due to several facts. Firstly, we analytically estimated only the NoC communication energy. With this approach we compute the entire SoC energy (IP cores energy is also included). Secondly, the analytical model is unable to capture the dynamic network effects (network congestions). Thirdly, FADSE does not obviously perform an exhaustive search. It is possible that we might get better energy results with mapping one than with mapping eight. This shows the need to perform better exploration of the design space. Using domain-knowledge to constrain the search space and applying fuzzy rules are two approaches that could improve the DSE technique [70].

Finally, we combined all the Pareto fronts obtained with all our algorithms, for all ten *telecom* mappings.

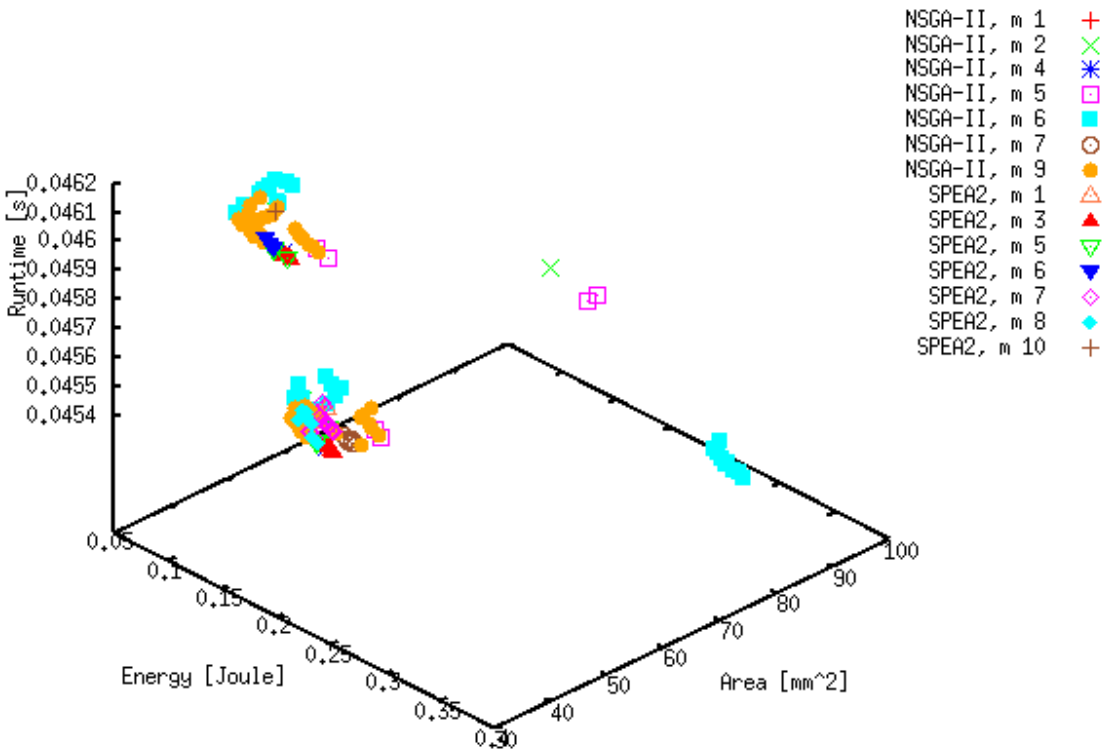

Fig. 29 Combined Pareto front for telecom benchmark

It can be observed that all the solutions found with SMPSO and OMOPSO are dominated by the solutions found with the genetic algorithms. While in terms of SoC area the best solutions are the ones found with SPEA2 (with mappings 3 and 5), in terms of energy and runtime, NSGA-II found, with mapping six, better results than SPEA2 (with mapping eight).

The following table summarizes the best SoC designs found for the *telecom* application. Due to space constraints, we do not show the 30 IP cores selected for every SoC architecture.

Objective	Algorithm	Mapping	NoC parameters					SoC energy [Joule]	SoC area [mm^2]	Application runtime [ms]
			Frequency [MHz]	Buffer size [flits]	Flit size [bytes]	Packet size [flits]	Routing			
Energy	NSGA-II	6	100	4	4	10	YX	0.09516	50.11	46.1144
Area	SPEA2	5	200	1	4	10	XY	0.15818	37.37	46.1132
Area	SPEA2	3	400	1	4	10	YX	0.16793	37.37	46.1111
Runtime	NSGA-II	6	900	4	32	6	YX	0.34191	81.22	45.4

The lowest energy was obtained (in accordance with our intuition) when the NoC operated at the lowest frequency allowed by our DSE workflow. The SoCs with the smallest area use some of the smallest IP cores. Also, the NoC buffers are only one flit in size. As compared with the best energy and runtime SoC designs, the two area designs use only 25% NoC buffering resources. The two designs with the smallest area essentially differ by the NoC frequency. The faster one uses a NoC that is twice faster. The SoC with the best runtime runs *telecom* with more than half a millisecond than the

other three SoCs, which are differentiated in terms of speed by only a few fractions of a microsecond. The best runtime SoC architecture also requires a much faster NoC. It also operates with bigger packets. All these reflect on considerably higher energy and bigger area. Finally, we also observe that routing also influences the architecture's performance. Our best SoC designs for *telecom* use both XY and YX routing protocols.

Now we continue with the MPEG-4 DSE. The following figure presents the hypervolume of each DSE algorithm, for the best MPEG-4 mapping found analytically.

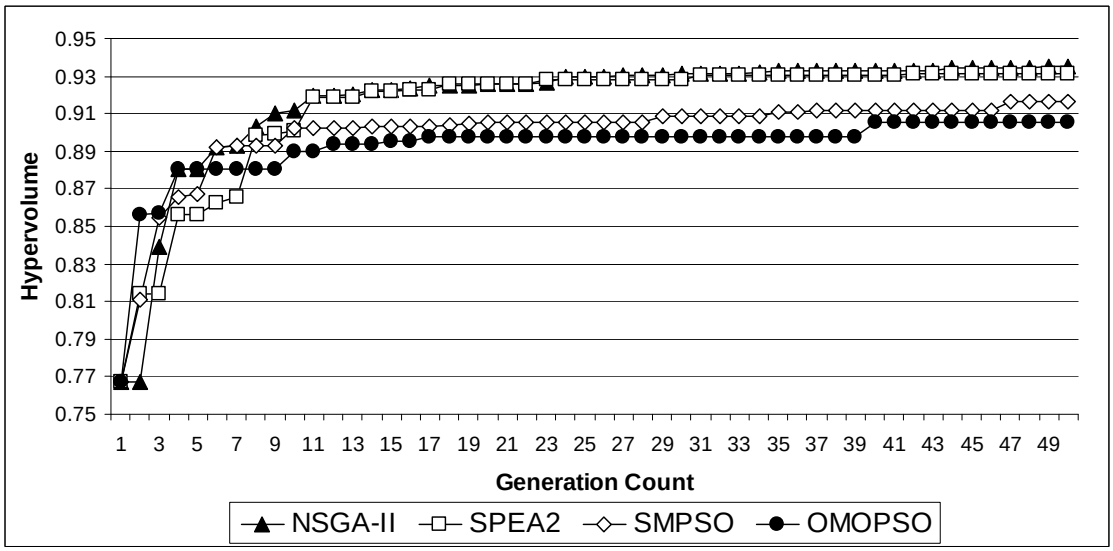

Fig. 30 Hypervolumes for the first best MPEG-4 mapping

The results obtained for *telecom* are consistent with the ones presented here. Again the two genetic algorithms perform better than the particle swarm optimization algorithms. NSGA-II converges faster than SPEA2. In terms of quality of results it seems that NSGA-II is the best. Again, SMPSO performed better than OMOPSO. Like for *telecom*, MPEG-4 results show us that it matters more the class the algorithm belongs to (evolutionary or bio-inspired), rather than the specific implementation.

We computed the coverage, trying to choose the best algorithm from each class. The results are presented in Fig. 31 and Fig. 32.

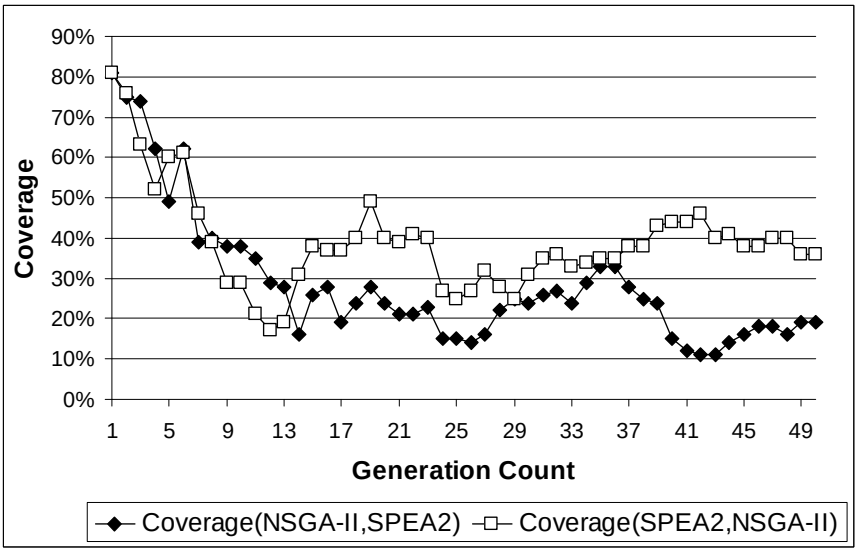

Fig. 31 Coverage comparison between NSGA-II and SPEA2, for MPEG-4

For the first generations no clear distinction can be made between the two algorithms. However, looking at the last generations, we conclude that there are more individuals produced by SPEA2 that dominate the NSGA-II individuals. This contradicts the hypervolume chart where NSGA-II seemed to perform better. We thoroughly analyzed the Pareto fronts obtained by the two genetic algorithms. Some of the solutions discovered by NSGA-II are better than the ones obtained by SPEA2 and some are worse (in accordance with the coverage metric). It is hard to establish the best one because it depends on the requirements of the designer. Still, the results obtained by NSGA-II seemed a little more spread in the objective space.

The same behavior can be observed between OMOPSO and SMPSO. SMPSO performed better from the hypervolume point of view, but here OMOPSO is the best. Again, we analyzed the Pareto fronts approximations and from our point of view SMPSO had better results. We emphasize that this is a subjective appreciation and for other designers the order might be changed.

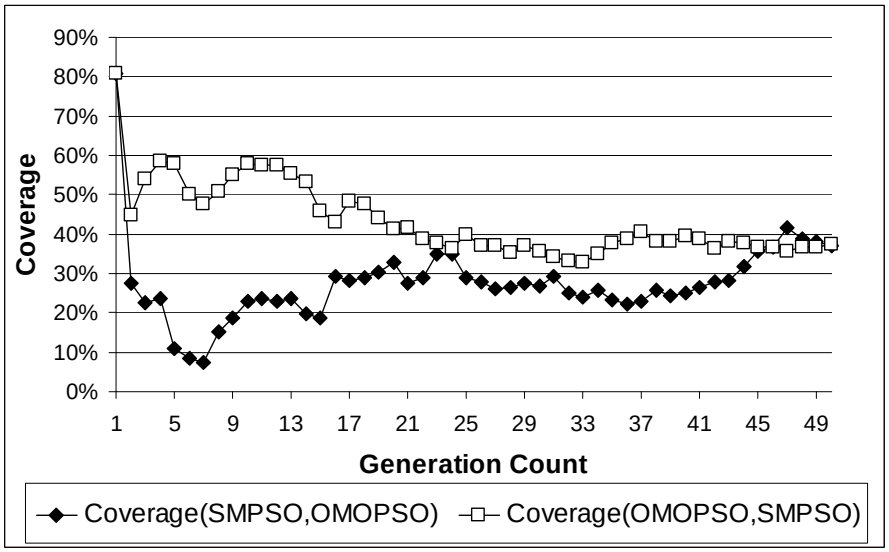

Fig. 32 Coverage comparison between SMPSO and OMOPSO, for MPEG-4

For our last comparison we selected the best algorithms from the coverage point of view: SPEA2 and OMOPSO. In the next figure we present the coverage comparison between the two algorithms. SPEA2 is clearly the best, by dominating almost 100% of the individuals found by OMOSPO. OMOSPO does not dominate almost any individuals obtained by the genetic algorithm. It is interesting to observe that OMOPSO is better for the first generations. This is because of the faster convergence speed of the PSO algorithms.

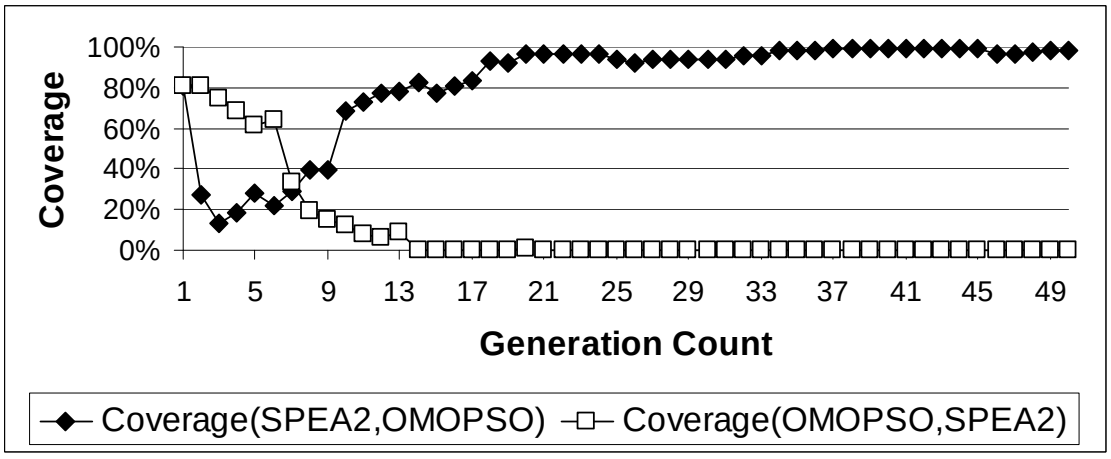

Fig. 33 Coverage comparison between SPEA2 and OMOPSO, for MPEG-4

The following figure presents the most spread Pareto front, which was obtained by the NSGA-II algorithm. Through interpolation we also obtained a surface grid that gives us a better view of the Pareto surface.

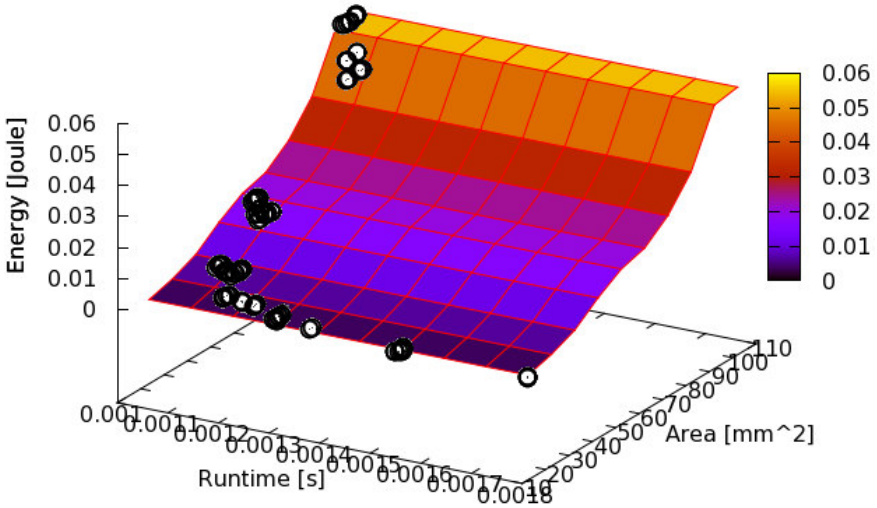

Fig. 34 MEPG-4 NSGA-II Pareto front

As expected, it can be observed that there is no SoC design for the MPEG-4 application that is best for all three objectives. The fastest designs consume more energy and occupy more area. The slowest architectures consume less energy and need less area. In between we have a lot of solutions that are better for energy and worse for area and vice versa.

We conclude this preliminary research by presenting the hypervolumes obtained for the first best analytical mapping of H.264 and VOPD benchmarks.

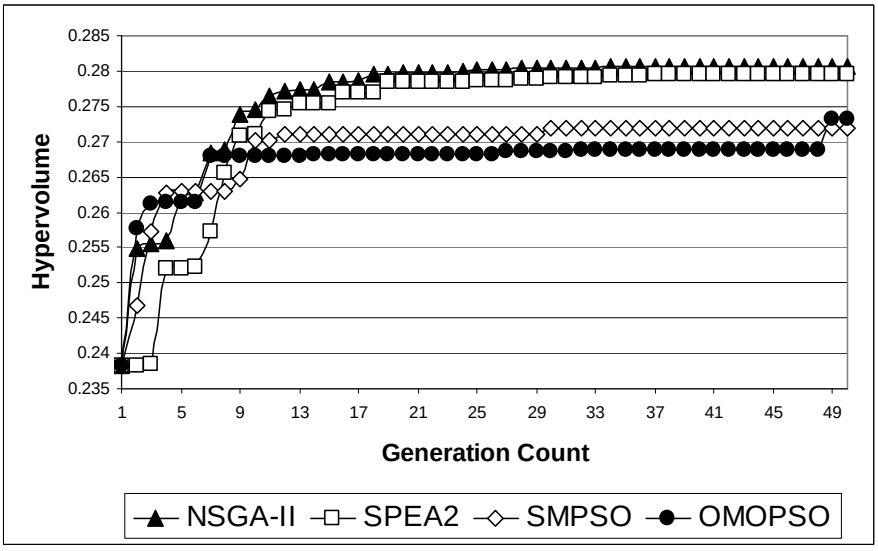

Fig. 35 Hypervolumes for the first best H.264 mapping

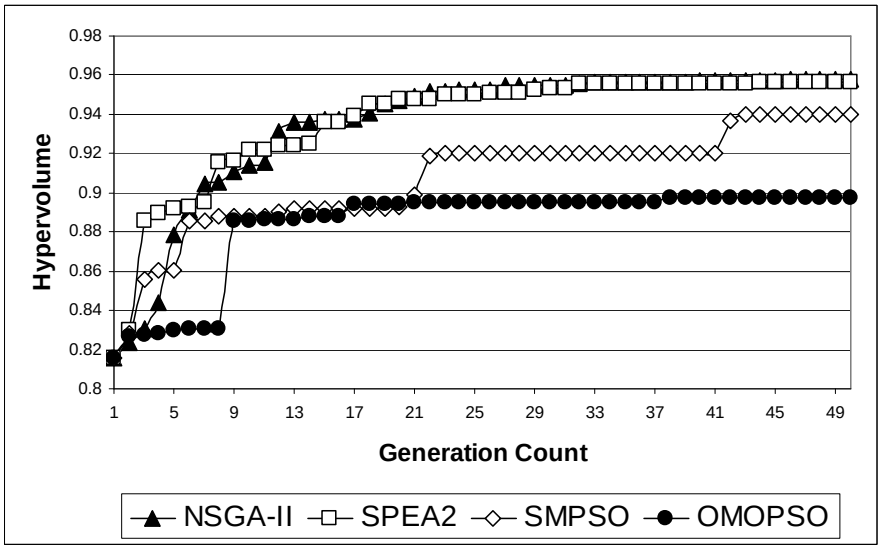

Fig. 36 Hypervolumes for the first best VOPD mapping

These H.264 and VOPD hypervolume results are in correlation with our previous results. Our conclusion is that the genetic algorithms find better solutions than the particle swarm optimization methods. The PSO algorithms manage to converge faster only for the H.264 decoder. For VOPD, SMPSO performs clearly better than OMOPSO. We also observe an unsteady convergence speed for the PSOs. For a large number of generations their evolution is insignificant. Then, they manage to find at least one significantly better individual, which makes their hypervolume grow noticeable.

9 Conclusions and Further Work

This work addresses the Network-on-Chip application mapping problem. After we introduced the novel Network-on-Chip paradigm in Chapter 0, we focused on the mapping problem. Chapter 3 presents the problem along with a state of the art on the heuristic algorithms used to address it. In Chapter 4 we show our developed unified framework for the evaluation and optimization of Network-on-Chip application mapping algorithms. In Chapter 5 we presented the benchmarks used in our research, in this emerging NoC research field that still lacks a standard benchmarking methodology. With UniMap, we evaluated and optimized a simulated annealing algorithm using a domain-knowledge approach (Chapter 6). We also evaluated and optimized evolutionary algorithms by proposing problem aware genetic operators (Chapter 7). Finally, in Chapter 8, we proposed and used a design space exploration workflow for an application driven automatic design space exploration for Systems-on-Chip. Our algorithms' evaluations were performed using both analytical models and simulators. We considered single and multi-objective approaches.

More precisely, this thesis makes the following contributions:

- An introduction to Network-on-Chip architectures with an emphasis on the most common network topologies and routing protocols used in this research field;
- Taxonomy for the classification of Network-on-Chip application mapping algorithms;
- State of the art regarding algorithms for Network-on-Chip application mapping;
- UniMap: a developed unified framework for the evaluation and optimization of NoC application mapping algorithms;
- UniMap runs on High Performance Computing Systems using job schedulers to automatically and optimally distribute simulations;
- Common model based on XML schemas for representing real applications and networks;
- UniMap integrates state of the art NoC application mapping algorithms like Simulated Annealing and Branch and Bound;
- UniMap integrates jMetal, a library with single objective and multi-objective state of the art evolutionary algorithms, which can be used as application mapping algorithms;
- ns-3 NoC, our developed Network-on-Chip simulator, with two router architectures, three routing protocols, three switching mechanisms and k-ary d-cube topologies;
- Network traffic generator based on communication patterns of real applications, described through Communication Task Graphs and Application Characterization Graphs;
- ns-3 NoC integrates ORION 2.0, a state of the art tool for Network-on-Chip power consumption and area estimation;
- Using ns-3 NoC, we showed that the Irvine architecture helps at decreasing the network congestion. The network is significantly less congested when data flits are transmitted faster than head flits;

- With ns-3 NoC, we showed how increasing the network buffers' size improves the NoC's average packet latency;
- Using ns-3 NoC, we evaluated different network topologies: 2D mesh, 2D torus, 3D mesh, 3D torus and hypercube. We concluded that topologies like tori and hypercube can give better NoC performance than meshes can;
- UniMap integrates the E3S benchmark suite and some of the most used CTGs and APCGs available in literature. Because NoC benchmarking is still work in progress, we effectively created our own benchmark suite;
- We propose and use for Network-on-Chip benchmarking two communication patterns taken from a H.264 decoder system available in the research community;
- Using domain-knowledge, we developed an Optimized Simulated Annealing (OSA) algorithm. It performs a dynamic and implicit core clustering and limits the number of iterations per annealing temperature based on the given application and network.
- We showed that Simulated Annealing can be feasible for NoC application mapping when domain-knowledge is used. OSA is approximately 99% faster than a generic Simulated Annealing algorithm, without losing the solution quality;
- The results obtained with OSA showed that Simulated Annealing is feasible for NoC 2D meshes larger than 10x10. Previous research stated the contrary;
- OSA is comparable to Branch and Bound in terms of memory consumption and speed. It mapped 97 cores on a 10x10 2D mesh in a time slower by only 3% than the time required by Branch and Bound;
- As the problem size increases, OSA gives significantly better solutions than Branch and Bound. The mappings found with Branch and Bound were with more than 70% worse than OSA's mappings when working with more than 64 IP cores;
- We showed Branch and Bound's limitations. This algorithm was unable to map an application with 215 cores, onto a 15x15 NoC, because more than 98% of the search space was pruned;
- We developed an Elitist energy- and performance-aware Genetic Algorithm (EGA). EGA is integrated in jMetal;
- We extended jMetal with the Position Based crossover;
- We evaluated EGA and an Elitist Evolutionary Strategy (EES) using different genetic operators (four crossovers, two mutations => 12 algorithm variants);
- We concluded that evolutionary algorithms are superior to algorithms like OSA, for NoCs with tens, hundreds of nodes. We found that, for the big benchmarks, all the best solutions were given by evolutionary algorithms (none by OSA);
- We proposed a meta-heuristic algorithm consisting of an evolutionary algorithm that uses as mutation operator a state of the art application mapping algorithm;
- EGA and EES work better with OSA mutation than with swap mutation. OSA integrated successfully into the Evolutionary Algorithms;
- We designed two problem specific crossover operators: NoC Position Based and Mapping Similarity. NoC Position Based crossover improves the standard Position Based crossover for our problem. Mapping Similarity crossover exchanges information between the parent individuals. It does not simply work as a mutation operator, like the other state of the art NoC application mapping crossover operators do;

- With NoC Position Based crossover, EGA had the best solution percentage on the big benchmarks;
- We found Mapping Similarity to be the crossover operator that contributes the most at obtaining a good mapping. It performed best at 50% - 60% mutation probability. The rest of crossovers required higher mutation rates;
- We found EES to perform better than EGA. Although we managed to improve the genetic algorithm through our crossover operators, using an algorithm that works only with (context-aware) mutation proved to be better. Finding a suitable context-aware crossover for NoC application mapping is more difficult than finding an efficient context-aware mutation;
- EES with OSA mutation was the algorithm that managed to converge the fastest;
- Using two state of the art multi-objective algorithms (NSGA-II and SPEA2) with our genetic operators, we evaluated (with analytic models) the mappings in terms of NoC communication energy and NoC thermal balance. The two objectives are contradictory and, as such, our developed operators did not lead to the best performance. However, we did find the best solutions, in terms of energy, with OSA mutation. A suitable crossover operator for the NoC application mapping problem is even more difficult to find if we consider multi-objective optimization;
- UniMap connects with the Framework for Automatic Design Space Exploration;
- We proposed an application driven automatic Design Space Exploration technique for System-on-Chip architectures. The goal is that, for a given application, to automatically determine the best System-on-Chip design, with the following objectives: SoC energy, SoC area and application runtime;
- Using our developed ns-3 NoC simulator and FADSE, we explored the NoC architectural space for different real applications;
- We showed that the best analytical mappings are not necessarily the best ones when using a NoC simulator;
- The genetic algorithms (NSGA-II and SPEA2) were clearly more suited for our design space exploration workflow than the particle swam optimization methods (SMPSO and OMOPSO). Still, the PSO algorithms converged faster.

As future work, we intend to improve UniMap. We are interested in extracting communication patterns from parallel applications. The first step will be to integrate CETA tool. This will allow us to obtain Communication Task Graphs from shared memory parallel programs. The second step will be to similarly use an MPI library that allows intercepting the communications from message passing parallel applications.

Another direction for extending our unified framework is to implement other state of the art Network-on-Chip application mapping algorithms. For example, the comparisons between OSA and Cluster Simulated Annealing [57] runtimes are very likely to be unfair. This can be due to several reasons: (1) OSA is written in Java but, we do not know yet how CSA is implemented, (2) OSA is energy aware and uses the cost function from [25], while CSA is bandwidth and latency constrained, using the cost function from [76] and (3) CSA does not specify the number of generations per temperature level.

Also, we consider further improving our developed NoC simulator. Improving the router architecture with virtual channels and allocators is an example. This will bring our

router implementation closer to real router designs.

Regarding our developed crossover operators (see Section 7.3) they are suitable only for the communication energy objective. They must be adapted to work in a multi-objective case. Even OSA mutation was designed only for energy minimization. Therefore, evaluation and optimization of such algorithms, in a multi-objective context will be more difficult. Using standard crossover and mutation operators simplifies the problem a lot but, such operators are not aware of the problem.

We also plan to continue our research regarding application driven automatic design space exploration for System-on-Chip architectures (see Chapter 8). The presented results are still preliminary. We intend doing more simulations so that we identify the best SoC designs for applications other than *telecom*, too. We also intend to do a more accurate modeling of our SoC designs by increasing the accuracy with which we simulate the IP cores and by varying the NoC topology as well. As for the design space explorer, we intend to use more domain-knowledge so that we can constrain and better explore the huge architectural space. We believe our approach can be extended for performing automatic design space exploration for High Performance Computing systems.

Finally, we refer to a research niche that we identified during this PhD thesis but, unfortunately we have not had enough time to exploit it, yet. We believe that Network-on-Chip application mapping problem can be addressed using graph theory. More precisely, we refer to **graph isomorphism**, which is the problem of verifying if two graphs are actually the same. Two graphs $A = (V_A, E_A)$ and $N = (V_N, E_N)$ are isomorphic if and only if there is a *bijective* mapping $M : V_A \rightarrow V_N$, between the graph nodes, such that the following equivalence is true: $\forall e_1, e_2 \in V_A : (e_1, e_2) \in E_A \Leftrightarrow (M(e_1), M(e_2)) \in E_N$. This means a unique mapping between the corresponding edges of the two graphs is required. For weighted graphs, the condition can be extended to include the weights as well. **Subgraph isomorphism** requires the mapping *M* to be only *injective*. **Graph monomorphism** is a weaker type of subgraph isomorphism. The equivalence relation must be just an implication $(\forall e_1, e_2 \in V_A : (e_1, e_2) \in E_A \Rightarrow (M(e_1), M(e_2)) \in E_N)$. Considering the above definitions and that the two graphs (*A* and *N*) are an Application Characterization Graph (APCG) and, respectively, a NoC topology graph, the Network-on-Chip application mapping problem can be viewed as a graph monomorphism problem. Indeed, it is mentioned in [77] that the quadratic assignment problem can be formulated as a graph monomorphism problem. Currently, there is no known polynomial-time algorithm for the monomorphism problem [78]. However, special graph types, like planar graphs, can theoretically be solved in a linear time [79]. Using the Boyer-Myrvold algorithm [80], we tested for planarity all the APCGs used in this work. All of them proved to be planar graphs. We also integrated in UniMap the VF2 [81] graph matching algorithm and used it to determine if an isomorphism exists between any APCG and its corresponding NoC topology graph. We found none but, this is understandable because we should search for monomorphisms, not for isomorphisms. We found little NoC research using this idea. Graph isomorphism is used in [82] to identify the isomorphically unique NoC topology graphs. VF2 algorithm is used in [83] to perform subgraph isomorphism in order to decompose an APCG into a set of predefined communication pattern graphs. We believe approaching the NoC application mapping problem as a graph monomorphism problem is worth researching.

10 Selected References

[1] M. Duranton et al., "The HiPEAC Vision," *HiPEAC Roadmap*, 2010. [Online]. Available: http://www.hipeac.net/system/files/LR_3910_hipeac_roadmap-2010-v3.pdf.

[2] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach, 4th Edition*, 4th ed. Morgan Kaufmann, 2006.

[3] R. Marculescu, U. Y. Ogras, L.-S. Peh, N. E. Jerger, and Y. Hoskote, "Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 28, no. 1, pp. 3-21, 2009.

[4] G. Moore, "Cramming More Components onto Integrated Circuits," *Electronics*, vol. 38, no. 8, pp. 114-117, Apr. 1965.

[5] G. E. Moore, "Excerpts from a conversation with Gordon Moore: Moore's Law, 2005," *URL ftp://download. intel. com/museum/Moores_Law/Video-Transcripts/Excepts_A_Conversation_with_Gordon_Moore. pdf*.

[6] L. Vinţan N., *Arhitecturi de procesoare cu paralelism la nivelul instrucțiunilor*. Editura Academiei Române, București, 2000.

[7] L. Vinţan N., *Prediction Techniques in Advanced Computing Architectures*. Matrix Rom Publishing House, Bucharest, 2007.

[8] A. Florea and L. Vinţan N., *Simularea şi optimizarea arhitecturilor de calcul în aplicaţii practice*. Editura Matrix Rom, Bucureşti, 2003.

[9] L. Vinţan N., "Direcţii de cercetare în domeniul sistemelor multicore," *Revista Română de Informatică și Automatică, ICI București*, vol. 19, no. 3, 2009.

[10] **C. Radu**, H. Calborean, A. Crapciu, A. Gellert, and A. Florea, "An Interactive Graphical Trace-Driven Simulator for Teaching Branch Prediction in Computer Architecture," in *The 6th EUROSIM Congress on Modeling and Simulation*, 2007, p. 58.

[11] A. Florea, **C. Radu**, H. Calborean, A. Crapciu, A. Gellert, and L. Vintan, "Designing an Advanced Simulator for Unbiased Branches Prediction," *Proceedings of 9th International Symposium on Automatic Control and Computer Science*, 2007.

[12] A. Florea, **C. Radu**, H. Calborean, A. Crapciu, A. Gellert, and L. Vinţan, "Understanding and Predicting Unbiased Branches in General-Purpose Applications," *Buletinul Institutului Politehnic Iasi, Tome LIII (LVII), fasc. 1-4, Section IV, Automation Control and Computer Science Section*, pp. 97-112, 2007.

[13] **C. Radu**, "Implementing a multicore Shared Memory Architecture using Transaction Level Modelling with UNISIM," Diploma project (Bachelor), "Lucian Blaga" University of Sibiu, Romania (in Romanian, supervisor Professor Lucian Vintan, PhD), Sibiu, Romania, 2008.

[14] **C. Radu**, H. Calborean, A. Florea, A. Gellert, and L. Vintan, "Exploring Some Multicore Research Opportunities. A First Attempt.," in *Advanced Computer Architecture and Compilation for Embedded Systems*, Terrassa (Barcelona), Spain, 2009.

[15] T. Bjerregaard and S. Mahadevan, “A survey of research and practices of Network-on-chip,” *ACM Comput. Surv.*, vol. 38, no. 1, Jun. 2006.
[16] K. Asanovic et al., “The landscape of parallel computing research: A view from berkeley,” Citeseer, 2006.
[17] P. Guerrier and A. Greiner, “A generic architecture for on-chip packet-switched interconnections,” *Proceedings of the conference on Design, automation and test in Europe*, pp. 250–256, 2000.
[18] A. Hemani et al., “Network on chip: An architecture for billion transistor era,” in *Proceeding of the IEEE NorChip Conference*, 2000, pp. 166–173.
[19] W. J. Dally and B. Towles, “Route packets, not wires: on-chip inteconnection networks,” in *Proceedings of the 38th annual Design Automation Conference*, Las Vegas, Nevada, United States, 2001, pp. 684-689.
[20] D. Wingard, “Micronetwork-based integration for SOCs: 673,” *Proceedings of the 38th annual Design Automation Conference*, p. 677–, 2001.
[21] E. Rijpkema, K. Goossens, and P. Wielage, “A Router Architecture for Networks on Silicon,” *IN PROCEEDINGS OF PROGRESS 2001, 2ND WORKSHOP ON EMBEDDED SYSTEMS*, p. 181--188, 2001.
[22] S. Kumar et al., “A network on chip architecture and design methodology,” in *isvlsi*, 2002, p. 0117.
[23] G. de Micheli and L. Benini, “Networks on Chip: A New Paradigm for Systems on Chip Design,” *Proceedings of the conference on Design, automation and test in Europe*, p. 418–, 2002.
[24] M. Duranton et al., “The HiPEAC Vision,” *HiPEAC Roadmap*, 2010. [Online]. Available: http://www.hipeac.net/system/files/LR_3910_hipeac_roadmap-2010-v3.pdf.
[25] J. Hu and R. Marculescu, “Energy-aware mapping for tile-based NoC architectures under performance constraints,” in *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*, Kitakyushu, Japan, 2003, pp. 233-239.
[26] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman & Co. New York, NY, USA, 1979.
[27] I. Walter, I. Cidon, A. Kolodny, and D. Sigalov, “The era of many-modules SoC: revisiting the NoC mapping problem,” in *2nd International Workshop on Network on Chip Architectures, 2009. NoCArc 2009*, 2009, pp. 43-48.
[28] U. Y. Ogras, J. Hu, and R. Marculescu, “Key research problems in NoC design: a holistic perspective,” in *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, Jersey City, NJ, USA, 2005, pp. 69-74.
[29] R. P. Dick and N. K. Jha, “MOCSYN: multiobjective core-based single-chip system synthesis,” in *Proceedings of the conference on Design, automation and test in Europe*, Munich, Germany, 1999, p. 55.
[30] C. Grecu et al., “Towards Open Network-on-Chip Benchmarks,” in *Proceedings of the First International Symposium on Networks-on-Chip*, Princeton, NJ, 2007, p. 205.
[31] R. P. Dick, D. L. Rhodes, and W. Wolf, “TGFF: task graphs for free,” in *Proceedings of the 6th international workshop on Hardware/software codesign*, Seattle, Washington, United States, 1998, pp. 97-101.

[32] Y.-K. Kwok and I. Ahmad, “Static scheduling algorithms for allocating directed task graphs to multiprocessors,” *ACM Computing Surveys (CSUR)*, vol. 31, pp. 406–471, Dec. 1999.

[33] D. Towsley, “Allocating programs containing branches and loops within a multiple processor system,” *IEEE Transactions on Software Engineering*, vol. 12, pp. 1018–1024, Oct. 1986.

[34] H. El-Rewini and H. H. Ali, “Static scheduling of conditional branches in parallel programs,” *Journal of Parallel and Distributed Computing*, vol. 24, pp. 41–54, Jan. 1995.

[35] A.-H. Liu and R. P. Dick, “Automatic run-time extraction of communication graphs from multithreaded applications,” in *Proceedings of the 4th international conference on Hardware/software codesign and system synthesis*, Seoul, Korea, 2006, pp. 46-51.

[36] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 1st ed. Prentice Hall, 1995.

[37] J. Hu and R. Marculescu, “Energy- and performance-aware mapping for regular NoC architectures,” *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS*, vol. 24, no. 4, p. 551--562, 2005.

[38] J. Hu and R. Marculescu, “Communication and task scheduling of application-specific networks-on-chip,” *IEE Proceedings - Computers and Digital Techniques*, vol. 152, no. 5, p. 643, 2005.

[39] R. Pop and S. Kumar, “A survey of techniques for mapping and scheduling applications to network on chip systems,” *School of Engineering, Jonkoping University, Research Report*, vol. 4, p. 4, 2004.

[40] **C. Radu** and L. Vinţan, “UNIMAP: UNIFIED FRAMEWORK FOR NETWORK-ON-CHIP APPLICATION MAPPING RESEARCH,” *Acta Universitatis Cibiniensis Technical Series*, May 2011.

[41] **C. Radu** and L. Vinţan, “Towards a Unified Framework for the Evaluation and Optimization of NoC Application Mapping Algorithms,” in *ACACES 2010 Poster Abstracts*, Terrassa (Barcelona), Spain, 2010, pp. 163 - 166.

[42] **C. Radu**, “Unified Framework for Network-on-Chip Application Mapping,” *unimap - Project Hosting on Google Code*. [Online]. Available: https://code.google.com/p/unimap/. [Accessed: 04-Feb-2011].

[43] “ULBS HPC cluster.” [Online]. Available: http://zamolxe.hpc.ulbsibiu.ro/. [Accessed: 07-Feb-2011].

[44] “The Embedded System Synthesis Benchmarks Suite (E3S) website.” [Online]. Available: http://ziyang.eecs.umich.edu/~dickrp/e3s/.

[45] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi, “ORION 2.0: a fast and accurate NoC power and area model for early-stage design space exploration,” in *Proceedings of the Conference on Design, Automation and Test in Europe*, 3001 Leuven, Belgium, Belgium, 2009, pp. 423–428.

[46] **C. Radu** and L. Vinţan, “Optimizing Application Mapping Algorithms for NoCs through a Unified Framework,” in *Roedunet International Conference (RoEduNet), 2010 9th*, Sibiu, Romania, 2010, pp. 259 - 264.

[47] S. E. Lee and N. Bagherzadeh, “Increasing the throughput of an adaptive router in network-on-chip (NoC),” in *Proceedings of the 4th international conference on Hardware/software codesign and system synthesis*, Seoul, Korea, 2006, pp. 82-87.
[48] S. Schlingmann, “Selbstoptimierendes Routing in einem Network-on-a-Chip,” Augsburg, Germany, 2007.
[49] A. Gancea, “Simulator pentru proiectarea, evaluarea și optimizarea unor rețele de interconectare tip NoC,” Diploma project (Bachelor), “Lucian Blaga” University of Sibiu, Romania (in Romanian, supervisor Professor Lucian Vintan, PhD), Sibiu, Romania, 2011.
[50] E. Salminen, K. Srinivasan, and Z. Lu, “OCP-IP Network-on-chip benchmarking workgroup,” *OCP-IP*, Dec-2010. [Online]. Available: http://www.ocpip.org/uploads/dynamic_areas/Cv8XdaKTKDztFpWKPqsl/6189/NoC%20Working%20Group%20Overview%20WP.pdf.
[51] E. B. van der Tol, “Mapping of H.264 decoding on a multiprocessor architecture,” in *Proceedings of SPIE*, Santa Clara, CA, USA, 2003, pp. 707-718.
[52] S. Kirkpatrick, C. Gelatt, and M. Vecchi, “Optimization by Simulated Annealing,” *Science*, vol. 220, no. 4598, pp. 671-680, May 1983.
[53] M. A. . Elmohamed, P. Coddington, and G. Fox, “A comparison of annealing techniques for academic course scheduling,” *Practice and Theory of Automated Timetabling II*, p. 92, 1998.
[54] **C. Radu** and L. Vinţan, “Optimized Simulated Annealing for Network-on-Chip Application Mapping,” in *Proceedings of the 18th International Conference on Control Systems and Computer Science (CSCS-18)*, Bucharest, Romania, 2011, vol. 1, pp. 452–459.
[55] SLD:: System Level Design Group @ CMU, “NoCmap: an energy- and performance-aware mapping tool for Networks-on-Chip,” *SLD:: System Level Design Group @ CMU*, 2010. [Online]. Available: http://www.ece.cmu.edu/~sld/wiki/doku.php?id=shared:nocmap.
[56] H. Orsila, E. Salminen, and T. D. Hämäläinen, “Best Practices for Simulated Annealing in Multiprocessor Task Distribution Problems,” in *Simulated Annealing*, I-Tech Education and Publishing KG, 2008, pp. 321-342.
[57] Z. Lu, L. Xia, and A. Jantsch, “Cluster-based Simulated Annealing for Mapping Cores onto 2D Mesh Networks on Chip,” in *Proceedings of the 2008 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, Washington, DC, USA, 2008, pp. 1–6.
[58] **C. Radu**, “Optimized Simulated Annealing for Network-on-Chip Application Mapping,” Computer Science Department, “Lucian Blaga” University of Sibiu, PhD Technical Report no. 3, Jun. 2011.
[59] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Springer, 2008.
[60] C. A. C. Coello, D. A. V. Veldhuizen, and G. B. Lamont, *Evolutionary Algorithms for Solving Multi-Objective Problems*, 1st ed. Springer, 2002.
[61] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1st ed. Addison-Wesley Professional, 1989.
[62] C. J. Glass and L. M. Ni, “The turn model for adaptive routing,” *J. ACM*, vol. 41, no. 5, pp. 874-902, 1994.

[63] Ge-Ming Chiu, "The odd-even turn model for adaptive routing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 7, pp. 729-738, Jul. 2000.
[64] G. Ascia, V. Catania, and M. Palesi, "Multi-objective mapping for mesh-based NoC architectures," in *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, Stockholm, Sweden, 2004, pp. 182-187.
[65] G. Syswerda, "A Study of Reproduction in Generational and Steady State Genetic Algorithms," in *Foundations of Genetic Algorithms*, 1990, pp. 94–101.
[66] D. E. Goldberg and R. Lingle, "Alleles, Loci, and the Traveling Salesman Problem," in *Proc.\ of the International Conference on Genetic Algorithms and Their Applications*, Pittsburgh, PA, 1985, pp. 154-159.
[67] G. Ascia, V. Catania, and M. Palesi, "A Multi-Objective Genetic Approach to Mapping Problem on Network-on-Chip," *JUCS*, vol. 22, p. 2006.
[68] G. Ascia, V. Catania, and M. Palesi, "Mapping cores on network-on-chip," *International Journal of Computational Intelligence Research*, vol. 1, no. 1-2, pp. 109–126, 2005.
[69] **C. Radu**, "Evolutionary Algorithms for Network-on-Chip Application Mapping," Computer Science Department, "Lucian Blaga" University of Sibiu, PhD Technical Report no. 4, Jun. 2011.
[70] H. Calborean, "Multi-Objective Optimization of Advanced Computer Architectures using Domain-Knowledge," PhD Thesis, "Lucian Blaga" University of Sibiu, Romania, 2011 (PhD Supervisor: Prof. Lucian Vintan, PhD), Sibiu, Romania, 2011.
[71] H. Calborean and L. Vintan, "An Automatic Design Space Exploration Framework for Multicore Architecture Optimizations," in *Proceedings of The 9-th IEEE RoEduNet International Conference*, Sibiu, Romania, 2010, pp. 202-207.
[72] H. Calborean and L. Vinţan, "Framework for Automatic Design Space Exploration of Computer Systems," *Acta Universitatis Cibiniensis Technical Series*, May 2011.
[73] J. J. Durillo, A. J. Nebro, and E. Alba, "The jMetal Framework for Multi-Objective Optimization: Design and Architecture," in *CEC 2010*, Barcelona, Spain, 2010, pp. 4138-4325.
[74] R. Jahr, T. Ungerer, H. Calborean, and L. Vintan, "Automatic Multi-Objective Optimization of Parameters for Hardware and Code Optimizations," in *Proceedings of the 2011 International Conference on High Performance Computing & Simulation (HPCS 2011)*, 2011, pp. 308 – 316.
[75] H. Calborean, R. Jahr, T. Ungerer, and L. Vintan, "Optimizing a Superscalar System using Multi-objective Design Space Exploration," in *Proceedings of the 18th International Conference on Control Systems and Computer Science (CSCS), Bucharest, Romania*, Calea Grivitei, nr. 132, 78122, Sector 1, Bucuresti, 2011, vol. 1, pp. 339–346.
[76] S. Murali and G. D. Micheli, "Bandwidth-Constrained Mapping of Cores onto NoC Architectures," in *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 2*, 2004, p. 20896.
[77] D. E. Ghahraman, A. K. C. Wong, and T. Au, "Graph Optimal Monomorphism Algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 10, pp. 181-188, 1980.

[78] "Graph Monomorphism Algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 10, pp. 189-196, 1980.
[79] G. S. Lueker and K. S. Booth, "A Linear Time Algorithm for Deciding Interval Graph Isomorphism," *Journal of the ACM*, vol. 26, pp. 183-195, Apr. 1979.
[80] "On the Cutting Edge: Simplified O(n) Planarity by Edge Addition." [Online]. Available: http://academic.research.microsoft.com/Publication/1734993. [Accessed: 07-Sep-2011].
[81] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "Performance evaluation of the VF graph matching algorithm," pp. 1172-1177.
[82] N. K. Bambha and S. S. Bhattacharyya, "Joint application mapping/interconnect synthesis techniques for embedded chip-scale multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, pp. 99-112, Feb. 2005.
[83] U. Y. Ogras and R. Marculescu, "Energy- and Performance-Driven NoC Communication Architecture Synthesis Using a Decomposition Approach," pp. 352-357.