DOTTORATO DI RICERCA IN
Elettronica, Informatica e delle Telecomunicazioni

CICLO XXIV

Settore concorsuale di afferenza: 09/E3 Elettronica

Settore scientifico disciplinare: ING-INF/01 Elettronica

VARIABILITY-TOLERANT HIGH-RELIABILITY MULTICORE PLATFORMS

Presentata da: FRANCESCO PATERNA

Coordinatore Dottorato:
Chiar. mo Prof. Ing. LUCA BENINI

Relatore:
Chiar. mo Prof. Ing. LUCA BENINI

Esame Finale Anno 2012

Abstract

Next generation electronic devices have to guarantee high performance while being less power-consuming and highly reliable for several application domains ranging from the entertainment to the business. In this context, multicore platforms have proven the most efficient design choice but new challenges have to be faced. The ever-increasing miniaturization of the components produces unexpected variations on technological parameters and wear-out characterized by soft and hard errors. Even though hardware techniques, which lend themselves to be applied at design time, have been studied with the objective to mitigate these effects, they are not sufficient; thus software adaptive techniques are necessary.

In this thesis we focus on multicore task allocation strategies to minimize the energy consumption while meeting performance constraints. We firstly devise a technique based on an Integer Linear Problem formulation which provides the optimal solution but cannot be applied on-line since the algorithm it needs is time-demanding; then we propose a sub-optimal technique based on two steps which can be applied on-line. We demonstrate the effectiveness of the latter solution through an exhaustive comparison against the optimal solution, state-of-the-art policies, and variability-agnostic task allocations by running multimedia applications on the virtual prototype of a next generation industrial multicore platform.

We also face the problem of the performance and lifetime degradation. We firstly focus on embedded multicore platforms and propose an idleness distribution policy that increases core expected lifetimes by duty cycling their activity; then, we investigate the use of micro thermoelectrical coolers in general-purpose multicore processors to control the temperature of the cores at runtime with the objective of meeting lifetime constraints without performance loss.

To my father.

Acknowledgements

I would like to express my deepest gratitude to my advisor, Prof. Luca Benini, for giving me the opportunity to work in his research team, for sharing with me his remarkable knowledge and for his precious lessons.

I would also like to thank:

Prof. Andrea Acquaviva for his support during my PhD studies and for reviewing this thesis.

Prof. Sherief Reda who gave me the possibility to join his team at Brown University and for the accurate work in reviewing this thesis.

Giuseppe Desoli and Francesco Papariello for all I have learnt at ST Microelectronics in Milan.

Eric Flamand and Diego Melpignano for making possible to enlarge my knowledge through my experience at ST Microelectronics in Grenoble.

Prof. Mauro Olivieri for believing in me at the beginning of my research activity.

Prof. Alberto Caprara for his kind and punctual assistance.

All the guys of Micrel Lab for sharing important goals and fun. In particular, Andrea Marongiu for his patience as reader.

All the people I met in Bologna, Milan, Grenoble and Providence for the wonderful moments we shared.

Salvatore and Gianpiera.

My brothers, my sister, and my mother.

Contents

List of Figures

List of Tables

List of Abbreviations

ASIC	Application Specific Integrated Circuit
BFD	Best Fit Decreasing
BP	Bin Packing
CISC	Complex Instruction Set Computer
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
D2D	Die to Die
DC	Direct Current
DRAM	Dynamic Random access memory
DSP	Digital Signal Processor
DVFS	Dynamic voltage and frequency scaling
EDA	Electronic design automation
EM	Electromigration
GOP	Giga operations
GOPS	Giga Operations per Second
GPE	General Purpose processing Element
HCI	Hot Carrier Injection
IDCT	Inverse Discrete Cosine Transform
ILP	Integer Linear Programming
IPC	Instruction Per Cycle
ISA	Instruction Set Architecture
ISS	Instruction Set Simulator
ITRS	International Technology Roadmap for Semiconductors
LP	Linear Programming
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
MPEG	Moving Picture Experts Group
MPSoC	Multi Processor System on Chip
MTTF	Mean Time To Failure
NBTI	Negative Bias Temperature Instability
NoC	Network on Chip
NUMA	Non Uniform Memory Accesses
OS	Operating System
OVP	Open Virtual Platform
PE	Processing Element
QoS	Quality of Service
RAMP	Reliability Aware Micro Processors
RF	Rank Frequency task allocation technique
RISC	Reduced Instruction Set Computer
RN	Rank Energy task allocation technique
RNM	Random task allocation technique
RP	Rank Power task allocation technique
RTL	Register Transfer Level
SIMD	Single Instruction Multiple Data
SM	Stress Migration
SMP	Symmetric Multi Processor
SoC	System on Chip
SPMD	Single Program Multiple Data
TC	Thermal Cycling
TDDB	Time Dependent Dielectric Breakdown
TEC	Thermo Electric Cooler
TLM	Transaction Level Modeling

LIST OF TABLES

VAM	Variability aware modeling	**VP**	Virtual Platform
VLIW	Very Long Instruction Word	**WID**	Within Die
VLSI	Very Large Scale Integration		

Chapter 1

Introduction

1.1 Overview

Fast and complex computations are no longer required only by institutes of research, big companies and banks for manipulating huge amounts of data. Nowadays, personal computers and mobile devices are largely adopted to facilitate the users' life in several application domains, from the entertainment to the business. Thus, to be capable of executing a wide range of sophisticated applications, these systems are increasingly becoming very performance demanding.

For example, complex multimedia applications are now a must for all portable systems, as well as the security of sensitive data, which needs to be always guaranteed in modern, always-connected embedded devices. Such requirements lead to the necessity of executing intensive computation within short times, which translates in high performance demand.

High performance obviously implies high consumption of power. Most of the electronic devices are portable, then they need batteries supporting long usage time. Reducing power consumption is obviously paramount for battery-operated embedded systems, but energy reduction is becoming a critical issue also for personal computers and workstations, as all the world community has become aware of the energy problem in general. Nowadays, all the energy produced is becoming insufficient to meet the demand and increasingly expensive. While the search for alternative sources of energy is underway, engineers must put all their efforts in designing less energy-demanding products.

To further complicate the picture, the miniaturization and high stress of utilization of current CMOS designs, have caused them to become less reliable. In particular the lifetime of any electronic device can no longer be considered infinite and some solutions need to be devised to prevent hard and soft errors. Of course in specific fields such as military, aviation and so on, long lifetimes must be guaranteed, but this is becoming a requirement also for commercial devices. This is because almost all personal data is today stored into digital devices which must be ensured to work efficiently for the longest possible time. Moreover, the big companies are aware of the problem of the impact on the environment, and they are required to adhere to design processes which lead to produce long-lasting and reliable systems.

In conclusions, next generation electronic devices have to guarantee high performance while being less power-consuming and highly reliable. The design processes at the hardware layer take into account these requirements, but even for single-application systems the results are less effective because the workload changes very often over time. For this reason, system-level adaptive techniques, in particular those at the software layer are an unavoidable choice.

1.2 The performance issue

In 1965 Gordon E. Moore described that the density of the components in integrated circuits would have increased over the following years [55]. Since 1975 the number of transistors per area in fact has kept doubling every twenty-two months, thus this allowed the increase of the speed of single-core processors while keeping a stable reliability.

Over the last years the single-core processor speed increase has begun to diminish. This performance increase has always been around 60% until the year 2000, but it dropped to 40% in 2004. The limits of current transistor technology mainly regard the power. The gate of small CMOS transistors gets thinner and less able of blocking the flow of the electrons, then even if transistors are not switching they consume power. Moreover increased clock frequencies cause faster transistor switching, which translates in more heat [31]. The heat increase also produces higher temperatures that in turn lead to reduced reliability [72, 77].

The bottleneck in performance that single-core processors are experiencing is described through the *Pollack's Rule*, which states that performance increase is proportional to the square root of the increase in complexity. Moreover the core power consumption is proportional to the area [12, 65].

Multicore processors have been introduced to keep pace with *Moore's law*. For instance, for the same chip area and power, two small cores can potentially provide the 70-80% performance increase postulated by the Moore's law, against the 40% obtained by a large monolithic core.

Despite Pollack's Rule might suggest pursuing the direction of using a large number of small cores integrated on the same chip, some other bottlenecks in performance exist. To receive high throughputs from multicores, applications need to be parallelized. This process is not always easy. If *N* is the number of cores, the *Amdahl's law* states that the parallel speedup is limited by the serial code in a program, which severely affects the throughput as follows: ***Parallel Speedup = 1/(Serial% +(1-Serial%)/N)*** [37]. In addition, the performance improvement given by a high number of short parallel execution parts could suffer the latency towards the memory and the other devices. In conclusion, different architectures can be adopted such as platforms with a small number of complex cores or platform with a high number of small cores. Hybrid platforms also exist, composed of heterogeneous cores [13].

1.3 The power issue

The power consumption in CMOS circuits is produced by two major contributions. The first one is related to the switching activity of the transistors and it is well-known as *dynamic power* [32] which is proportional to the clock frequency of the core and the total capacitance and the square of the voltage. The other contribution also persists in static conditions due to the *leakage* current which is highly dependent on the threshold voltage [17]. The supply voltage plays an important role for the dynamic power as well as for the performance. The dynamic power can be brought down by lowering the supply voltage while the clock frequency required is still kept. In multicore processors, dynamic voltage and frequency scaling (DVFS) mechanisms can be adopted to optimize the power and set the speed of each singular core to configure energy-efficient systems [66]. Basically low-power processors require low supply voltages [25] but as the CMOS technology scales down to 65nm and beyond, electronic devices work near-threshold region which can cause performance loss, performance variation, and memory and logic failures [27].

Beyond 65-nm dimension, process variations impacting the delay and the power of the circuits have become a critical issue in the design of very large scale integrated (VLSI) circuits using advanced CMOS technologies[64]. A large magnitude of the power variability is due to the exponential relationship between transistor gate length and subthreshold leakage current. On the other hand, performance variability is primarily impacted by physical variations in interconnects. [51, 56, 60]

1.4 The reliability issue

In addition to variations, also the *wear-out* is a factor affecting next generation nanoscale platforms [11, 71]. The impact is not only on power and performance. *Negative bias temperature instability* (NBTI) and dielectric breakdown are critical mechanisms enabling degradation over time that can lead to system failures. Several mechanisms of failure have been classified [70]. The main factors which lead to failures are the usage of the system and the temperature. Typically before incurring a failure a circuit becomes slower [67]. Without applying workload-balancing based techniques, lifetime can be preserved by lowering the execution speed [3].

1.5 Thesis contributions and organization

The aim of this thesis is to demonstrate the effectiveness in performance, power, and lifetime of the software techniques for workload balancing in next generation multicore platforms. We firstly provide an exhaustive background on the multicore processors analyzing the hardware and the software aspects as well as the technological issues, then we present the adaptive techniques we have devised to improve the solutions present in literature.

More in details the structure of the thesis is depicted in Figure1.1. Chapters 2, 3, 4 form the background part, whereas the contribution of this work is presented over Chapters 5, 6, 7, 8. Finally, in Chapter 9 we summarize the results and make some remarks regarding upcoming issues.

In Chapter 2 we provide a classification of the primary types of architecture adopted in multicore processors. We highlight the differences concerning the processing units, the cache memories, and the interconnection systems. Moreover we introduce the concept of virtual platform and what are the needs to adopt such simulators in hardware verification and software development. To conclude the chapter, we present the xSTsim multicore platform by STMicroelectronics that we have largely used in the experimental part of this work.

In Chapter 3 we introduce the issue of the parallel programming needed in multicore processing to reach high performance. We review the most popular paradigms of parallel computing, and then we illustrate how we have parallelized two applications for the xSTsim platform. The first application is the MPEG2 decoder, and the other one is the *integral image* which is a

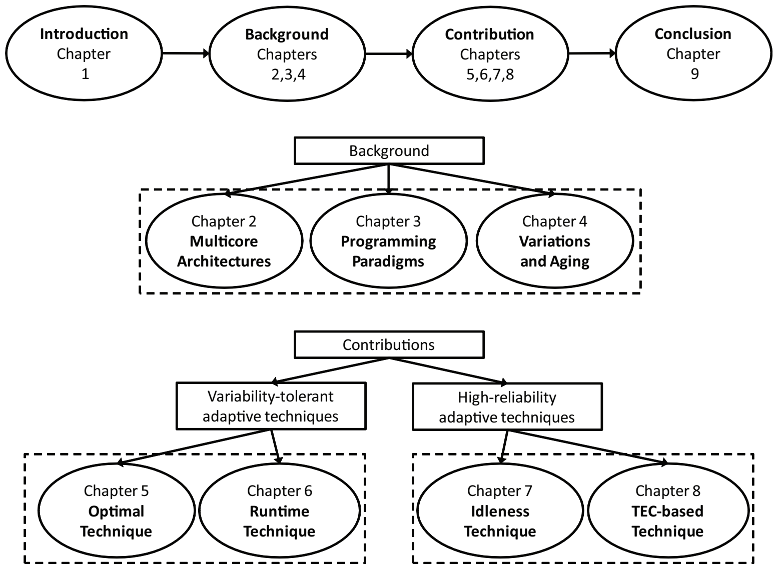


Figure 1.1: Structure of the thesis.

very popular computational kernel used in image detection. These two applications have been used in the experiments.

In Chapter 4 we illustrate the problem of variations and wear-out in sub 45-nm CMOS technology multicore platforms and how much they impact the performance, power, and lifetime of such systems. We illustrate the models and the tools used to estimate those effects. Furthermore, we show how we have linked such tools to the virtual platform of the xSTsim platform. Thus we were able to study and test variation-aware software techniques.

After the background part we illustrate the contributions of this work. The goal we want to reach is to discover adaptive techniques able to meet requirements in performance, power and lifetime. We divide that problem into two separate issues. In Chapters 5, 6, we face the problem of how to minimize the energy consumption while meeting performance constraints in presence of static variations, whereas in Chapters 7, 8 we investigate how to meet lifetime constraints considering also dynamic variations.

In particular, Chapter 5 presents an optimal solution based on an Integer Linear Programming (ILP) formulation of the problem, and a sub-optimal solution based on two-steps namely Linear Programming (LP) and Bin Packing (BP). We have tested the proposed strategies on the virtual cycle-accurate prototype of the target industrial platform comparing the results in terms of energy and performance with the state-of-the-art techniques. Even though we demonstrate the effectiveness of the proposed strategies, they cannot be applied at runtime since they need time-demanding algorithms. In particular to solve the ILP problem we used the ILOG Solver[41] and to solve the LP part of the other technique we adopted the Simplex algorithm.

In Chapter 6 we exploit the key properties of the LP problem of the sub-optimal strategy based on two steps and presented in Chapter 5 to prove that it can be solved through few arithmetic computations and without using the Simplex algorithm. Thus we demonstrate that the LP+BP technique can be solved in linear time and can be definitely applied at runtime. We tested its effectiveness on the virtual cycle-accurate prototype of the target industrial platform running two multimedia applications; the complete MPEG2 decoding, and the *integral image*. We also verify that this strategy outperforms the state-of-the-art solutions.

We face the problem of the performance and lifetime degradation in Chapters 7, 8. In particular we focus on embedded multicore platforms in Chapter 7 and on general-purpose multicore processors in Chapter 8.

In Chapter 7 the objective is to mitigate the impact on lifetime uncertainty and unbalancing among the cores. To this purpose, we developed an idleness distribution policy that increases core expected lifetimes by duty cycling their activity.

In Chapter 8 we investigate the use of micro thermo-electric coolers (TECs) to control the temperatures of the cores and then the lifetime of the system. Based on a real dual-core processor, we first devise a model to estimate the mean-time-to-failure (MTTF) for each core and the entire processor as a function of each core operating conditions, such as power, temperature, and voltage. We then develop a thermal model for the processor and the TECs to capture the thermal and power interactions between the processor, the TECs, and the heat sink. We then propose a number of strategies to use TECs in conjunction with dynamic voltage and frequency scaling (DVFS) to improve reliability and performance.

Chapter 2

Architectures

2.1 From general-purpose processors to embedded system platforms

Multicore architectures can be classified in many ways. We review in the following some of the most representative, considering three popular criteria, namely: the application class, power performance, and processing elements.

The application class. Machines targeted to specific application domains leverage hardware architectures reflecting those specific requirements. This solution ensures the most efficient implementation for the targeted domain, but on the other hand lacks flexibility, thus resulting very weak in executing programs from different application domains. The most notable example in this sense is *application-specific integrated circuit* (ASIC). Tailoring the system design to a specific application domain has many advantages, such as high energy efficiency. *Digital signal processors* (DSPs) are a significant example of this design paradigm.

However this kind of architecture is not recommendable when designing systems that are meant to run varied workload. For example, data-intensive computations such as video and audio processing typically involves executing numerous different signal processing algorithms, for what on-chip multi-core systems are bound to provide better power/performance ratio. The same rationale also applies to control-dominated applications, where file compression/decompression and network processing algorithms may be more efficiently executed over multiple parallel general purpose processing elements. In this manner the unstructured nature of control codes can be better handled.

PowerPerformance. Many devices must execute applications within very strict power budgets and performance requirements. Examples are mobile phones, which are nowadays devised to support video playback while consuming less power and keeping adequate *quality of service* (QoS). Currently, general-purpose multicore processors are the most suitable choice for similar devices, but they are becoming very energy demanding.

Processing elements. Another possible classification can be made based on the type of processing element used as a main building block. Each core has an Instruction Set Architecture (ISA) corresponding to a general-purpose processor plus some few instructions, such as atomic instructions for synchronization to support parallelism. ISAs can be historically classified in reduced instruction set computer (RISC) and complex instruction set computer (CISC). The CISC ISAs provide benefits in code sizes; in fact ISAs for processing elements are extended by the major vendors of processors like Intel and AMD which equip their cores with particular operations for multimedia applications like ARM does for its NEON [47].

The most popular architectures are homogenous and hence all cores have the same design. However, heterogeneous architectures may be more suitable to have same performance with lower power consumption. To increase the performance deep pipelines are typically used because they require a minimal logic per stage; this allows faster clocks and incurs lower penalties from broken execution sequence due to branches.

In-order processing elements are preferable to out-of-order ones because they need smaller area, lower power and are more suitable for high thread-level parallelism. A representative example in this sense is general-purpose graphical processing units (GPGPUs): The, NVIDIA's G200 has 240 in-order cores [21]. Out-of-order processors work better to improve performance of serial codes because instructions can be dynamically (re)scheduled to keep full the pipeline, but the related mechanisms are very power hungry. When the application domains are characterized by poor degrees of thread-level parallelism, out-of-order cores are preferable, particularly when implementations adhering to either the *single-instruction multiple-data* (SIMD) or the *very long instruction word* (VLIW) architecture paradigm are considered. SIMD and VLIW eliminate some complexity of the logic to optimize the execution of instruction streams. SIMDs use large split registers to process multiple data requiring only one instruction. An example of this architectural solution is vector instructions. An example of multi SIMD-core processors is the IBM Cell [50] which is targeted for data-intensive applications. Instead VLIWs use multiple pipelines to execute groups of instructions in parallel. Most of the complexity in this case is moved at the compiler level. VLIWs can provide some advantages over SIMDs but exhibit poor performance if under-utilized. Both VLIW and SIMD are the most efficient solution, performance- and energy-wise, for applications with high levels of data-parallelism.

Memory hierarchy. From the point of view of the memory hierarchy, caches have become more and more important in multicore processors, because they provide each core with local, fast memory. Caches can be further tagged and managed by the hardware or explicitly used as local store memory (i.e. scratchpads).

Tagged caches are very common because they handled transparently by the instruction stream which believes only one uniform memory exists. Scratchpads can instead provide deterministic performance and offer more storage capacity for the same area.

The number of cache levels increases as processing elements become faster and numerous in platforms [9]. Typically, as the cache gets close to the main memory it becomes larger, slower, and it is shared among the cores. Thanks to memory hierarchies, processing elements perform very fast accesses even though the main memory is hundreds of cycles away. For instance, AMD Phenom [42] has three levels of cache. In embedded multicores the main memory may be a few tens of cycles away and one level of cache may be sufficient to conserving both die area and power [9].

Some multicore platforms integrate an embedded DRAM (eDRAM) bank on the same die to improve system performance by reducing the memory latency. Multi-bank DRAMs are adopted to hide long latencies by allowing the memory to process multiple accesses in parallel. This may incur a significant area penalty and will therefore restrict the density of the embedded DRAM main memory [84].

Tightly-coupled, multi-banked shared memories are adopted in embedded multicore platforms as P2012 [53] from STMicroelectronics and the Hypercore architecture line (HAL) [48] from Plurality, both of them contain sets of RISC-32 bit Harvard processors.

Shared memory-based systems often leverage coherent cache systems to ease application writing. Cache coherence can be broadcast-based or directory-based. In the broadcast way operations visible to all processor can be executed from only one core at time. The directory instead enables multiple coherence actions to occur concurrently. This is possible because a directory contains information about which caches contain each memory address.

Directory coherence is typically adopted for weak consistency models and for large systems containing many cores, such as the Tilera TILE64 [22].

Sometimes cache coherence is omitted to reduce design complexity, examples are the TI TMS320DM6467 [76] and the IBM Cell [50]. In this case the software has to enforce the visibility of the memory among the cores at runtime. This limits the programming models to custom variants of message passing. This can be feasible when the size of the memory is limited [9].

In multicore processors the processing elements communicate each other via intra-chip interconnects. The different types of interconnects can be classified in buses, crossbars, rings, and NoCs. The bus requires simple logic but it suffers from limited bandwidth when the number of processing elements increases. Instead the NoC scales very well but many challenges rise up at the design phase.

Interconnects are also responsible for the cache coherence which determines the programming models supported by the overall architecture. For programming models based on shared memory, cache coherence keeps a unique image of the memory visible to all cores in the system. ARM Cortex A9 supports this view [47].

2.2 Virtual platforms

Due to the increase of the Multiprocessor System-on-chips (MPSoCs) complexity and to tight time-to-market requirements, the hardware and the software parts of the system need to be designed simultaneously.

Software engineers have to develop operating systems, device drivers, and protocols of communications on the hardware prototype while hardware engineers are designing the platform at register-transfer level (RTL).

The overall software part cost of MPSoCs is quickly increasing. The International Technology Roadmap for Semiconductors (ITRS) have predicted that software will cost as much as the hardware by 2013.

Prototypes of the target applications, namely virtual platforms (VPs), are adopted. Thanks to virtual platforms software development and hardware validation can be largely facilitate.

A simplified flow of concurrent hardware/software design starts from a system-level functional specification of the overall system. In the next step several functions are identified and mapped on either hardware or software blocks. Then the hardware team develops the RTL specification of the hardware components while the software team starts to work on the virtual platform. As the hardware progresses the virtual platform is updated and provided to the software development team. In this way, the hardware and software processes can progress together in lockstep.

Virtual platforms are developed at various level of abstraction. For each one of these levels a certain degree of accuracy is possible. Typically high levels of abstractions are mostly useful for the software development while more accurate models are needed for hardware verification. Virtual platforms are also characterized by the speed of simulation. At high level of abstraction virtual platforms are faster and less accurate.

VPs can be developed though several system (hardware and software) description languages that in most cases match programming languages such as C, C++, Matlab or are extensions of those, such as SystemC. Each language better lends itself to one or few specific levels of abstraction. Figure 2.1 shows several levels of abstraction indicating the system description languages associated. For each level the graph also indicates the degree of both accuracy and simulation speed.

Several emerging standards exist to develop virtual platforms of MPSoCs.

SystemC is a C++ library that provides the concepts of concurrency, bit-accuracy and timing required in chip design to the C++-based programming. [1].

Transaction-level modeling (TLM) is an interface modeling methodology. TLM models complex system-on-chip using instruction-set simulators (ISSs) of processors and high-level, fully functional SystemC/C++ models of the other hardware building blocks.

Many electronic design automation (EDA) vendors are producing tools to develop virtual platforms, such as Synopsys and Carbon Design Systems.

System-on-Chip design companies widely exploit virtual platforms and provide simulator of their own IPs to be easily imported in virtual platform of complex systems designed by third

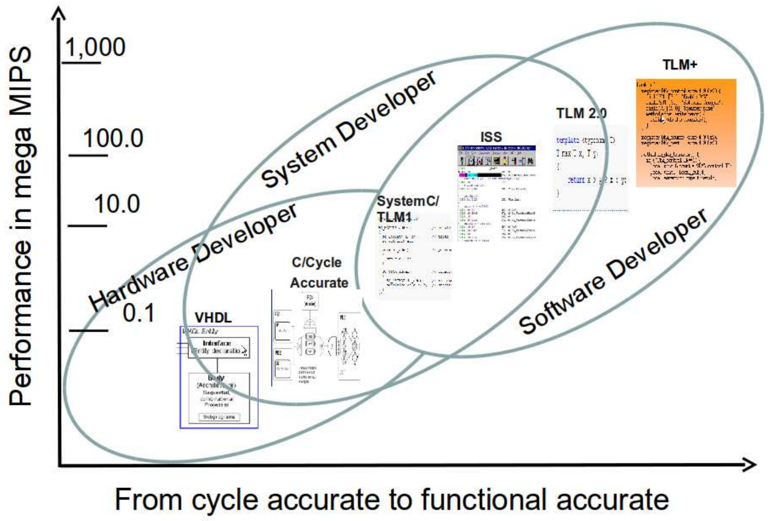


Figure 2.1: Abstraction levels of a system. Comparison between accuracy and simulation speed.

parts. For example, ARM provides a large set of fast models of several ARM processors[79]. Tensilica proposes a simulation environment including XTENSA processors, memories, and connectors that can run as a SystemC model or as a C/C++ model[80]. ST Microelectronics is a pioneer of SystemC and of the TLM-2 standard, and its design teams use SystemC models for both software design and hardware verification. [59]

2.3 A multicore platform model

In this section we illustrate the xSTsim multicore platforms by STMicroelectronics and the related VP which will be largely used in the experiments of this work.

The platform is composed of a general purpose processing element (GPE, in particular it is an ST231 [28]) acting as *host processor* and a number of programmable accelerators, acting as *streaming engine* (or fabric), as shown in Figure 2.2. The processing elements (xPEs) of the streaming fabric are connected through a Network-on-Chip supporting very high data bandwidth and throughput. The platform is meant to address the needs of data-flow dominated, highly computational intensive tasks, typical of many embedded systems. This platform model adheres to the STMicroelectronics xSTsim architecture. For the applications we target in this work the GPE acts as a task dispatcher for the xPEs.

The xPEs of the streaming fabric are relatively simple programmable processors with a simple ISA extended with SIMD and vector mode instructions. The engines include a set of features for improving performance and efficiency, such as wide data-paths, simple pipelines, multi-threading etc. At the same time they execute instruction fetches from local memories instead of caches, a great simplification at the pipeline forefront. Local memory is also used for wide data accesses.

Each xPE can be *frozen* and restarted writing to a control register that can be accessed by the GPE. The system has a global memory containing the program (typically the operating system, OS) running on the GPE and its data. Any xPE can access the global memory and each processor can access the local memory of another xPE, even though with a significant cost in terms of latency. Hardware based memory coherency is not needed because of the lack of caches for the xPEs, and cache coherency is explicitly maintained in software with the GPE. The GPE and the global memory are connected through a shared bus which is one node of the NoC interconnect.

A C-based model of the xSTsim platform has been developed by STMicroelectronics. This virtual platform can be configured at the beginning of the simulation through a configuration file. It is possible to specify the number of the accelerators, the type of the interconnection such as bus, crossbar, NoC, and the level of accuracy of each block. The ISSs of both the ST231 and the XPE processor are cycle-accurate. Despite the xSTsim simulator allows to reach very high levels of accuracy it is highly efficient in terms of performance.

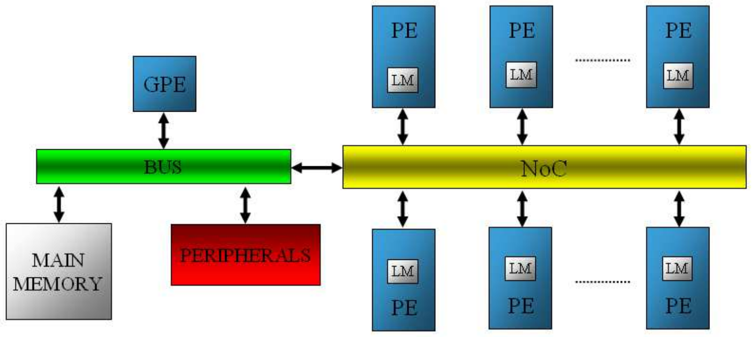


Figure 2.2: xSTsim platform model.

Chapter 3

Parallel programming on multicore processors

3.1 Overview

Smart mechanisms for dynamic, network-wide resource sharing have enabled the creation of clusters of processors to be used in large-scale computing systems, achieving high performance and scalability.

In recent computers and workstations, parallelism appears both in hardware and software at various layers: signal, circuit, component, and system levels. At signal and circuit levels, parallelism is performed using hardware parallelism. At a slightly higher level, better performance is obtained by exploiting multiple functional units operating in parallel. This level of parallelism is well-known as instruction level parallelism. At a still higher level, symmetric multi processors (SMPs) have multiple CPUs working in parallel. At an even higher level of parallelism, several computers can be connected together and work as a single machine, namely cluster computing. Parallelism at component and systems levels is mostly possible by exploiting various software techniques, popularly known as software parallelism.

Software parallelism can be identified and outlined at different granularities in the application code. These granularities determine different kinds of parallelism. In particular, the parallelism can be extracted automatically in hardware, or through software techniques at various levels: (semi-)automatically in the compiler or manually in the application code. Table 3.1

classifies the kinds of parallelism on a code grain size basis.

Grain size	Code item	Parallelized by
Fine	Instruction	Processor
Medium	Loop or instruction block	Compiler
Large	Threads and processes	Programmer

Table 3.1: Levels of parallelism and grain code size.

All the approaches to parallelization have the common goal to boost processor efficiency. Possibilities to parallelize the code of an application can be detected at the several levels, as depicted in Figure 3.1. Starting from the application it is possible to find some functionalities that can be split in tasks, or processes that can be run in parallel; this is the coarse grain level. Each task can be further composed by functions that can be run in parallel; this is a medium grain level. More deeply, each function can be characterized by sequences of equal operations that work on different data; thus those operations can run in parallel too actually performing the instruction-level parallelism, this is a fine grain level. Some processor microarchitectures are characterized by different functional blocks that allow executing in parallel different kind of operations; this is the very-fine grain level [18].

Among the four identified levels of parallelism, the very-fine and the fine grain level are supported transparently either by the hardware or parallelizing compilers, while programmers mostly deal with the large and the medium levels.

Parallel programs exploit concurrently running threads or processes, and support for inter-thread communication is needed. The two primarily models of communication are the shared memory and the message-passing.

In the shared-memory paradigm, processes communicate using references to shared data which typically are stored in a global memory visible to all cores. The accesses towards the global memory are asynchronous. This requires protection mechanisms such as locks and semaphores. The shared memory model can be emulated on distributed-memory systems but non-uniform memory accesses (NUMA) can degrade the performance.

In a message-passing communication model, processes communicate using messages. There is no common address space for data, but each process accesses its own dedicate address space which may correspond to a private local memory. These communications can be asynchronous or synchronous [45].

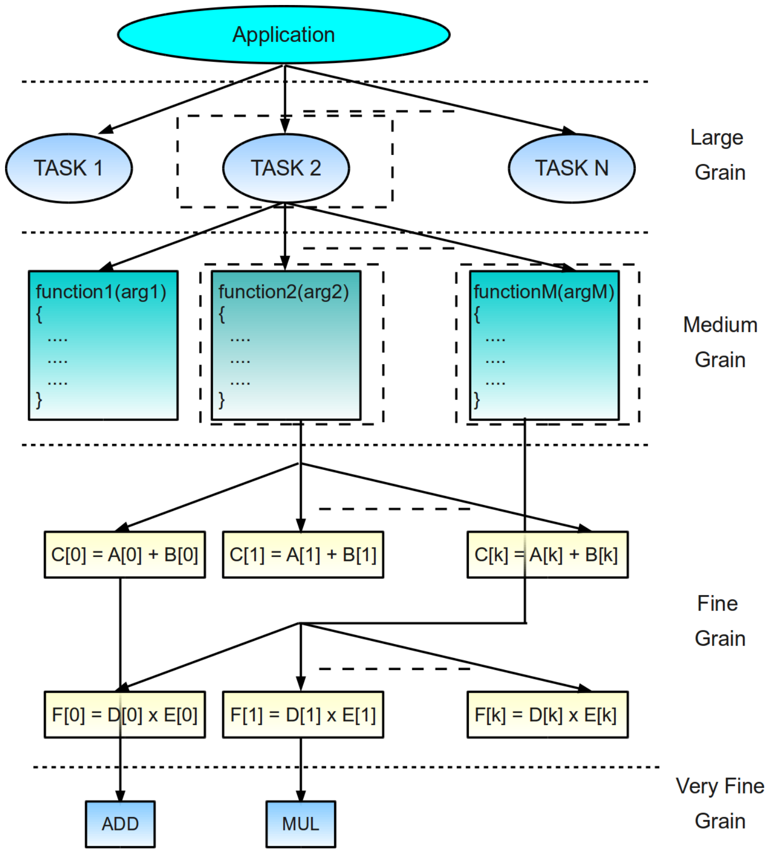


Figure 3.1: Levels of paralellism

3.2 Parallel programming paradigms

Parallel programming techniques can be classified in few paradigms that are used repeatedly to develop many parallel programs. A paradigm is identified by a class of algorithms which have the same control structure. The choice of a paradigm strongly depends on the parallel computing resources and the application. In particular the resources identify the granularity level at which the parallelism can be more efficient whereas the structure of the application or the data determines the type of the parallelism. There is a functional parallelism when it is possible to extrapolate from the application different tasks that can be executed concurrently and in a co-operative way. Data parallelism exists when it is easy to identify identical processes that can be executed in parallel but on different data. In literature many different classifications of parallel programming paradigms exist [30, 33]. A classification based on process properties, interaction process, and data properties can be found in [45]. We review three of the most important parallel programming paradigms which are named respectively: Master/Slave, Single-Program Multiple-Data, and Data Pipelining.

Master/Slave

The Master/Slave or Task-farming paradigm is characterized by a master entity and multiple slave entities. The master has to organize the problem into small tasks and then distribute them to the slaves. When the slaves terminate their works, the master has also to collect the results. Figure 3.2a) shows the Master/Slave diagram. The workload-balancing across the slaves can be static or dynamic. In the first case, all tasks are assigned to the slave processes at the beginning of the computation. The allocation can be done at compile-time or at runtime. In the second case it is possible to map the tasks on the slave cores dynamically and one-by-one basis. This mechanism can be applied only at runtime. The dynamic load-balancing is more suitable when the number of tasks either exceeds the number of the cores or it is unknown. Moreover, it can be very useful to adapt the workload to the conditions of the systems by giving the possibility to optimize the execution in terms of performance as well as power and reliability.

Single-Program Multiple-Data (SPMD)

One of the most popular paradigms is the SPMD. Basically the application has to be decomposed in processes having the same piece of code which works on different data. Figure 3.2b) illustrates the SPMD diagram. This paradigm is highly recommended when it is possible to recognize geometric structures and data-independent computation in the applications. Processes firstly access to their own data and then work simultaneously. A barrier of synchronization is typically used between different computations.

Data Pipelining

This paradigm suggests identifying sequences of separate functions in the applications and assigning each one of them to a process. In this manner a sequence of computation stages is created. In general, each stage produces a data which will be the input of the next stage as depicted in Figure 3.2c). A system for communicating across the stages is needed; thus it determines the robustness of the paradigm. The communication may be completely asynchronous; this means that mechanisms like barriers are not needed. The effectiveness of this paradigm depends on the possibility to well balance the workload across the stages. Data Pipelining is often used in data reduction or image processing applications.

Sometimes the paradigms can show fuzzy boundaries. In addition, for some complex structures high levels of parallelism could not be reached by using only one paradigm. Typically applications are parallelized exploiting mixes of paradigms [18].

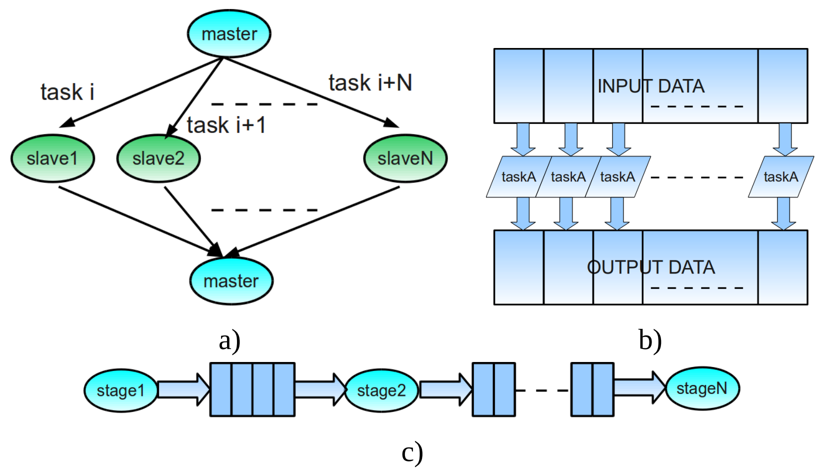


Figure 3.2: Parallel programming paradigms: a) Master Slave; b) SPMD; c) Data Pipeling.

3.3 Case study

In this section we describe two examples of parallel applications for multicore platforms. The first example is a simple computational kernel very common in many algorithms from the computer vision domain; the second is a complete MPEG2 decoder.

3.3.1 Integral image

The integral image algorithm is becoming popular in many image processing applications. In particular it is used for feature evaluation in the face detection problem [8].

This algorithm is applied on an image characterized through a pixel matrix. Let x be the pixel row identifier and let y be the pixel column identifier, the integral image consists in formula (3.1).

$$II(x, y) = \sum_{x'<x, y'<y} I(x', y') \tag{3.1}$$

It can be easily parallelized according to the SPMD paradigm. We divide the computation in two steps. In the first step, for each row r we replace the value at element i with the sum of its current value plus the value of element $i-1$. Let r be composed by N elements, from 0 to $N-1$. We start from $i=1$ until $i=N-1$, while for element 0 we do not replace its value. We can execute the computations regarding each row in parallel. The second step is similar to the first one, but the computation involves the elements of the columns. Also in this case we can proceed by parallel computations.

For instance if we have a matrix of 96 $\times$ 96 elements and we want to divide each step in 8 parallel tasks, for the first step we assign the computation of the first 12 rows at the first task, the computation of the second 12 rows at the second task, and so on. For the second step we act in the same way, then we assign at each task the computation of 12 contiguous columns.

For this application, to evaluate how much the performance increases with the number of the parallel tasks, we have executed the following experiments on the multicore platform simulator illustrated in Section 2.3. We have analyzed the time needed by the execution of the integral image described through Formula (3.1) by varying the image size and the number of the parallel tasks. On each xPE accelerator of the platform we have allocated only one task.

In Figure 3.3, we show how much the execution time scales down with the increase of the number of the used xPE accelerators. The figure plots the execution times for different image sizes. For each of these sizes, we have normalized the execution times over the longest one.

The execution time is roughly halved by passing from 2^x accelerators to 2^{x+1}, for any x. For the smallest image size, beyond 4 cores the latency toward the memory hides the benefits that the parallel execution provides, and then the execution time does not further scale.

3.3.2 Parallel MPEG2 decoder

We started working on a MPEG2 decoder [34, 54] originally written for the ST231 multi-threaded processor [28]. This program was designed to run on 1, 2, or 4 threads dividing each frame in two vertical halves or four quadrants. The aim of the effort was to transform the code into a realistic benchmark for a class of parallel multimedia codec suitable of being deployed on massively parallel embedded multiprocessor arrays. To do this we had to restructure the application to remove bottlenecks stemming from Amdahl law limitation to available parallelism when the number of concurrent threads is increased.

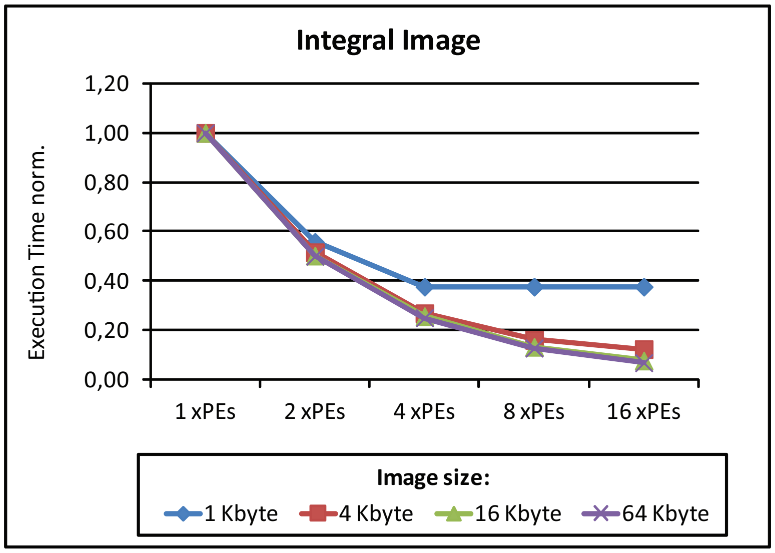


Figure 3.3: Integral Image. Execution time over the number of core accelerators at different image sizes.

The task graph it is depicted in Figure 3.4 and it is composed of three parts: a control part which scans the current frame, a slice decoding, and an Inverse Discrete Cosine Transform (IDCT). There is also a fourth step, performed after the decoding of each frame, associated with the commit of results.

We modified the program so that the scan of the current frame is performed by the host core, the slice decoding and the IDCT can be parallelized and executed on a generic number of accelerators, and the commit of results is performed by the host core. The slice decoding and the IDCT have been divided in independent tasks whose number can be equal or greater than the number of accelerators. Regarding the latter case, a dispatcher has been implemented on the host core to schedule the different tasks on the accelerators. To increase the performance we further modified the code to execute the commit of the previous frame during the execution of the current frame on the accelerators.

This example combines all the three parallel programming paradigms presented in Section 3.2.

We have conducted experiments on the MPEG2 decoder presented in this section to evaluate the benefits that the parallelism provides. As input we have used a videoclip characterized by frame ratio 25 frame per second (fps), length 1 second, resolution 720$\times$576. We have still

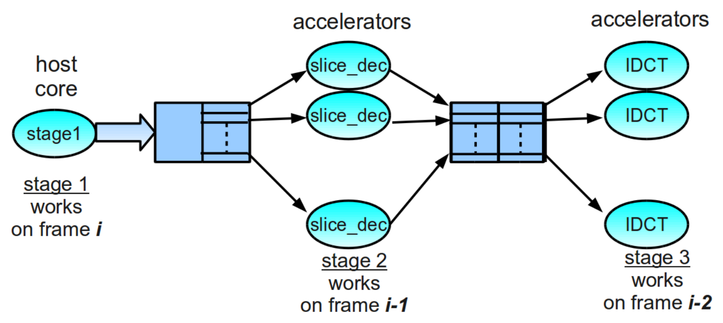


Figure 3.4: Task graph of a parallel characterization of the MPEG2 decoder.

used the simulator presented in Section 2.3. In each simulation, the platform has decoded 25 frames, and we have measured the execution time. We have divided the workload in 2, 4, and 8 tasks and allocated on each accelerator only one task.

Figure 3.5 shows how much the execution time scales down over the number of the used xPE accelerators / parallel tasks. The execution time is roughly reduced by 40% passing from 2^x accelerators to 2^{x+1}, for any x.

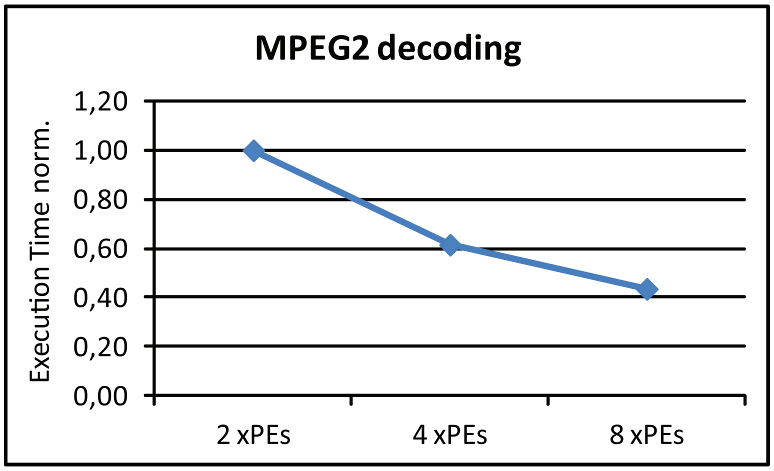


Figure 3.5: MPEG2 Decoder. Execution time over the number of core accelerators.

Chapter 4

Process variation and aging of CMOS architectures

4.1 Impact of static variations

Multicore architectures will be adopted in the sub-45nm CMOS technology nodes for virtually all application domains with energy efficiency requirements exceeding 10GOPS/ Watt. Unfortunately, future technology nodes will be increasingly affected by variation phenomena, and multicore architectures will be impacted in many ways by the variability of the underlying silicon fabrics [29, 77].

The main causes which produce process variability in these technologies are imperfection in lithographic patterning of small devices and random doping effects [73] especially for multicore systems [15, 40, 74].

The causes of process variations are classified in relation to two kinds of effects; die-to-die (D2D) or within-die (WID). This means that in multicore processors if we compare two chips of the same model, we can experience differences in speed and power between the two chips and also among the cores of each chip. Whereas D2D is mainly caused by atomic-scale oxide thickness variations and also dielectric thickness variations, two components are handled to model WID variations. One component is systematic and the other one is random. Systematic variations show a spatial correlation; this means that nearby transistors exhibit similar parameter values. On the other hand, random variations are mostly induced by materials effects and show different profiles across the transistors [7, 77].

However, also single core platforms are strongly impacted by variability. In superscalar processors, variability causes non-uniform performance among the various units, so that the clock frequency must be set to accommodate the slowest unit, thus degrading the overall throughput. An alternative is to set the clock based on the fastest units and leave more cycles to the slowest. The instructions are then scheduled in the functional units to maximize the throughput [57].

In multicore processor, intra-die process variations result in significant core-to-core frequency variations [19, 36]. More in detail, critical paths can be faster or slower than nominal and the clock frequency of each accelerator needs post-fabrication calibration.

In addition to the performance, variability also impacts the power consumption and since intra-die variations cause a non-uniform behavior of the components across the chip surface, multicore platforms become heterogeneous both from a performance and energy viewpoint [63]. Large variations are measured for the leakage because of the exponentially dependency from the threshold voltage [2]. In conclusion, beyond 90-nm CMOS technology process variability can affect dies leading to 30% in delay and 20 $\times$ in leakage [14].

Furthermore, temperature dependencies and wear-out add dynamic variations on top of static inter-die process variability [77].

4.2 Performance degradations and reliability limitations

Multicore architectures on next generations are also experiencing effects due to aging and failure processes. These effects cause dynamic variations and can be orthogonally treated with respect to the variability which primarily leads to static variations.

Elevated power densities and practical limitations on heat removal have led to high junction temperatures in modern computing processors. These elevated temperatures limit the performance and reduce the reliability of computing systems.

In particular, progressive slowdown in processors is induced by Negative Bias Temperature Instability (NBTI) and Hot-Carrier Injection (HCI) [7] and several other mechanisms, which are strongly dependent on temperature, cause chip failure mechanisms [70].

At the system level, NBTI and HCI produce a gradually slowdown of the transistors switching, and hence slower critical paths. This roughly is due to the stress of the transistors that causes a continuous movement of charges.

4.2.1 NBTI characterization for multicore platforms

NBTI affects PMOS transistors causing shifts in threshold voltages with relation to operating conditions [4]. One of the most qualified models of NBTI is characterized by two phases. The *stress* phase happens when the logic input 0 is applied to the gate of a PMOS transistor. The *recovery* phase happens when the logic input 1 is applied. We used the mathematical model described by Tiwary *et al.* [77].

Let's denote the stress time as t_{stress} and the recovery time as t_{rec} , we have the threshold voltage increment (ΔV_{t_stress}) during the stress phase modeled as in (4.1). The total degradation (ΔV_t) that further takes into account the recovery time is given by the (4.2).

$$\Delta V_{t\_stress} = A_{NBTI} \times t_{ox} \times \sqrt{C_{ox}(V_{dd} - V_t)} \times e^{(\frac{V_{dd}-V_t}{t_{ox}E_0} - \frac{E_a}{kT})} \times t_{stress}^{0.25} \tag{4.1}$$

$$\Delta V_t = \Delta V_{t\_stress} \times (1 - \sqrt{\eta \times t_{rec}/(t_{stress} + t_{rec})}) \tag{4.2}$$

We set the following parameters as descripted in [77]: $t_{ox} = (0.65nm)$ (oxide thickness) , $C_{ox} = 4.6 \times 10^{-20} F/nm^2$ (gate capacitance per unit area), $E_0 = 0.2V/nm$, $E_a = 0.13eV$, $k = 8.6174 \times 10^{-5} eV/K$, $\eta = 0.35$ (constants). The parameter A_{NBTI} is a constant depending on the aging rate.

The delay of a transistor in relation with V_t is expressed by (4.3), where $\alpha \approx 1.3$ ([77]). Setting $V_{dd} = 1.10V$, $V_t = 0.5V$, $Lf = 5.24^{-10}$ we have a delay of $T_s = 1.12^{-9} sec$. This determines the maximum support clock frequency of a core, thus we calculate it as $f_{ck\_max} = 1/T_s \approx 893MHz$.

$$T_s = \frac{L_f V_{dd}}{(V_{dd} - Vt)^{\alpha}} \tag{4.3}$$

We can now define the *guardband* of a core as the relative difference between the working clock frequency $f_{ck}$and the maximum one (4.4).

$$GB = \frac{(f_{ck\_max} - f_{ck})}{f_{ck}}; \tag{4.4}$$

Of course to be working a core must have a positive value for GB. Once we know the temperature and the constant A_{NBTI} we can estimate the guardband. We assume a temperature constant at $330K$ and set $A_{NBTI} = 15.2^6$. We can now estimate the lifetime in terms of years assuming to be able to impose a fixed recovery / stress ratio over the time.

Let's assume to consider acceptable a core whose guardband is larger than 1%.

Figure 4.1 shows for a generic value of idle / activity ratio (X axis) the guardband (Y axis) after a certain number of years (curves), if the value is found above the dashed horizontal line which indicates the GB = 0.01, that number of year is guaranteed.

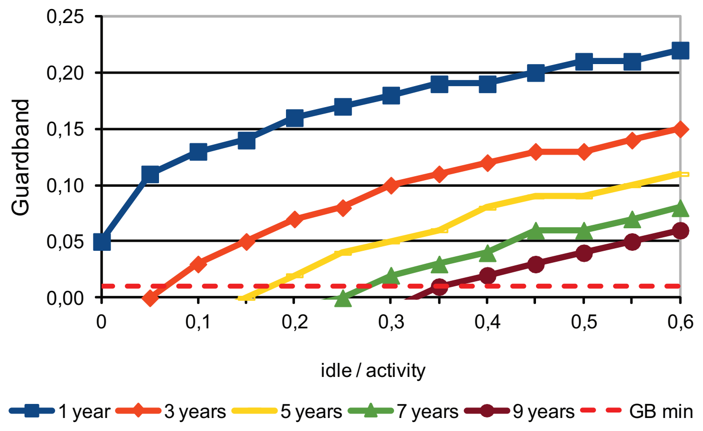


Figure 4.1: Per-core guardband analisys over recovery ratio.

For example if idle / activity = 0.1 the core will work for 3 years, but not for 5 years. Again, imposing a ratio of 0.4 the system will work until 10 years.

The lifetime is intended as the sum of the time spent in idle and the time spent in activity.

Figure 4.2 shows the maximum total lifetime, total activity time, and total recovery time in year (Y axis) in relation with the ratio (X axis). The area below the total lifetime curve gives all the guaranteed working years for each idle ratio imposed.

4.2.2 Mechanisms of Failure

Another approach to study aging and wear-out is to find a relationship between the mechanism of failures and the lifetime expressed in number of years. This information can be

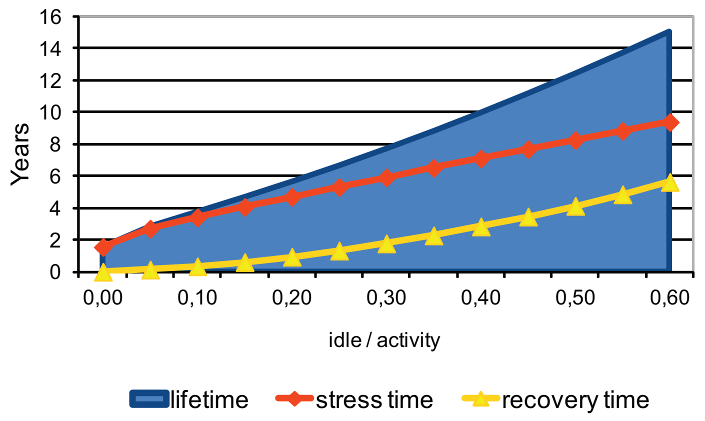


Figure 4.2: System lifetime analysis.

obtained through a first characterization which lends itself to estimate the Minimum-time-to-failure (MTTF) of the chip structures.

Device and interconnect failures can occur in any structure of the processor die [16, 52, 70]. Failures can be classified into five critical mechanisms:

1. **Electromigration (EM).** EM occurs when conductor metal atoms are being transported within the processor interconnect. The MTTF related to this mechanism decreases with the current density, then with the power, and with the temperature in an exponential way. The model is given by

$$MTTF_{EM} \propto (J)^{-n} e^{\frac{E_{aEM}}{kT}}, \tag{4.5}$$

where J is the current density in the interconnect, k is the Boltzmann's constant, T is the absolute temperature in Kelvin. $n = 1.1$ and $E_{aEM} = 0.9$ are constants that depend on the interconnect material.

2. **Stress Migration (SM).** SM is due to the migration of metal atoms in the interconnects caused by mechanical stress. The MTTF decreases on the temperature in a non-linear way as given by

$$MTTF_{SM} \propto |T_0 - T|^{-n} e^{\frac{E_{aSM}}{kT}}, \tag{4.6}$$

where $E_{aSM} = 0.9$ is a constant that depends on the interconnect material, and T_0 is the metal deposition temperature (typically 500 K).

3. **Time-Dependent Dielectric Breakdown (TDDB).** TDDB is generated by the gate dielectric's gradual wear out leading to transistor failure. The MTTF of this mechanism is affected by the temperature and the voltage as given by

$$MTTF_{TDDB} \propto (\frac{1}{V})^{a-bT} e^{\frac{|X+(Y/T)+ZT|}{kT}}, \tag{4.7}$$

where V is the operating voltage, a, b, Z, Y, and Z are all fitting parameters. Authors in [72] assume the following values: $a = 78, b = 0.081, X = 0.759eV, Y = -66.8eVK, Z = -8.37 \times 10^{-4}eV/K$.

4. **Thermal Cycling (TC).** TC in processors can be caused by different phenomena like variations in power consumption or workloads. TC can lead to failure. The MTTF depends on the temperature as given by

$$MTTF_{TC} \propto (\frac{1}{T - T_{amb}})^q, \tag{4.8}$$

where $q = 2.35$ is the Coffin-Manson exponent, and T_{amb} is the ambient temperature.

5. **Negative Bias Temperature Instability (NBTI).** NBTI affects the P-channels of MOSFET transistors. This mechanism generates a threshold voltage increase which can lead to timing violations and failures. NBTI is given by

$$\begin{aligned} MTTF_{NBTI} \quad \propto \quad & \{[\ln(\frac{A}{1+2e^{B/kT}}) + \\ & - \ln(\frac{A}{1+2e^{B/kT}} - C)] \times \frac{T}{e^{-D/kT}}\}^{1/\beta}, \end{aligned} \tag{4.9}$$

where A, B, C, D, and β are all fitting parameters with the following values $A = 1.6328$, $B = 0.07377$, $C = 0.01$, $D = 0.06852$, $\beta = 0.3$.

The parameters and the constants of the models illustrated above are here explained. We report the parameter values adopted in [72]. J is the current density in the interconnect, $n = 1.1$, $E_{aEM} = 0.9$, $m = 2.5$, $E_{aSM} = 0.9$ are constants depending on the interconnect metal used (copper is assumed), k is the Boltzmann's constant, T is the absolute temperature in Kelvin, T_{amb} is the ambient temperature in Kelvin, $T_0 = 500K$ is the metal's stress-free temperature, $a = 78, b = 0.081, X = 0.759eV, Y = -66.8eVK, Z = -8.37 \times 10^{-4}eV/K$ are fitting parameters, $q = 2.35$ is the Coffin-Manson exponent, $A = 1.6328, B = 0.07377, C = 0.01, D = 0.06852, \beta = 0.3$ are fitting parameters.

4.3 Tools

4.3.1 Variability Aware Modeling (VAM)

VAM is a tool presented by IMEC in 2007 to percolate process variability and reliability information from the electrical device model level to the system level. It reuses the same abstraction interfaces as currently found in existing digital design flows but augmented with additional information for representing the statistical influence of such process variability and technology reliability effects. Through a Monte-Carlo approach, it achieves sufficient statistical relevance using a limited number of simulations. Furthermore VAM describes in detail the process to predict system yield from technology variability, and apply this to a concrete system [58, 78].

It can be used to emulate degraded multicore platforms. VAM starts from its gate-level netlist and can generate some instances of the system that are characterized by the values of leakage power, dynamic power and delay of each core. In this way D2D and WID variations can be modeled.

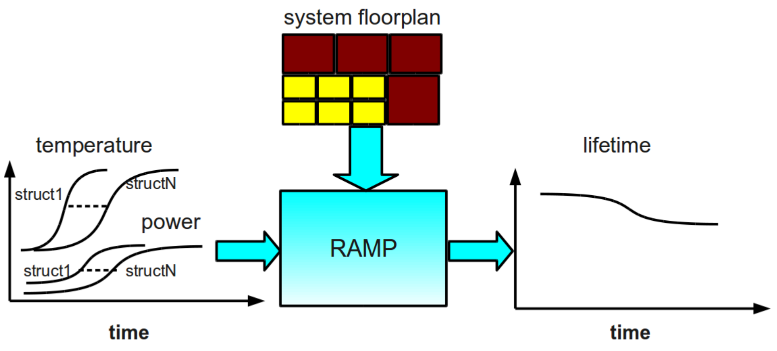


Figure 4.3: Diagram for lifetime estimation using the RAMP tool.

4.3.2 Reliability-aware Micro-processors (RAMP)

RAMP is a tool developed by the University of Illinois to emulate the mechanisms of failure described in Section 4.2.2. It can be very useful at design time and for devising adaptive techniques for lifetime preservation. Basically it uses two inputs. First of all, a floorplan describing the topology specifying all the structures of the target system has to be provided to the tool. The other input regards a trace file containing the temporal information about temperature and power of each structure. In a first step RAMP calculates the MTTFs of each structure for all mechanisms by exploiting the models presented in Section 4.2.2. In a second step through a Montecarlo simulation it estimates the lifetime of the system. In particular the tool can be used to calculate the lifetime over the time by cumulating for each instant the information trace from the beginning to the actual time. This means that it could be also exploited at runtime to dynamically preserve the lifetime, for example by adapting the workload among the cores of an MPSoCs. Figure 4.3 shows the diagram of RAMP.

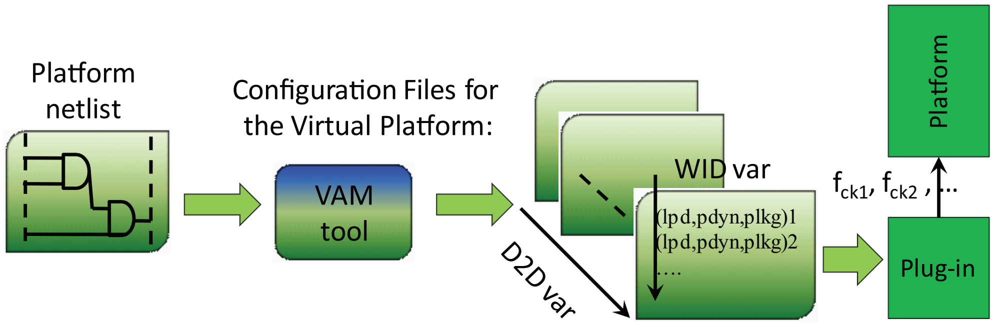


Figure 4.4: Modeling variability in Virtual Platforms.

4.4 Integratation of tools into virtual platforms

We integrated variability and aging models into the xSTsim virtual platform presented in Section 2.3 by building a plug-in which uses the simulator API functions to have access to the simulator structures and functionalities. The idea is to simulate hardware monitors present in modern multicore processors to expose at the software layer the information about the power, the speed, and the lifetime degradation of the cores. In this way the runtime can modulate the workload among the cores to meet given constrains on performance, energy and lifetime. In particular monitors are simulated by memory-mapped registers for each core.

The plug-in provides the following features:

a) it differentiates the cores in relation with their parameters;

b) it scales the clock frequency of each core according to its longest path delay;

c) it stores the cycles spent in the different states of each core;

d) since it knows the core parameters and the stored cycles, it evaluates the energy consumption.

The plug-in needs to be configured at the beginning of the simulation through a text file named *configuration file*, that specifies the core parameters.

To emulated static variations among the cores in terms of leakage power, dynamic power, and longest path delay we used VAM which works as back-end of our plug-in; Figure 4.4 depicts this mechanism. From the netlist of the platform VAM generates the values of leakage power, dynamic power, and longest path delay of each core. Those values are written into the configuration file of the simulator. The plug-in reads this information and exposes them to the software layer. Furthermore, it can automatically change the frequency of each core in according to its own longest path delay. In this way we are able to emulate both WID and D2D variations.

To emulate the aging we further implement into our plug-in the model shown in Section 4.2.1.

Chapter 5

Variability-tolerant multicore platforms

5.1 Overview

In this chapter we study multicore platforms whose accelerators are nominally homogeneous, but unfortunately variability causes significant perturbations on their performance and power consumption. More in detail, critical paths can be faster or slower than nominal and the clock frequency of each accelerator needs post-fabrication calibration. Faster cores are overclocked and slower cores are clocked at a lower frequency. Frequency adjustments are supported by the platform, but the accelerators do not have independently controllable power supply voltages for system and die cost as well as pinout reasons. All accelerators are in the same power island; hence per-accelerator supply voltage calibration is not an option in our platform case study. Unfortunately, due to its overhead in terms of area occupation, per-core dynamic voltage scaling is amortized only for large and complex cores. As such, it is not a realistic option in embedded platforms featuring small processing elements such as the one we are targeting in this work. In Figure 5.1, we show a chart for overhead on mm^2 provided by ST Microelectronics for power switches at 45nm CMOS technology. Power switches and independent power grids are needed to support fine-grained DVS. In addition, having multiple supply voltages for each core implies a high cost for the power controller (e.g. DC-DC converter). These overheads are clearly not affordable at the granularity of the data-processing cores used in embedded media-processing. For this reason we assume in our work that the cores are in a common voltage island.

We link the variability on platform multicore as the different supported frequencies and

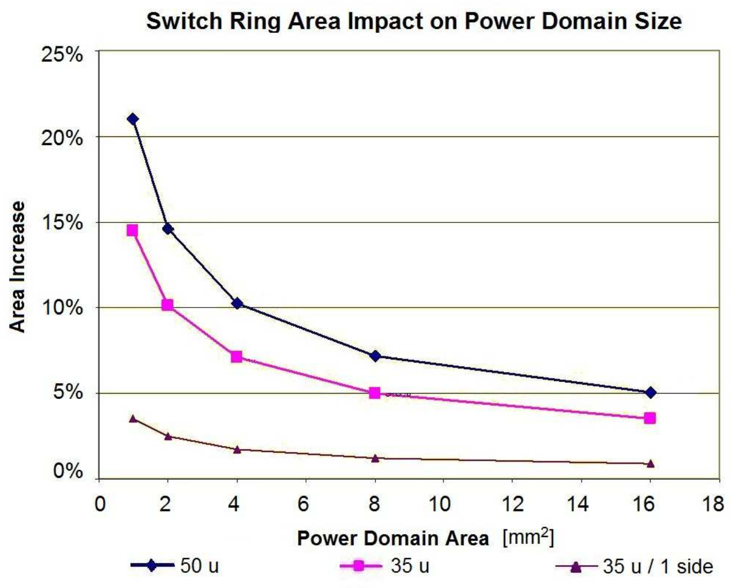


Figure 5.1: Switch ring area impact on power domain size. ST Microelectronics for power switches at 45nm CMOS technology.

power consumptions among the cores. The main contribution of this study is the definition and experimental validation of optimal non-uniform workload allocation policies that compensate for platform variability both in terms of predictability and energy efficiency.

We address the problem of distributing tasks onto accelerators with the primary objective of minimizing deadline violations and the secondary goal to minimize energy consumption. This goal ordering is dictated by the fact that frame-rate violations may severely degrade the quality of user experience and should be avoided as much as possible.

We define a static allocation policy where globally optimal allocation is computed with a computationally intensive Integer-Linear Programming (ILP) solver. This approach is useful as a design-time lower-bounding analysis step to assess optimality losses of on-line policies, or it can be used at application start-up time if the number of accelerators is not large and thus ILP solution time on the CPU is smaller than a couple of seconds.

Second, we define a two-phase approach based on linear programming (LP) and customized bin packing algorithm (BP). This algorithm is sub-optimal but it is much faster than ILP and can definitely be applied at application start-up even for large coprocessor arrays. Allocation policies computed at application start-up are applicable when the workload does not change significantly on a frame-by-frame basis, as in the case of image enhancement applications, which perform very regular pixel operations (e.g. Gaussian filtering, color-space conversion, etc.).

The proposed policies exploit the knowledge of the degradations of the performance (i.e. maximum supported clock frequency) among the cores, which can be provided either by offline characterization or by online monitors. For instance in [61] the authors propose a monitoring structure which can anticipate timing violations. Moreover the paper demonstrates that this monitor can be scalable, low power, and with low area overhead. In [26] a high bandwidth critical path monitor is proposed. This monitor can provide real-time timing information to a variable voltage/frequency scaling. Power-reduction techniques such as clock gating cause wide fluctuations in supply voltage. Those variations impact timing violations. This problem is also referred as voltage emergency. In [62] a voltage emergency predictor is proposed to learn the combinations of control flow and microarchitectural events causing voltage emergencies and prevent the timing violations. In [35] the authors exploit hardware solutions with additional run-time software to address problematic code sequences that cause recurring voltage swings. In [43] the authors present a microarchitectural control that limits supply voltage fluctuations with a nearly negligible impact on performance and energy.

To test the effectiveness of the proposed policies for variability compensation, in the experiments we explore the design space in terms of numbers of accelerators, and we test a large set of different workloads and tightness levels of deadline constraints. We also compared with state-of-the-art solutions for variability-aware energy minimization [75]. To show the impact on variability compensation, we generated a number of variability affected platforms with different performance/power characteristics and we analyzed the variability compensation capabilities to demonstrate that our policies are much more robust against platform variations in terms of real-time predictability while providing competitive energy savings.

5.1.1 Target system and variability model

The target application we consider in this work is characterized by a set of independent tasks synchronized on a barrier for which a global deadline is specified. We assume that each task is characterized by a number of instructions which is known at release time. This number corresponds to a given number of cycles, which also takes into account cycles lost for shared memory contention as a fraction of the executed ”useful” cycles. We considered a fixed number of cycles spent for shared memory accesses for each task which may result from task execution profiling or worst case analysis. The goal of our allocation policies is to map tasks to cores such that deadline constraints are met with minimum energy consumption. The platform we refer is xSTsim presented in Section 2.3. We generated set of degraded platform by applying VAM, the tool presented in Section 4.3.1, on xSTsim.

The rest of the chapter is organized as follows. In Section 5.2 we discuss related work, in Section 5.3 we present our variability-tolerant workload allocation policies, with details on ILP and LP+BP formulations. Finally in Section 5.4 we show experiments and results.

5.2 Related work

Allocation and scheduling in multicore architectures which are not affected by variations has been extensively studied, very often using ILP (see for instance [85]). Recently, much attention has been given to task allocation and scheduling strategies for MPSoCs affected by variability and aging. Integer Linear Programming (ILP) techniques for variability affected platforms have been proposed in [75, 83, 86], where the objective is the minimization of the product between the energy consumption and the delay squared.

The works in [75, 83] assume a different workload model, which can be described by a task graph with inter-task dependencies. Moreover, the approaches are fully static and cannot be applied on-line. A process variation-aware thread mapping has been recently proposed in [38]. In this work the main purpose is to maximize performance with focus on loop-intensive applications: threads undergo a first run of the main loop of each task to detect the impact of core speed on the thread execution time. This information is then used for the following mapping step. Compared to our work, this approach does not provide an optimal solution and does not take energy consumption into consideration.

In [81] a statistical scheduling approach is proposed to mitigate the impact of parameter variations in a multiprocessor platform. The strategy assumes that task executions are statistical rather than deterministic. A new metric is introduced called performance yield, defined as the probability of the assigned schedule meeting the timing constraints. In this work, authors demonstrate that using a statistical scheduling approach consistently improves the performance yield. The proposed policy is based on a static estimation of task execution times and variability information and it does not consider power consumption.

Task allocation and scheduling techniques have been recently proposed to handle aging effects. In [39] a task allocation and scheduling technique is presented whose objective is to maximize system lifetime under a given performance constraint, however energy consumption is not taken into account.

Most closely related to our approach, variability aware workload allocation policies for independent task sets are presented in [75, 86]. In the former paper, two policies are considered, aiming at maximizing performance or minimizing power, with the assumption that voltage scaling is available on a per-core basis (this is not supported in our platform). Moreover [75] assumes that the number of tasks is not larger than the number of cores. In our experiments we compare with modified versions of the policies described in [75], with suitable extensions for our system setup. In [86] the proposed policies explicitly consider time constraints as input of the problem, as in our case. However, energy minimization is achieved by using an ILP solution, which has a large computational cost and can not be applied online.

5.3 Variability-tolerant workload allocation

To the purpose of deriving an effective formulation of the optimal workload allocation problem, some assumptions have been made that are described in this section. We start from the knowledge of the total number of tasks and of the cycle budget for each task. Furthermore, we assume that the actual frequency of each core (considering the impact of variations) and its power consumption, both static and dynamic, are also known.

Based on these assumptions, we formulated the problem as described in the next subsection. We first describe the optimal ILP technique, and then we describe the approximated LP+BP approach.

5.3.1 ILP problem formulation

The ***ILP***, *Integer Linear Programming* formulation considers binary variables to represent the allocation of a generic task j on core i. The total number of binary variables is given by the number M of tasks times the number N of cores.

The total energy is expressed as a function of the binary variables and the static and dynamic contributions in active and idle states such that a linear function is obtained. It must be noted that in this formulation we consider two power states, active and idle. However the proposed approach can be generalized to consider a larger number of idle states (e.g. power gating, clock gating).

Each core *i* is characterized by $(P_{dynA}, P_{staA}, P_{staI})_i$, while each task *j* is characterized by the number of cycles C_j. For each core *i* running at frequency f_{cki} we express its active time as T_{Ai} and its idle time T_{Ii} as follows:

$$T_{Ai} = \frac{C_{Ai}}{f_{cki}} \quad T_{Ii} = \frac{C_{Ii}}{f_{cki}} \tag{5.1}$$

where C_{Ai} is the number of cycles spent in *activity* state while C_{Ii} is the number of cycles spent in *idle* state.

By considering the (t_j, c_i) pair that characterizes the mapping of task *j* on core *i*, the associated binary variable $x_{i,j}$ assumes value 1 if the task is mapped on the core, 0 otherwise. In this case the total energy is given by:

$$E_{TOT} = \sum_{i=1}^{N} \left[\frac{(P_{dynAi} + P_{staAi})\, C_{Ai}}{f_{cki}} + \frac{P_{staIi} C_{Ii}}{f_{cki}} \right] \tag{5.2}$$

For each core *i*, its execution time in *activity* and *idle* states can be expressed as a function of the task execution times:

$$T_{Ai} = \textstyle\sum_{j=1}^{M} \frac{x_{i,j} C_j}{f_{cki}} \quad T_{Ii} = T - T_{Ai} \tag{5.3}$$

where the T is the time constraint by which the workload must be executed. The total energy becomes:

$$E_{TOT} = \sum_{i=1}^{N} \left[\frac{\left(P_{dynAi} + P_{staAi} - P_{staIi}\right)}{f_{cki}} \sum_{j=1}^{M} (x_{i,j} C_j) \right] + T \sum_{i=1}^{N} P_{staIi} \tag{5.4}$$

To obtain a linear function, we add N dummy variables:

$$E_{TOT} = \sum_{i=1}^{N} \left[\frac{\left(P_{dynAi} + P_{staAi} - P_{staIi}\right)}{f_{cki}} \sum_{j=1}^{M} (x_{i,j} C_j) \right] + T \sum_{i=1}^{N} x_{N+1,i} P_{staIi} \tag{5.5}$$

Now, given the following vector of binary variables:

$$\begin{aligned} X = (&x_{1,1}, \ldots, x_{1,M}, \ldots, x_{N,1} \ldots, x_{N,M}, \\ &x_{N+1,1} \ldots, x_{N+1,N}) | x_{i,j} \in \{0,1\} \, \forall i, j \end{aligned} \tag{5.6}$$

The ***ILP*** formulation of the problem becomes:

$$\min_X E_{TOT}$$
$$\begin{cases} \sum_{j=1}^{M} x_{i,j} = 1 & \forall i : 1 \ldots N \\ x_{N+1,j} = 1 & \forall j : 1 \ldots N \\ \sum_{j=1}^{M} \frac{x_{i,j} C_j}{f_{cki}} \leq T & \forall i : 1 \ldots N \end{cases} \tag{5.7}$$

The first constraint imposes that each task is allocated on only one core. The second constraint determines the dummy variables while the third one concerns the execution time constraint T. The ILP solver (we used ILOG [41]) wants both coefficient and variable vectors with the same size; indeed we need the dummy variables.

As mentioned in the introduction, the ILP solution mainly represents an optimal reference for the faster heuristic policies described below. On the other hand, it could be actually applied before application start-up if the number of accelerators is not large and thus the solution can be computed in a time much smaller with respect to application execution time.

5.3.2 LP+BP problem formulation

An approximate approach that lends itself to be applied to platforms with a larger number of accelerators is based on a two-phases approach based on Linear Programming ***LP*** followed by Bin Packing ***BP***. The algorithm can be run at the beginning of the application and requires the knowledge of the cycle budget for each task.

5.3.2.1 LP: first step

The LP step starts from the total number of cycles of all the tasks (called K). The goal of the LP is to assign a cycle budget to each core disregarding task granularity. We express the energy consumption as for ILP but here the number of variables is ***2N*** and they represent the number of cycles each core must execute in active and idle cycles. Referring to (5.2) we consider the vector R of $2N$ coefficients:

$$R = (\frac{P_{dynA1} + P_{staA1}}{f_{ck1}}, \frac{P_{staI1}}{f_{ck1}}, \ldots, \frac{P_{dynAN} + P_{staAN}}{f_{ckN}}, \frac{P_{staIN}}{f_{ckN}}) \tag{5.8}$$

and the vector C of $2N$ real variables that will then be rounded up to the closest integer:

$$C = (C_{A1}, C_{I1}, \ldots, C_{AN}, C_{IN}) \tag{5.9}$$

The ***LP*** minimization problem can be expressed as:

$$\min_C R \cdot C^T \\ \begin{cases} \dfrac{C_{Ai} + C_{Ii}}{f_{cki}} = \dfrac{C_{Aj} + C_{Ij}}{f_{ckj}} & \forall i, j : 1 \ldots N, i \neq j \\ \sum_{i=1}^{N} C_{Ai} = K \\ \dfrac{C_{A1} + C_{I1}}{f_{ck1}} \leq T \end{cases} \tag{5.10}$$

The first constraint concerns the sum of idle and active times that must be equal for all the cores, the second one concerns the total number of cycles while the third one the maximum execution time T.

5.3.2.2 BP: second step

Thanks to the LP solution, each core *i* is assigned an optimal budget of cycles C_{Ai}. If task allocation were able to exactly match this budget the minimum energy condition will be achieved within the time constraint.

However this is not possible in general. To achieve a good mapping a Bin Packing algorithm is used. We considered *Best Fit Decreasing* solution, which ranks the tasks from the biggest to the smallest and the cores from the one with lower capacity to the one with higher budget (also called capacity, i.e. C_{Ai} LP solutions). The algorithm proceeds by taking the current task and mapping it into the core with minimum capacity to fit it, then the cores are reordered considering the remaining capacities and the next task is considered. We show two different implementations of the Bin Packing, the first one is composed by two steps, namely ***Step1()*** and ***Step2standard()*** and the second one composed by three steps, namely ***Step1()***; ***Step2custom()***; ***Step3custom()***. The second version is a custom version which is more suitable for our problem.

The pseudo-code of *Step1* is shown in Listing 5.1 and is described as follows. The *Tasks* array contains the number of cycles needed by each task, and the *Cores* array contains the remaining capacities of the cores. Line02: Tasks are sorted from the biggest to the smallest according to their estimated cycles. Line04: For each task the cores are sorted according to the smallest residual capacity. Line05: A loop around the cores to find the first one that has enough left capacity for the current task is done. When such core has been found, the algorithm records the mapping and updates the remaining capacity for the selected core. The function returns the number of mapped tasks that is useful to understand if all tasks have been mapped.

Listing 5.1: the Step1 function which implements the Best Fit Decreasing to solve the formulated Bin Packing Problem.

```
int Step1(Tasks, Cores, CORE_NUMBER, TASK_NUMBER) {
int mapped_t_num=0;
DescentingSort(Tasks);
for (t=0;t<TASK_NUMBER;t++) {
      AscentingSort(Cores);
      for (c=0;c<CORE_NUMBER;c++) {
            if (Task[t] <= Cores[c]) {
                  Assign(t,c);
                  Cores[c] -= Tasks[t];
                  mapped_t_num++;
                  break;
            }
      }
}
return mapped_t_num;
}
```

By applying this solution, it is in general possible that some tasks cannot be allocated because none of the cores has enough remaining cycle budget. In this case, if the unassigned tasks are mapped by minimizing the exceeding cycle budgets, the time constraint would be violated. This is the behavior of the standard BP. We refer to this as the *Step2standard*, and we show it in Listing 5.2.

Listing 5.2: the Step2standard function allocates the tasks while minimizing the exceeding cycle budget.

```
int Step2standard(Tasks, Cores, CORE_NUMBER, TASK_NUMBER, t_large) {
int mapped_t_num=t_large;
for (t=t_large;t<TASK_NUMBER;t++) {
      c_exc=0;
      min_exc = abs(Cores[c_exc] - Tasks[t]);
      for (c=1;c<CORE_NUMBER;c++) {
            if (min_exc > abs(Cores[c] - Tasks[t]) ) {
                  c_exc = c;
                  min_exc = abs(Cores[c_exc] - Tasks[t]);
            }
      }
      Assign(t,c_exc);
      Cores[c_exc] -= Tasks[t];
      mapped_t_num++;
}
return mapped_t_num;
}
```

The Step2standard function, starting from the largest task (having index t_large) that has not been allocated in Step1, finds the core that can execute it with the minimum exceeding cycle budget. The function then continues with the other tasks and returns the number of the mapped tasks that can be equal to the total number of tasks (it can be useful for checking). It must be noted that the subtraction between the number of remaining cycles for a core and the cycles needed by the current task is always negative.

In this work, we propose a variant, where the idea is to check time constraints instead of minimizing the exceeding cycle budget. This applies when there are no cores with enough remaining capacity to fit a certain task. In this case the residual time is computed as the difference between the time constraint and the estimated assigned execution time so far, given by the number of already assigned cycles plus the cycles needed by current task; all is then divided by the frequency. The task is assigned to the first core for which the estimated activity time is shorter than the deadline. We explore the cores starting from the one with larger capacity (i.e. the inverse order). In this way we force to fit the tasks minimizing the exceeding cycle budgets of the cores (i.e. that means the exceeding LP solution, which is the input of BP), in order to lower the energy consumption. The variant that we propose for the *Step2standard* is the *Step2custom* which is shown in Listing 5.3.

Listing 5.3: the Step2custom function tries to allocate the tasks while meeting the deadline.

```
  int Step2custom(Tasks, Cores, Freqs, StartCoreBudgets ,CORE_NUMBER
                ,TASK_NUMBER, t_large , time_constr) {
01 int mapped_t_num=t_large;
02 for (t=t_large;t<TASK_NUMBER;t++) {
03      AscentingSort(Cores);
04      for (c=CORE_NUMBER-1;c>=0;c--) {
05              if (time_constr >
06                      ((StartCoreBudgets[c]  - (Cores[c] - Tasks[t]))
                                                / Freqs[c]) ) {
07                      Assign(t,c);
08                      Cores[c] -= Tasks[t];
09                      mapped_t_num++;
10                      break;
11              }
12      }
13 }
14 return mapped_t_num;
15 }
```

In the *Step2custom* function, starting from the largest task that has not been allocated in *Step1*, we find the core that can execute this task while meeting the time constraint ($time\_constr$). For each task we find the first core that can meet the deadline when adding the execution time of the current task. In Line06, we can see that we use the current remaining cycles ($Cores[c]$) by subtracting the cycles of the current tasks; from the initial core budget, we can estimate the allocated cycles and the execution time. This function needs the *Freqs* array which contains the frequencies of the cores, and the *StartCoreBudgets* array which contains the start cycles budget of the cores. The function returns the total number of allocated tasks. In case there are some tasks that are not allocated we need a third step shown in the function in Listing 5.4.

Listing 5.4: the Step3custom function allocates the tasks while minimizing the overrunning of the deadline.

```
   int Step3custom(Tasks, Cores, Freqs, StartCoreBudgets
                 ,CORE_NUMBER,TASK_NUMBER, t_large) {
01 int mapped_t_num=t_large;
02 for (t=t_large;t<TASK_NUMBER;t++) {
03       AscentingSort(Cores);
04       c_exc=CORE_NUMBER-1;
05       t_min_exc = (StartCoreBudgets[c_exc]
                  - (Cores[c_exc] - Tasks[t]))
                     / Freqs[c_exc];
06       for (c=CORE_NUMBER-2;c>=0;c--) {
07              if (t_min_exc > ((StartCoreBudgets[c] - (Cores[c] - Tasks[t]))
                                                     /Freqs[c]) ) {
08                     c_exc = c;
09                     t_min_exc = (StartCoreBudgets[c_exc]
                                 - (Cores[c_exc] - Tasks[t]))
                                    /Freqs[c_exc];
10              }
11       }
12       Assign(t, c_exc);
13       Cores[c_exc] -= Tasks[t];
14       mapped_t_num++;
15 }
16 return mapped_t_num;
17 }
```

In the *Step3custom* function, we find for the remaining tasks a mapping that minimizes the execution time over the deadline. For each task we find the core that can execute it while minimizing the time of the deadline miss. We explore the cores starting from the one with the biggest capacity (i.e. in the inverse order) to minimize the exceeding cycles and thus the energy consumption. If the platform is designed with conservative time margins for the target applications, this step should not be executed.

5.3.3 Rank-based techniques

The proposed strategies will be compared in the experimental results section with approaches that have been presented in literature to address the problem of allocation of independent tasks on variability affected cores [75].

- **Rank Frequency.** This technique oriented to performance maximization performs a dynamic allocation by assigning the current task on the available core with higher frequency. It derives from the VarF&AppIPC presented in [75] but differs from the original version in that it can be applied also when the number of tasks is larger than the number of cores. Moreover, we do not sort tasks based on the IPC, rather we consider it constant. Finally, we do not apply the second stage exploiting voltage assignment because we consider platforms having a fixed supply voltage.
- **Rank Power.** This technique oriented to power minimization performs a dynamic allocation by assigning the current tasks on the available core characterized by the minimum power consumption. It derives from the VarP&AppP presented in [75] as the cores with smaller total power consumption are selected first. Differently from the original versions, we do not sort tasks based on dynamic power.

VarF&AppIPC and VarP&AppP are the names of policies in [75]. We extended them into the Rank policies. Rank Frequency allocates a new task on the fastest core available; Rank Power allocates a new task on the lower-power core available. Rank policies are the closest to our approach we found.

5.4 Experiments

5.4.1 Setup

In the first set of experiments, reported below, the xSTsim cores have been synthesized on STMicroelectronics 65nm high-speed technology. Due to confidentiality concerns, all results are expressed in normalized form with respect to the nominal frequency and power. Variability data has been obtained through the VAM methodology, as outlined in Section 4.3.1. It is important to notice that the ratio between leakage and dynamic power is not constant, as higher operating frequency is generally coupled with faster, higher-leakage transistors. Thus, the leakage power of the fastest core accounts for as much as 20% of the dynamic power. We consider leakage consumption in power-gating state as variation-free because it is controlled by very large power gating transistors turned off in power-gating state. These transistors can be biased with a suitable gate voltage to ensure that variability effects are negligible. In Table 5.1 the normalized frequency and power characteristics of the cores used for the experiments in this section are detailed.

core	f_{ck}	P_{dynA}	P_{lkgA}/P_{dynA} (4)	P_{lkgPG}/P_{dynA} (4)	P_{totA}/f_{ck}
1	1.14	1.07	$2.14E-01$	$2.00E-05$	1.11
2	1.07	1.04	$1.56E-01$	$2.00E-05$	1.11
3	1.01	1.01	$7.10E-02$	$2.00E-05$	1.06
4	1.00	1.00	$1.00E-02$	$2.00E-05$	1.00
5	0.97	0.99	$6.90E-03$	$2.00E-05$	1.01
6	0.95	0.97	$4.86E-03$	$2.00E-05$	1.01
7	0.93	0.95	$3.81E-03$	$2.00E-05$	1.02
8	0.89	0.93	$2.79E-03$	$2.00E-05$	1.04
MV :	21.93%	13.18%	98.70%	0.00%	10.28%
av :	$9.95E-01$	$9.95E-01$	$5.87E-02$	$2.00E-05$	$1.05E+00$
std :	$7.97E-02$	$4.58E-02$	$8.27E-02$	$0.00E+00$	$4.67E-02$

Table 5.1: Variability-affected MPSoC. In the headlines we used the following notations: *A* for Activity, *lkg* for leakage, *MV* for Maximum Variation (i.e. $(max - min)/max$), *av* for average, *std* for deviation, f_{ck} is the maximum clock frequency supported by the core.

From the values in Table 5.1, we obtain that the maximum variation of the energy required to execute a task on any two different processors is 10.28%. This can be considered as an upper-bound in the energy consumption difference achievable by task allocation. We considered four and eight-core platforms. Referring to Table 5.1, four-core platforms use core numbers 2, 4, 6, and 8.

Tasks are characterized by their instruction budget. For our experiments we generated task sets, characterized by the number of tasks and the deviation of the number of instructions per task. The total instruction budget of application is fixed for each task set. For a given number of tasks, we considered one task set with no deviation, i.e. all tasks are equal, and additionally we generated 8 different task sets for each non-zero value of deviation. We used two non-zero values of deviation.

In our experiments another key parameter is the tightness of timing constraints. These constraints have been selected to obtain variable platform utilization. We computed the minimum time to execute a given total number of cycles, which imposes a 4 cores platform utilization of 100%. We then obtained more relaxed deadline constraints (*tconstri*) as follows:

$$\begin{array}{l} tconstr1 = 1.05 \cdot time_{min} \\ tconstr2 = 1.10 \cdot time_{min} \\ tconstr3 = 1.20 \cdot time_{min} \\ tconstr2 = 1.40 \cdot time_{min} \end{array} \tag{5.11}$$

where:

$$time_{min} = \frac{K}{\sum_{i=2,4,6,8} f_{cki}} \tag{5.12}$$

In our experiments we considered the total number of cycles *K* being 80e+6. It must be noted that the more relaxed constraint imposes a platform utilization of just 60%.

5.4.2 The advantage of variability-aware allocation

In this section, we show what is the advantage in terms of energy consumption and performance of variability-aware allocation using the proposed methods by comparison with rank-based techniques. To achieve this objective, we first compute the maximum and the minimum energy consumption to execute a given task set on the platform. The minimum energy (without taking into account timing constraints) is obtained when all the tasks are executed by the core with minimum energy and similarly for the maximum energy.

We use these extreme values to normalize the energy when comparing the different techniques under consideration. I.e. for the generic energy E spent during the execution of a given workload, we normalize it using the following metric: $(E - E_{min})/(E_{max} - E_{min})$. We compute the deadline miss rate related to each group of 8 task sets, where a group is characterized by a total number of tasks (i.e. 8, 32, and 128) and by a deviation. The miss rate is computed as the number of tasks violating the deadline. The entity of the deviation is expressed in a relative way with respect to the average number of cycles per task. We identified three levels of deviation, namely 0, 0.25 and 0.5. For instance, a deviation of 0.5 means that the number of cycles of tasks can be half the average. For 0 deviation only one task set exists. In what follows we show the comparison results for the two cases of a 4-core platform and 8-core platform. We will use the following abbreviations: ILP: Integer Linear Programming -based policy, LP+BP: Linear Programming + Bin Packing -based policy, RF: Rank Frequency policy, RP: Rank Power policy.

5.4.2.1 Results using 4-core platform

Figure 5.2 shows the energy comparison among the policies when the total number of task of the application is 8 and they are characterized by high deviation (level 0.5). The proposed ILP solution provides better results in terms of energy consumption and lots of deadlines are met. Also LP+BP meets many deadlines but uses more energy. It must be noted that our policies are able to save energy when the time constraint is more relaxed. Rank based policies spend the same energy independently from the constraints (they do not take them into account) and violate the deadlines in most of the cases. Details about timing violations are shown in Table 5.2, where miss rates are reported.

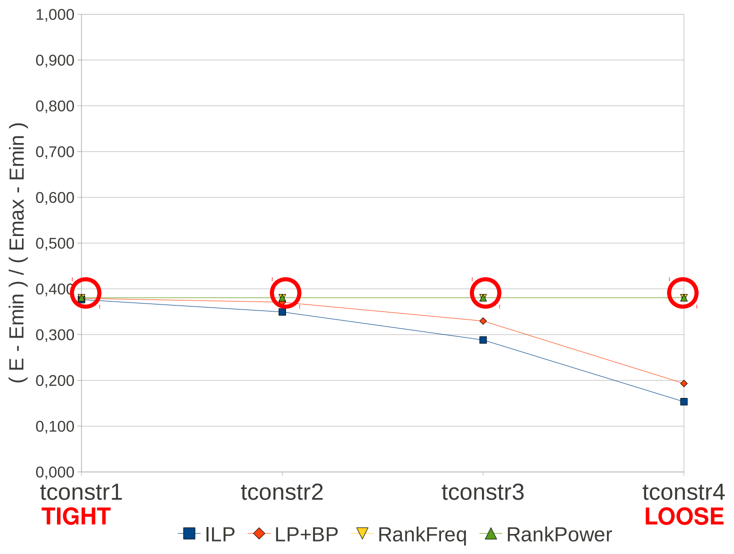


Figure 5.2: 4 cores. Normalized Energy Comparison. The Number of Task is 8, the deviation is 0.5. A circle means that some deadlines are not met. tconst*i* is the constraint level *i*. The rank approaches give very close results, so they are hardly distinguishable in the plot.

	tconstr1	tconstr2	tconstr3	tconstr4
ILP	0.13	0.00	0.00	0.00
LP+BP	0.75	0.25	0.00	0.00
RankFrequency	1.00	1.00	0.75	0.38
RankPower	1.00	1.00	0.75	0.38

Table 5.2: Deadline miss rate. 4 cores. 8 tasks. 0.5 for deviation.

The same comparison has been done considering an application made of 32 tasks. Results show that in this case LP+BP achieves similar results with respect to ILP in terms of energy consumption (Figure 5.3). This is because a large number of smaller tasks (in terms of number of instructions) are easier to allocate. Also, from a miss rate point of view, rank based policies perform better than in the previous case, however they are always worse than both ILP and LP+BP, as shown in Table 5.3.

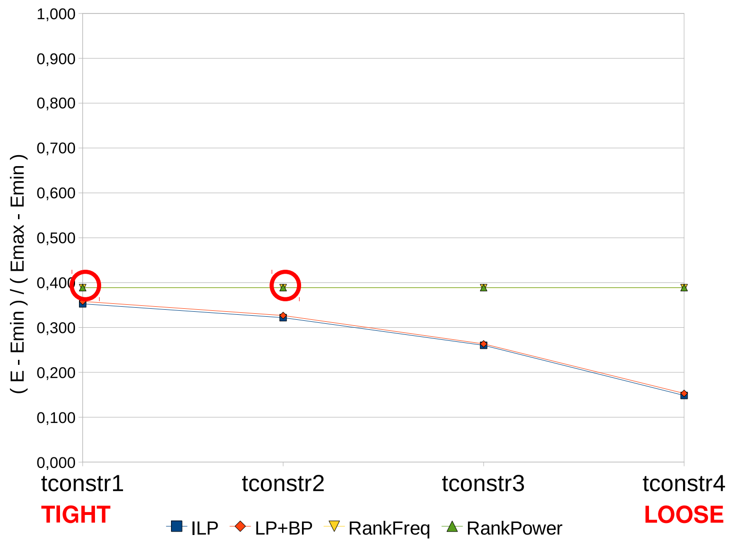


Figure 5.3: Normalized energy comparison. 4 cores. The Number of Task is 32, the deviation is 0.5. A circle means that some deadlines are not met. tconst*i* is the constraint level *i*. The rank approaches provide very close results, so they are hardly distinguishable in the plot.

	tconstr1	tconstr2	tconstr3	tconstr4
ILP	0.00	0.00	0.00	0.00
LP+BP	0.00	0.00	0.00	0.00
RankFrequency	0.88	0.38	0.00	0.00
RankPower	1.00	0.38	0.00	0.00

Table 5.3: Deadline miss rate. 4 cores. 32 tasks. 0.5 for deviation.

5.4.2.2 Results using 8-core platform

We performed experiments on a platform with higher parallelism. We considered task sets of 8 tasks and highest deviation. RankFrequency and RankPower spend a considerable amount of additional energy with respect to ILP and LP+BP (see Figure 5.4). Moreover, they provide much larger miss-rates. It must be noted that, differently from the 4-cores platform, the proposed strategies gain a considerable amount of power also for tighter time constraints. This is because they are able to better exploit the additional degrees of freedom for the allocation provided by the larger number of cores.

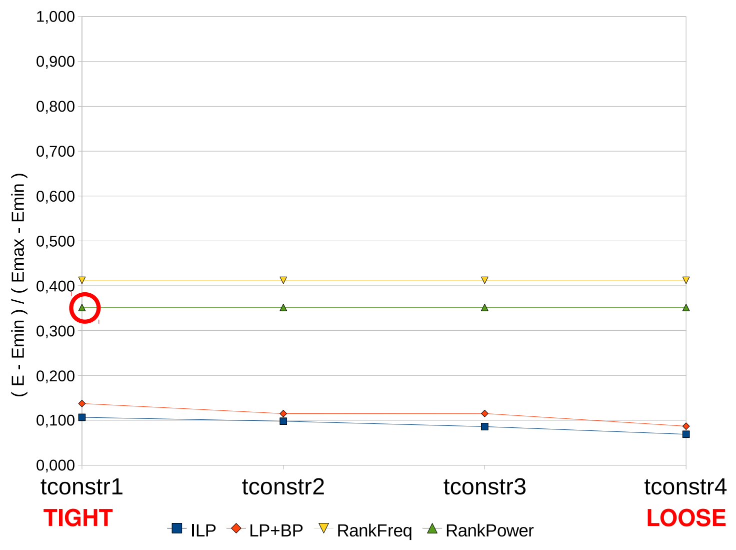


Figure 5.4: Normalized Energy Comparison. 8 cores. 8 tasks for each task-set with 0.5 for deviation. A circle means that some deadlines are not met. tconst*i* is the constraint level *i*.

To compare the capability of the proposed strategies to efficiently use the platform, we show platform utilization details for the 8-cores platform in Table 5.4. A utilization of 100% means that all the cores are used at least once. Since timing constraint values have been tuned to the 4-cores platform, the whole computational power of the 8-cores is under-utilized on average and a smart allocation policy should exploit this to reduce energy consumption by switching off some of the cores. In Table 5.4 the 45% value means that the 55% of cores are never used.

	tconstr1	tconstr2	tconstr3	tconstr4
ILP	42%	47%	50%	63%
LP+BP	36%	42%	45%	58%
RankFrequency	100%	100%	100%	100%
RankPower	100%	100%	100%	100%

Table 5.4: Platform Utilization Percentage. 8 cores. 8 tasks for each task-set with 0.5 as deviation.

5.4.3 Variability compensation analysis

The results obtained so far were referring to a specific variability-affected platform. However, being variability a statistical effect, we need to study the effectiveness of the policies on many of such platforms. To this purpose we performed a set of experiments using a number of platforms generated using VAM. The objectives of the proposed analysis are the following:

1. to show the impact of variations in terms of performance and energy at the application level;
2. to demonstrate how variability-aware task allocation policies in general are effective in reducing the impact of variability, however state of the art policies are not able to compensate both energy and performance impact with the same effectiveness at the same time;
3. to demonstrate that the policies we designed are able to reduce the impact of variability on energy while matching time constraints.

To highlight the energy gains with respect to RF/RP, we normalized the energy levels with respect to the energy provided by ILP (best case). This is the purpose of Figure 5.5, where for each platform the normalized energy consumption of LP+BP, Rank Frequency, Rank Power is represented. ILP consumes the minimum energy with no deadline misses. The plot highlights that LP+BP allows energy savings almost as significant as those achieved by ILP, whereas the Rank policies consume more energy and lead to deadline misses.

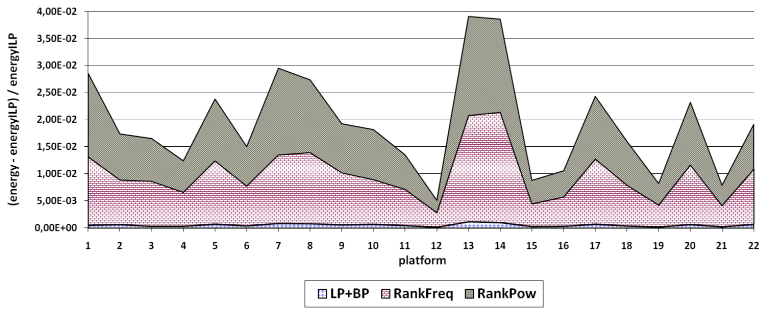


Figure 5.5: Energy comparison among LP+BP, Rank Frequency, and Rank Power techniques. Values are normalized by the ILP's.

To show the cumulative impact of these policies on variability affected platforms, in Figure 5.6 we reported the energy consumption for all the 4 policies. For each policy we reported minimum, maximum and average energy consumption values for the execution of the representative benchmark consisting of 80Mcycles. The plots show that LP+BP and ILP policies provide always lower energy by considering average cases. Considering the maximum energy consumption, RP provides slightly lower maximum energy, however this comes at the price of a very large miss rate.

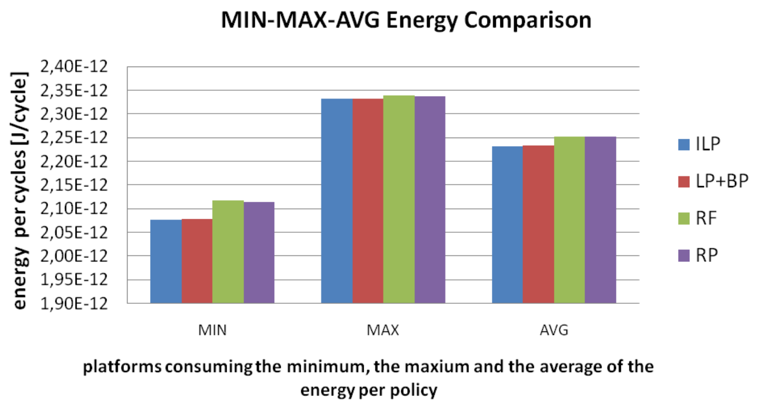


Figure 5.6: MIN-MAX-AVG Energy Comparison. Energy consumption comparison: cumulative results across all platforms considering MIN, MAX and AVG energy for each policy.

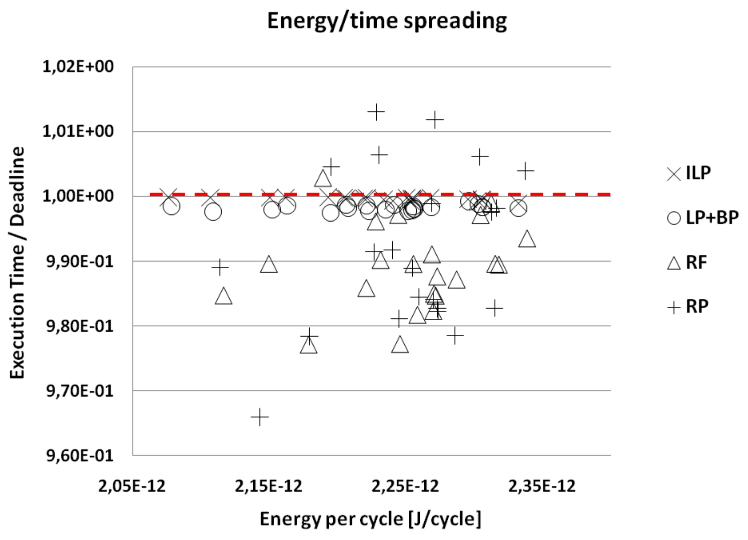


Figure 5.7: Energy per cycle / Time Spreading. Execution time vs. energy consumption per cycle. The execution time is divided by the deadline. The horizontal dashed row identifies the deadline.

As mentioned before, the proposed policies are much more effective in compensating performance impact of variations with respect to RF and RP. This is evident by observing Figure 5.7. Since each platform can be more or less energy consuming depending on the ratio between power and clock frequency of its own cores, to evaluate in a better way the spent energy across the different platforms we used the metric of energy per cycle. Here it can be noted that the proposed policy compensates variations by reducing time violations due to variability effects and leads to predictable performance results. Indeed, the execution times provided by LP+BP (and also by ILP) are very close but lower than the value 1, which identifies the time constraint, independently from the platforms, which is the time constraint we used for these experiments. On the other side, rank policies lead to much more variable execution times. It must be noted that, by considering each single platform, our policies provide always lower energy while matching time constraints. Finally, it must be noted that for our policies the energy spread is slightly larger, but mainly because our policies are aimed at minimizing energy (indeed the minimum energies are provided by our policies), not to match a given energy budget.

5.5 Summary

In this chapter, we presented the definition and experimental validation of optimal non-uniform workload allocation policies that compensate for platform variability both in terms of predictability and energy efficiency. We addressed the problem of distributing tasks onto accelerators with the primary objective of minimizing deadline violations and the secondary goal to minimize energy consumption. First we defined a static allocation policy where globally optimal allocation is computed with a computationally intensive integer-linear programming (ILP) solver. Second, we defined a sub-optimal two-phase approach based on linear programming (LP) and bin packing (BP). We demonstrated through experiments conducted on an industrial platform simulator the effectiveness of the proposed policies using a large set of different workloads and tightness levels of deadline constraints. We also compared with state-of-the-art solutions for variability-aware energy minimization to demonstrate that our policies are much more robust in terms of real-time predictability while providing competitive energy savings.

Regarding the two proposed approaches namely ILP and LP+BP, the first one gives the optimal solution but it is very time demanding whereas for LP+BP some improvement in execution time can be reached. In the next chapter we will show how it is possible to apply the LP+BP-based policy at runtime.

Chapter 6

A variability-aware run-time task allocation

6.1 Overview

The previous chapter gave important insights to the problem of the energy minimization under real-time constraints for multicore platforms. However, in most of real scenarios task allocation techniques need to be executed at runtime, and this means that their algorithms must be simple.

The aim of this chapter is to improve the implementation of the policy based on the LP+BP formulation to apply that strategy at runtime. We based on the same hypothesis regarding the platforms, the variations and the workload illustrated in Chapter 5. In particular the contribute we provide in this chapter are threefold and can be summarized as follows.

First, we propose a new formulation of the problem which allows to design a linear-time algorithm to solve it and that can be easily applied online, i.e. at run-time. Indeed, we demonstrate that the overhead of the LP+BP solution is minimal and enables its application on a frame-by-frame basis. Second, we propose a full implementation of the LP+BP on a multicore embedded multiprocessor SoC running representative multithreaded multimedia applications, namely an MPEG2 decoder and an Integral Image algorithm, that have been parallelized and ported to the target platform as shown in Section 3.3. Their implementation exploits on-board accelerators to execute various threads in parallel while the host core accomplish dispatching functionalities and takes decisions about the allocation of the tasks by running the algorithms of the policies discussed in this chapter. Finally, we provide a comprehensive study about the effectiveness of the proposed runtime allocation technique on multimedia applications in terms of energy, deadline miss rate, scalability (both in the number of cores and tasks) and variability conditions.

To well explain the new study we review in Section 6.2 the LP+BP formulation which has been previously shown in Section 5.3.2. In Section 6.2 we also review the illustration of the comparison techniques based on [75] and presented in Section 5.3.3 with the aim to better highlight the difference between the different approaches.

Finally, we demonstrate through experimental results that our technique compensates variability, while improving energy-efficiency and minimizing deadline violations in presence of performance and power variations across the cores. The proposed policy can save up to 33% of energy with respect to the state-of-the-art policies and 65% of energy with respect to one variability-un-aware task allocation policy while providing better Quality of Service (QoS).

6.2 Variability-tolerant run-time workload allocation

We begin the description of the workload allocation policies by introducing some notations. In active state, each core *i* consumes dynamic power expressed by P_{dynAi} and leakage power expressed by P_{lkgAi}. Each core *i* consumes only leakage power while in idle state, which is expressed as P_{lkgIi}. The clock frequency of a core *i* is f_{cki}. Each core spends a certain amount of energy per cycle D_{Ai} given by $D_{Ai} = \dfrac{P_{dynAi} + P_{lkgAi}}{f_{cki}}$ in activity state, and D_{Ii} given by $D_{Ii} = \dfrac{P_{lkgIi}}{f_{cki}}$ in idle state.

We start describing the rank policies used for comparison, as they are more intuitive. These techniques are based on the scheme shown in the block diagram in Figure 6.1. A ranking of the cores is performed on the basis either on the clock frequency, dynamic power, and leakage power, depending on the specific implementation. On the other side, tasks are sorted using information about the tasks cycle budget. Finally, the tasks are allocated one-by-one on the first available core following the ranking. The solution is characterized by a vector of the binary elements $x_{i.j}$**s**. For each core *i* if the task *j* is allocated on it $x_{i,j}$ is **1**, otherwise **0**. In what follows we detail the various rank policies, each one characterized by the way the rank is performed. This choice determines the behavior of the policy. For instance, a ranking based on clock frequency will lead to smaller execution time with respect to a ranking based on power.

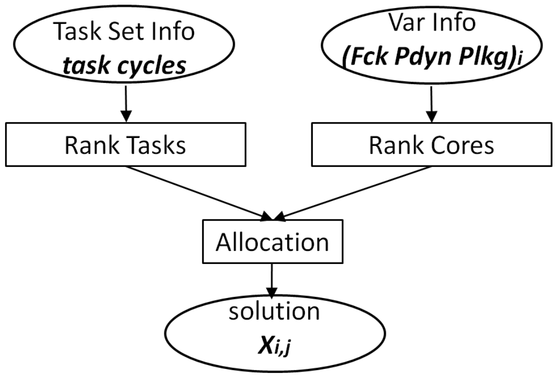


Figure 6.1: Rank policies block diagram.

6.2.1 Rank Frequency

The tasks are sorted in relation with their lengths in terms of cycles starting from the longest one. The cores are sorted in relation with their clock frequency f_{cki} starting from the speediest one. Then, the current task is allocated on the first available core; this implies that the largest task is executed by the speediest core and so on. This technique derives from the VarF&AppIPC policy presented in [75] but differs from the original version in that it can be applied also when the number of tasks is larger than the number of cores. Moreover, we do not sort tasks based on the Instruction per Cycle (IPC), rather we express each task with its activity cycles, but we sort the profiled tasks from the largest to the smallest. Finally, we do not apply the second stage exploiting voltage assignment because we consider platforms having a fixed supply voltage. The problems of this technique are: a) it does not take into account the power consumption of the cores, and then it only tries to minimize the execution time; b) when the number of tasks is greater than the number of cores it is not generally true that executing the largest task on the fastest core implies the fastest computation; it can be easily shown that executing two or more small tasks on the fastest core and the largest task on another core can be taken less time for the execution.

6.2.2 Rank Power

The tasks are sorted in relation with their lengths in terms of cycles starting from the longest one. The cores are sorted in relation with their power consumption $P_{dynAi} + P_{lkgAi}$ starting from the one at minimum power consumption. Then, the current task is allocated on the first available core; this implies that the largest task is executed by the least power consuming core.

It derives from the VarP&AppP presented in [75] as the cores with smaller power consumption are selected first. The problems of this technique are: a) it does not take into account the time needed to execute all tasks; b) executing one task on the core at minimum power does not imply that it will consume the minimum energy because the energy also depends from the execution time which depends from the clock frequency of the core.

6.2.3 Rank Energy

We introduce this technique to solve problem b) of Rank Power. We characterize each core *i* by its own ratio between power and clock frequency. This ratio consists in the energy per cycle D_{Ai}. Sorting the cores from the one at minimum energy per cycle and sorting the tasks from the longest one, we allocate the current task on the first available core. This implies that the largest task is executed by the least energy consuming core. Besides, the problem of this technique is that when the number of tasks is greater than the number of cores, it is not generally true that executing the largest task on the core at minimum energy per cycle implies the lowest energy consumption; it can be easily shown that executing two or more small tasks on the core at minimum energy per cycle and the largest task on another core, the platform can spend less energy.

6.2.4 LP+BP and its fast implementation

The objective of the proposed LP+BP approach is to approximate the optimal solution in a computationally efficient way. We firstly describe the rationale behind the policy, and then we cover its mathematical formulation. The block diagram is shown in Figure 6.2.

This approximation is obtained by first determining a fine grain (cycle-level) allocation of a cycle budget to each core to minimize energy consumption while matching a given time constraint. This is done using an optimized formulation of the LP problem that does not require the usage of a solver so that the solution can be computed very fast. After this is done, the tasks are fit into the given budgets using a customized BP algorithm that takes the time constraint into account to reduce the impact on QoS when the task allocation do not fit in the given budget for one or more cores.

The first part, namely the cycle budget allocation, is performed by using clock frequency, dynamic power, and leakage power, to sort the cores according to the quantity $D_{Ai} - D_{Ii}$ (where *i* identifies the core). We point out that, in contrast with the rank policies, this approach takes into account the idle power consumption as well as the activity power consumption and the clock frequency. In particular the first core has the minimum value of $D_{Ai} - D_{Ii}$ and the last core has the maximum value of $D_{Ai} - D_{Ii}$. By considering the quantity K as the sum of the cycles of all tasks, and the time constraint T, we are able to calculate the cycle budgets that each core must spend in activity state in order to minimize the energy consumption due to executing all tasks while meeting the deadline. This is done in three steps: 1) Computation of *Solution A*; 2) Computation of *Solution B*; 3) Comparison between Solutions A and B to select the best one.

Solution A allocates cycles to the core to minimize their execution time, without taking the deadline into consideration. On the other side, Solution B exploits the knowledge of the deadline T to allocate cycles exploiting the available time. The solution leading to the minimum energy is selected. Details about the solution computation are given later in this section. We point out that if the time needed by Solution A is longer than the time constraint, the application cannot be supported by the platform.

After the cycle budgets C_{Ai}***s*** have been computed, the BP phase allocates tasks on the cores (see Figure 6.2). This is obtained by fitting the cycles of each task in the bins given by the core cycle budgets. To solve this BP problem, we use the Best Fit Decreasing (BFD) algorithm that we customized as explained in Section 5.3.2.2. The final solution is characterized by a vector of the binary elements $x_{i.j}$***s***. For each core *i* if the task *j* is allocated on it $x_{i,j}$ is **1**, otherwise **0**.

Details of Solution A and B computation are given in what follows.

Solution A. Solution A is obtained by first computing the minimum time t_{min} to execute K cycles using all cores through the formula $t_{min} = K/sum\_fck$, where sum_fck is the sum of all core clock frequencies of the given degraded platform. Starting from the first sorted core we calculate for each core *i* the activity cycle budget as $C_{Ai} = t_{min} f_{cki}$. The total energy is given by $E_{tot} = \sum_{i=1}^{N} C_{Ai} D_{Ai}$ where N is the number of the cores. Solution A suggests executing the total amount of cycles in the minimum possible time t_{min}.

Solution B. Solution B calculates for core *i* the activity cycle budget as $C_{Ai} = T f_{cki}$ starting from the first core, $i = 1$. For each core cycle budget that has been calculated we evaluate the sum of the already allocated cycles : $C = C + C_{Ai}$. We proceed to calculate the C_{Ai}***s*** until $C < K$. When this condition is not supported anymore, we will find the id-core *r* for which the cores

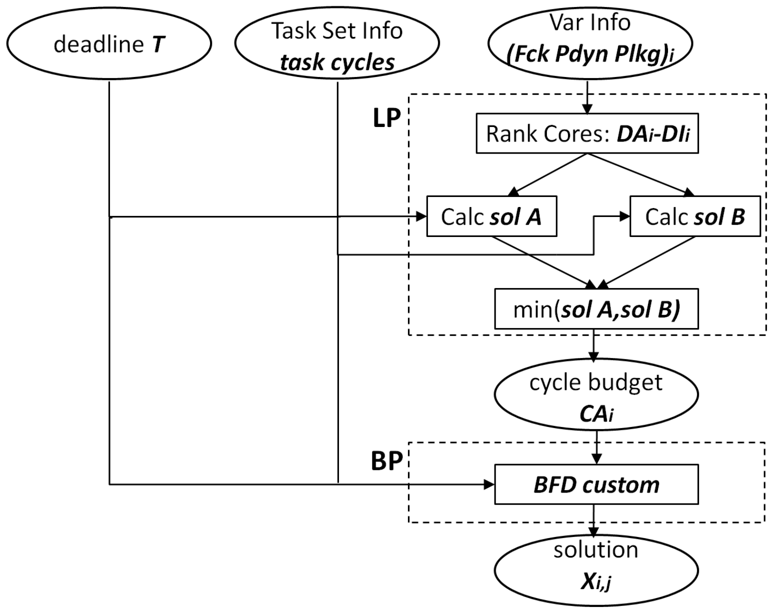


Figure 6.2: LP+BP block digram.

from 1 to $r-1$ will always work for all the time T while core *r* will generally work for shorter time spending the rest of the time in idle state, and finally the other cores will always stay in idle state. The time in activity state of core *r* can be calculated as C_{Ar}/f_{ckr} where its cycle budget C_{Ar} has been fixed to $C_{Ar} = K - \sum_{i=0}^{r-1} C_{Ai}$. Core *r* will generally spend a partial time in idle state, in particular its idle cycles will be: $C_{Ir} = T f_{ckr} - C_{Ar}$. We can now calculate the total energy given by Solution B: $E_{tot} = \sum_{i=1}^{r-1} C_{Ai}D_{Ai} + C_{Ar}D_{Ar} + C_{Ir}D_{Ir} + \sum_{i=r+1}^{N} C_{Ii}D_{Ii}$. Solution B suggests executing the total amount of cycles by exploiting the available time T. Note that if the solution B is taken, it is not guaranteed that tasks (after allocation performed by the BP algorithm) complete exactly at time T, as it depends on task granularity.

6.2.4.1 A closed-form solution of the LP

In this section we proove that the LP formulation of Section 5.3.2.1 can be solved thourgh few computations because it features some key properties. This means that there is no longer need of an LP solver, furthermore the overall LP+BP policy can be applied at runtime as we will demonstrate on the experimental results.

The above outlined LP formulation features some properties that simplify its solution. The main observation is that these properties reduce the set of possible optimal LP solutions: They are characterized by a number of cores that are fully active, a number of cores that are fully idle and at most one core characterized by an incomplete utilization (i.e. only one core is used for a fraction of the frame time). In what follows we provide the mathematical formulation of the closed form solution of the LP.

We call:

$$D_{Ai} = \frac{P_{dynAi} + P_{lkgAi}}{f_{cki}} \quad D_{Ii} = \frac{P_{lkgIi}}{f_{cki}} \tag{6.1}$$

We can rewrite (5.2) like this:

$$E_{TOT} = \sum_{i=1}^{N} D_{Ai} C_{Ai} + \sum_{i=1}^{N} D_{Ii} C_{Ii} \tag{6.2}$$

We can introduce an additional variable t expressing the execution time, replacing the first constraint in (5.10) with:

$$\frac{C_{Ai} + C_{Ii}}{f_{cki}} = t \quad \forall i : 1 \dots N \tag{6.3}$$

and the third one with $t \leq T$. Since

$$C_{Ii} = f_{cki} t - C_{Ai} \tag{6.4}$$

we can rewrite (6.2) like this:

$$E_{TOT} = \sum_{i=1}^{N} \left(D_{Ai} - D_{Ii}\right) C_{Ai} + t \sum_{i=1}^{N} D_{Ii} f_{cki} \tag{6.5}$$

We now define:

$$\begin{array}{l} x_i = C_{Ai}/f_{cki} \\ p_i = \left(D_{Ai} - D_{Ii}\right) f_{cki} = P_{dynAi} + P_{lkgAi} - P_{lkgIi} \\ q = \sum_{i=1}^{N} D_{Ii} f_{cki} = \sum_{i=1}^{N} P_{lkgIi} \end{array} \tag{6.6}$$

and rewrite the LP formulation as follows:

$$\begin{array}{l} \min_x \sum_{i=1}^{N} p_i x_i + qt \\ \\ \begin{cases} \sum_{i=1}^{N} f_{cki} x_i = K \\ 0 \leq x_i \leq t \leq T \qquad \forall i : 1 \dots N \end{cases} \end{array} \tag{6.7}$$

Note that x_i expresses the activity time of core i. Note also that the presence of term qt stresses the fact that there may be a gain in terminating all tasks before the deadline T, which is indeed the case in Solution A.

We assume that LP (6.7) has a feasible solution, which is easily seen to hold if and only if $\sum_{i=1}^{N} f_{cki}T \geq K$. Moreover, recall that the cores are ordered by increasing values of $\frac{p_i}{f_{cki}} = D_{Ai} - D_{Ii} = \frac{P_{dynAi}+P_{lkgAi}-P_{lkgIi}}{f_{cki}}$, i.e. the values of the cores: $\frac{p_1}{f_{ck1}} \leq \frac{p_2}{f_{ck2}} \leq \cdots \leq \frac{p_N}{f_{ckN}}$ are sorted and represent the penalty in energy for using a cycle of the core.

The following proposition states that there is an optimal solution of (6.7) in which either (a) there exists a core s such that either cores $1 \ldots s$ are always active during the execution time and cores $s+1 \ldots N$ are always idle, or (b) the execution time is equal to T, and there exists a core r such that cores $1 \ldots r-1$ are always active during the execution time, core r is partly active and partly idle, and cores $r+1 \ldots N$ are always idle. In fact, the proposition gives a closed form expression of the optimal solution depending on the specific values of K and T.

Proposition 1. *Let $s \in \{1, \ldots, N\}$ be the largest index such that $\frac{p_s}{f_{cks}} \leq \frac{\sum_{i=1}^{s-1} p_i + q}{\sum_{i=1}^{s-1} f_{cki}}$, with $s = 1$ if no index satisfies the property. Given K and T, the optimal solution x^*, t^* of LP (6.7) is the following:*

(a) if $\sum_{i=1}^{s} f_{cki}T \geq K$, then $t^* = \frac{K}{\sum_{i=1}^{s} f_{cki}}$; $x_i^* = t^*$ for $i = 1 \ldots s$; $x_i^* = 0$ for $i = s+1 \ldots N$;

(b) otherwise $t^* = T$; $x_i^* = T$ for $i = 1 \ldots r-1$; $x_r^* = \frac{K - \sum_{i=1}^{r-1} f_{cki}T}{f_{ckr}}$; $x_i^* = 0$ for $i = r+1 \ldots N$, where $r > s$ is such that $\sum_{i=1}^{r-1} f_{cki}T < K$ and $\sum_{i=1}^{r} f_{cki}T \geq K$.

Given a solution x^*, t^* to (6.7), the corresponding solution C^* to (5.10) is given by $C_{Ai}^* = f_{cki}x_i^*$ and $C_{Ii}^* = f_{cki}t^* - C_{Ai}^*$ for $i : 1 \ldots N$. According to the above proposition, LP (6.7), and therefore also LP (5.10), can be solved by a simple arithmetic calculation involving T and R, given that the partial sums $\sum_{i=1}^{j} f_{cki}$ can be computed once for all for every $j \in 1 \ldots N$.

6.2.4.2 Example

Once we have calculated the cycle budgets of each core able to execute the workload spending the minimum energy while meeting the time constraint, we have to solve the problem to allocate the tasks onto the cores. In particular the problem can be now formulated as follows: Find the best way to fit the task cycles into the core cycle budgets. In general this can be solved using a BP algorithm; however some customization to the specific requirements of multimedia applications must be performed. In particular, since the solution does not generally produce an exact match between the cycle budget of each core and the cycles of the tasks that are allocated on it, we must handle this case with minimum impact on energy and QoS.

The algorithmic details have been presented in Section 5.3.2.2 , here we give an example. Let us suppose to have the independent tasks represented by the following cycles $\{200, 220, 170, 70, 300\}$. Let us consider to have to execute them in 0.80 μsec.

The hypothesis is to have the 3-core degraded platform whose parameters are shown in Table 6.1. In the table the cores are already sorted in relation with their quantity $D_A - D_I$.

The closed form suggests the two solutions represented in Table 6.2. We point out that Solution A, which use all cores in order to execute all cycles in a minimum possible time, consumes 40 nJ. The minimum time of 0.37 μsec is given by dividing all cycles, which are 960, by the sum of the all core clock frequencies.

Solution B proposes to use for all the available time, which is the time constraint of 0.80 μsec, the core 3, while the core 1 for a partial time, and finally the core 2 never. The best solution is given by B because its expected energy is smaller. Then, we formulate the BP problem which tries to fit the task cycles into the bin C_{Ai}**s**.

The BFD algorithm indicates to sort the tasks from the largest one to the smallest one, while the cycle budgets from the shortest one to the largest one. We sorted tasks and cores as illustrated in Table 6.3 at *Starting Point*. For each core we will also take into account the execution time when the tasks are allocated on it, which is given by the cycles of the allocated tasks divided by the clock frequency of the core.

Now, each task will be assigned on the first core which has the minimum cycle budget to contain it. Once we allocate the current task, we remove it from the list and we calculate the remaining cycle budget for the core. This completes the first part of the BFD algorithm.

For instance, the first task, whose identification number is 5, is too large to be allocated on both core 2, which has a budget of 0 cycles, and core 1, which has a budget of 280 cycles. The task will be allocated on core 3 which has a budget of 680 cycles. We remove from 680 the 300 cycles and we will obtain 380 cycles while the execution time is $300/850 = 0.35\mu sec$. We sort the cores again and remove the first task, obtaining the situation shown in Table 6.3 at *1st assignment*.

Proceeding in this way we come to the situation shown in Table 6.3 at the *4th assignment*, where there are no cores with enough remaining cycle budget to execute task 4. The first step of the algorithm terminates with 4 allocated tasks and one missing. In this case the standard solution following the BFD algorithm allocates the task on the core which exceeds its cycle budget with the minimum number of cycles, which is core 1.

The customized version, on the other side, checks how long the cost in terms of execution time of the exceeding cycles is. Matching time constraint has a higher priority than reducing the exceeding cycles (which means being closer to the energy optimal solution computed by LP). Having the cores different speeds, these two metrics do not lead to the same result. As such, this version inspects the cores from the last one (the one which leads to the smaller cycle overflow) and selects for allocation the first core allowing to match the deadline. This leads to a trade-off between QoS and energy consumption. This completes the second part of the BFD algorithm.

Applying this method, task 4 will be allocated on core 1, leading to an execution time of $0.24 + 70/900 = 0.31\mu sec$ which is shorter than the time constraint $0.80\mu sec$. It can be easily verified that other allocations would lead to a deadline miss. It must be noted that in this case the custom solution corresponds to the standard one, but this is not true in general.

The final situation is shown in Table 6.3 at the *5th assignment*, and the final task allocation in Table 6.4. Core 3 will work for 0.78 μsec, core 2 will work for 0.31 μsec and stay in idle for the remaining time (0.47 μsec), finally core 1 will always stay in idle (0.78 μsec).

The expected execution time is 0.78 μsec with respect to 0.80 μsec provided by the LP solution, in addition the expected energy consumption is $E = \sum_{i=1}^{3} t_{Ai} D_{Ai} + \sum_{i=1}^{3} t_{Ii} D_{Ii} = 40nJ$ with respect to $35nJ$ provided by the LP solution. The energy slightly increases because core 1 has to execute 290 cycles instead of 280 as LP recommends, even if core 3 has to execute 670 cycles instead of 680 (core 1 has larger energy consumption per cycle).

In case there are no cores able to execute the remaining tasks within the time constraint, we used another customization to minimize the slack beyond the deadline. This concludes the third step of the BFD algorithm. Note that this part is not executed if the platform is designed with enough conservative time margins for the target applications.

Core Id	F_{ck} [Mhz]	P_{dynA} [mW]	P_{lkgA} [mW]	P_{lkgI} [mW]	D_A	D_I	$D_A - D_I$
3	850	21	9	0.2	3,53E-02	2,35E-04	3,51E-02
1	900	26	10	0.2	4,00E-02	2,22E-04	3,98E-02
2	870	28	14	0.2	4,83E-02	2,30E-04	4,81E-02

Table 6.1: A 3-core degraded platform example. Cores are sorted with respect to $D_{Ai} - D_{Ii}$.

	Execution Time $[\mu sec]$	CA3	CA1	CA2	CI3	CI1	CI2	Energy
solA	0.37	311	330	319	0	0	0	40
solB	0.80	680	280	0	0	440	696	35

Table 6.2: Core cycle budgets of the two LP candidate solutions with the expected energy consumptions.

Starting point				
Task Id	Task Cycles	Core Id	Remaining Cycle Budget	Allocated Execution Time μsec
5	300	2	0	0.00
2	220	1	280	0.00
1	200	3	680	0.00
3	170			
4	70			
1st assignment				
Task Id	Task Cycles	Core Id	Remaining Cycle Budget	Allocated Execution Time μsec
2	220	2	0	0.00
1	200	1	280	0.00
3	170	3	380	0.35
4	70			
...				
4th assignment				
Task Id	Task Cycles	Core Id	Remaining Cycle Budget	Allocated Execution Time μsec
4	70	2	0	0.00
		3	10	0.78
		1	60	0.24
5th assignment				
Task Id	Task Cycles	Core Id	Remaining Cycle Budget	Allocated Execution Time μsec
		1	-10	0.31
		2	0	0.00
		3	10	0.78

Table 6.3: LP+BP example.

6.2.5 Min and Max energy techniques

The energy spread across the cores of a given platform in terms of D_{Ai} is generally different with respect to the different extracted platforms. Given a workload, the maximum energy and the minimum energy that the platform can consume depend on the spread across the D_{Ai}

	task 1	task 2	task 3	task 4	task 5
core 1		X		X	
core 2					
core 3	X		X		X

Table 6.4: Final task allocation.X indicates that the task on the column is assigned on the core on the row.

values. Then, to understand how much a policy can save energy running an application on a given platform, the normalization of the energy between the minimum and the maximum values can help us. To know the minimum energy E_{min} and the maximum energy E_{max} to execute a workload on a given platform we introduce two additional policies.

The Min Energy technique finds E_{min} in the following way. There are two candidate solutions: a) all tasks are executed by the core at minimum $D_{Ai} - D_{Ii}$; b) we use all cores allocating on them the tasks in the way to have minimum possible idle time (i.e. we can choose Solution A of the closed form LP and then solve the BP problem).

We will choose the solution at minimum energy. Even if the b) solution uses all cores, there exists a possibility that the execution time is so short to have the minimum energy consumption.

The Max Energy technique finds E_{max} in the following way. There are two candidate solutions: a) all tasks are executed by the core at maximum $D_{Ai} - D_{Ii}$; b) the same of the Min Energy.

We will choice the solution at maximum energy.

6.3 Experimental results

The platform target we refer of these experiment is xSTsim which is described in Section 2.3. In expertiments we used the cycle-accurate simulator. We integrated the variability model in the target platform simulator to assess the impact of variations on the running software and enable the study of system level software policies. Details are reported in Section 4.3.1.

Our experiments are based on two different approaches of variability injection. In the first case, we exploit the VAM tool which reads the netlist of the cores of the platform and generates the configuration files of the simulator. This is depicted on the upper side of Figure 6.3(a).

The second approach, which has been used to evaluate the benefits of the policy as a function of the entity of variation, exploits a synthetically generated set of configuration files. In particular we modulated each core parameter according to a normal distribution. That is, varying the standard deviation of the normal distributions we generated different sets of configuration files. This approach is depicted in the bottom part of Figure 6.3(a).

We used two different representative multimedia algorithms as testcases: An MPEG2 decoder, and an Integral Image algorithm which are described in Section 3.3.

For the MPEG2 decoder, the workload is a video clip with 25 frame per second (fps), length 1 second, resolution 720×576. We conducted experiments by dividing the workload in 4, 8, and 12 tasks. The frame ratio of 25 fps implies a deadline of 40,000 μsec to decode each frame.

For the Integral Image, the workload is a queue of 25 matrices of 96×96 unsigned integer elements. This workload has been divided in 4, 8, and 12 tasks. We set a deadline of 4,500 μsec to compute the integral image of each matrix.

In order to provide the task sets to the policies we had to execute once the applications on the simulator platform before the tests (*profiling step*). The plug-in stored the cycles needed by each task for each frame/matrix. Since activity cycles do not change with the parameter variation, we could use the nominal platform for the profiling. We needed also to take into account the stall cycles. We made the realistic assumption that the ratio between the stall cycles and the activity cycles does not depend on the core and the specific frames/matrices. During the profiling step we evaluated the maximum ratio between the stall and the activity cycles among the cores and the frames/matrices.

We rearranged the formulation shown in Section 5.3.2.1 by adding to the parameters regarding activity cycles the contribution due to stall cycles. In the new formulation, we take into account the dynamic power consumption in stall state of core *i*, namely P_{dynSi}, and the leakage power consumption in stall state of core *i*, namely P_{lkgSi}. Referring to (5.2), and considering r as the ratio between stall cycles and activity cycles, which we suppose to be constant for each core, we adjust the first term of the summation as follows:

$$\begin{aligned} &\frac{(P_{dynAi} + P_{lkgAi})\, C_{Ai} + (P_{dynSi} + P_{lkgSi})\, C_{Si}}{f_{cki}} = \\ &= \frac{(P_{dynAi} + P_{lkgAi} + (P_{dynSi} + P_{lkgSi})\, r)\, C_{Ai}}{f_{cki}} \end{aligned} \tag{6.8}$$

In this way we take into account the power consumption in stall state by adapting the LP formulation by simply adjusting the coefficients of the $C_{Ai}s$ variables. Regarding the time we adjust the deadline in the second constraint of (6.7) as follows in (6.9).

$$x_i = \frac{C_{Ai}}{f_{cki}} \leq \frac{t}{1+r} \leq \frac{T}{1+r} \tag{6.9}$$

The algorithm can produce the C_{Ai} cycle budget for each core i. Likewise, we give to the BFD algorithm the same adjusted time constraint shown in (6.9) as input. We experienced through the following experiment that this assumption holds.

The upper side of Figure 6.3(b) shows the profiling approach: 1) decode the current frame/-matrix; 2) store for each task the needed activity cycles. During this phase, the plug-in writes into a text file the information about the task set and the stall cycles. On the bottom of the same figure it is represented how the test is performed by reading the text file.

fVAR platform							
	FCK	PdynA	PlkgA	PdynS	PlkgS	PdynI	PlkgI
xpe1	1.00	0.77	0.45	0.52	0.43	0.00E+00	7.98E-05
xpe2	0.81	0.85	0.50	0.57	0.48	0.00E+00	7.98E-05
xpe3	1.00	1.02	0.49	0.69	0.47	0.00E+00	7.98E-05
xpe4	0.84	0.87	0.50	0.59	0.48	0.00E+00	7.98E-05
xpe5	1.00	0.99	0.49	0.67	0.47	0.00E+00	7.98E-05
xpe6	0.84	0.83	0.49	0.57	0.47	0.00E+00	7.98E-05
xpe7	1.00	0.99	0.49	0.68	0.47	0.00E+00	7.98E-05
xpe8	0.84	0.80	0.48	0.54	0.46	0.00E+00	7.98E-05
pVAR platform							
	FCK	PdynA	PlkgA	PdynS	PlkgS	PdynI	PlkgI
xpe1	0.88	0.71	0.46	0.48	0.44	0.00E+00	7.98E-05
xpe2	1.00	1.03	0.50	0.70	0.48	0.00E+00	7.98E-05
xpe3	1.00	0.77	0.45	0.52	0.43	0.00E+00	7.98E-05
xpe4	1.00	1.04	0.50	0.71	0.48	0.00E+00	7.98E-05
xpe5	0.84	0.80	0.48	0.54	0.46	0.00E+00	7.98E-05
xpe6	1.00	1.04	0.50	0.71	0.48	0.00E+00	7.98E-05
xpe7	0.84	0.81	0.48	0.55	0.46	0.00E+00	7.98E-05
xpe8	1.00	1.05	0.50	0.71	0.48	0.00E+00	7.98E-05

Table 6.5: Degraded platforms.

6.3.1 Results

In this part of the results, we consider variability injected by the VAM tool. Among the generated degraded platforms we chose two of them having the largest spread in terms of performance and power, called *fVAR* and *pVAR* respectively. As such, *fVAR* can be considered a worst case in terms of performance degradation, while *pVAR* is the worst case in terms of power. The characteristics of their cores are described in Table 6.5. As nominal values for the

	MPEG2 Decoder									
	fVar					pVar				
tasks	RNM	RF	RP	RN	LP+BP	RNM	RF	RP	RN	LP+BP
4	1.00	0.40	0.88	0.40	0.40	0.48	0.40	0.96	0.64	0.40
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
12	0.12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Integral Image									
	fVar					pVar				
tasks	RNM	RF	RP	RN	LP+BP	RNM	RF	RP	RN	LP+BP
4	1.00	0.00	1.00	0.00	0.00	1.00	0.00	1.00	1.00	0.00
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 6.6: Deadline miss rate $[0..1]$ using fVAR and pVAR platforms.

parameters (i.e. longest path delay, dynamic and leakage power consumption for the different work states), we used the values for the xPE at 32nm CMOS technology [1].

We compared, for the two platforms and the two applications, the energy consumption to decode 25 frames and to integrate 25 matrices respectively, and the number of frames/matrices execution missing the time constraint, that is the deadline miss rate.

In particular we normalized the energy consumption in the following way : $(E - E_{min})/(E_{max} - E_{min})$, where E is the energy consumption related to a generic execution, while E_{min} and E_{max} are respectively the minimum and the maximum possible energy that can be consumed executing the same workload on the same platform without taking into account the time constraint.

We expressed the deadline miss rate as the ratio between the number of missed frames/-matrices on the number of total frames/matrices. Then, both the energy and the deadline miss ratio can assume values between 0 and 1.

We compared the different policies presented in Section 6.2. Hereafter we indicate *RF* for Rank Frequency, *RP* for Rank Power, *RN* for Rank Energy, and *LP+BP* Linear Programming + Bin Packing. Moreover, we made experiments using a Random technique which randomly allocates the tasks onto the cores; we indicate it as *RNM* on the tables.

In Figure 6.3(c), we represented in X-Y plots the average time to execute one frame/matrix (X-axis) and the normalized energy consumption (Y-axis). The best condition is therefore the bottom-left area, however the QoS requirements impose to spend less than 40,000 μsec for decoding each frame for the MPEG2 decoder and less than 4,500 μsec for the Integral Image (dashed vertical line on the graphs show the time constraints).

[1]The tables contain normalized values because of confidentiality reasons

For 4 tasks, LP+BP is always one of the policies that realizes the minimum execution time on average, indeed spending more energy than the policies that produce execution times being further from the deadlines.

For 8 and 12 tasks, all policies show execution times shorter than the time constraints. The comparison policies show smaller execution time but more energy consumption than LP+BP that meets the deadline in all cases. In conclusion LP+BP saves as much energy as possible with minimum impact on QoS requirements.

For the pVAR platform the energy consumption is generally higher than in the case of the fVAR platform, this depends on the difference between the maximum and minimum energy that the platform can consume. In fact, regarding the energy per cycle, fVAR has a spread of 0.79, instead pVAR has a spread of 0.58 (see Table 6.5).

In Table 6.6 we show the deadline miss ratio for both the applications and both the platforms.

Using 4 tasks, the workload is characterized by few large tasks and it becomes hard to execute them within the time constraint. In this case, RF can discover the fastest task allocation, and then it realizes the lowest deadline miss rate. LP+BP always finds out that to reach the lowest deadline miss rate the solution is to maximize the performance. RF and LP+BP always consume the same energy except for the pVAR-MPEG2 case where LP+BP is better (see Figure 6.3(c)).

Increasing the number of tasks, all the Rank policies meet all the deadlines, but LP+BP is also able to better exploit the available time producing the lowest energy consumption. In particular, in referring to the lowest energies of the comparison techniques, LP+BP can save up to the 33% of energy.

In order to evaluate the impact of variability-aware allocation strategy with respect to a variability un-aware one, we compared the normalized energy as a function of the entity of parameter variations. Results are shown in Figure 6.3(d). Here we compare LP+BP as well as the rank techniques with an algorithm that assumes that all cores run to their nominal parameters. We used LP+BP assuming the nominal platform instead of the actual variability-affected platform. This algorithm is referenced as *NOM*. The study has been conducted by varying the parameters according to a synthetically generated normal distribution.

In order to compare the energy consumption of the several task allocation techniques in relation with the increasing of the variation we chose the following levels of standard deviation: 0.0 (i.e. nominal platform), 0.5 and 1.0.

For each non-zero standard deviation we extracted five different degraded 8-core platforms and we averaged the results in normalized energy, deadline miss rate, time to execute one frame/matrix. Moreover we evaluated the functional yield expressed as the percentage of the number of platforms that executed the test applications realizing no missed deadline.

We conducted experiments using configurations of 4, 8, and 12 tasks. If the number of the tasks increases, it is generally easier to find a task allocation meeting the time constraint.

The results clearly show how LP+BP reduces energy consumption with respect to the NOM policy (up to 65%) and also in almost all the cases leads to lower energy than the other policies. In general, all the policies are equivalent using the nominal platform (standard deviation $= 0$). Clearly, by construction the LP+BP for the nominal platform performs as the NOM policy. It must be noted that in case of 12-tasks configuration, for both applications LP+BP slightly reduces energy consumption also in the nominal platform case. We observed that this is because its allocation, which tends to reduce the utilized cores, reduces the stall cycles as a side effect.

It must be noted that, besides the case of MPEG2 in 4-task configuration (upper-left side of the figure), all the policies lead to an energy consumption closer to the maximum for the nominal configuration (i.e. standard deviation $= 0$). In the nominal case cores are all equal and for this reason most of the policies tend to use all of the available ones. The case of MPEG2 4-tasks has a max energy corresponding to a configuration where only a single core is used. With respect to this reference value, policies lead to an improvement also in the nominal platform.

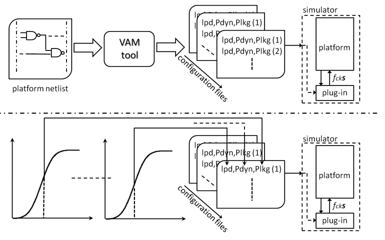
Another consideration concerns the fact that the normalized energy decreases from 0 to 0.5 of standard deviation and increases from 0.5 to 1. The reason is that the normalization range is not the same for all the standard deviation values, since max and min energy are recomputed depending on the actual platform values.

Finally, in the case of MPEG2 it is more difficult to meet the deadlines, then the normalized energy is higher on average.

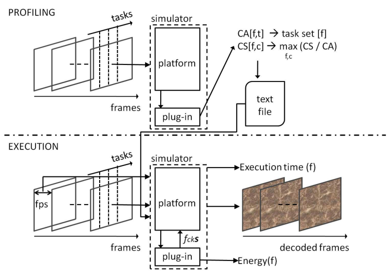
By looking at the functional yield (upper side of Table 6.7) this is lower than 100% using a 4-task configuration. In all the other cases all policies realize the 100% of yield. Similarly, the deadline miss rate (lower side of Table 6.7) is larger than 0% only in the 4-task configuration. In this case, RF and LP+BP always achieve the maximum yield and the minimum deadline miss rate. The reason is that RF always tries to maximize the performance and LP+BP finds, in this case, that the only solution to minimize the deadline miss rate is to use the faster cores. They also produce the same energy consumption.

Moving to 8-task and 12-task division, all policies realize the 100% of yield and LP+BP always consumes less energy.

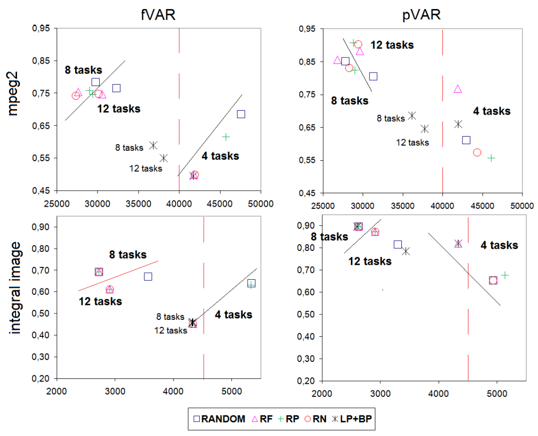
In conclusion, applying a variability-aware task allocation technique improves energy consumption and functional yield with respect to a variability un-aware policy and alternative variability-aware techniques.



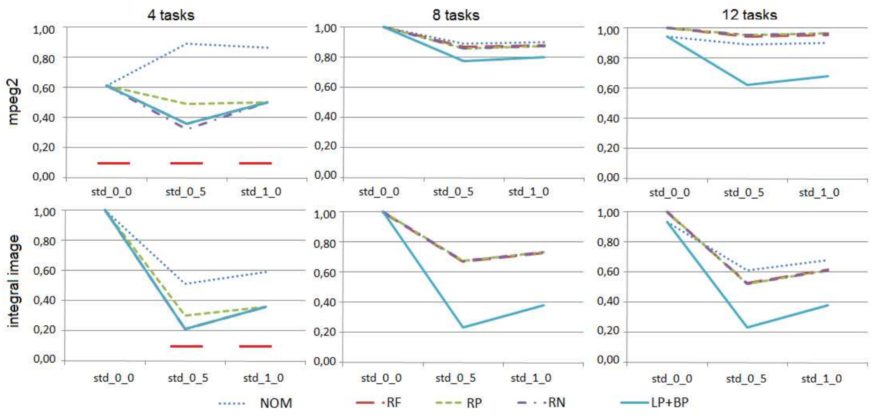
(a) Variability injection in core platform parameters. The first method uses VAM (upper side). The second method uses synthetic normal distributions (lower side).



(b) Task activity cycles and stall ratio profiling (upper side). Execution of test by using the profiled information (lower side). t: task, c: core, f: frame.



(c) Energy $[0..1](Y$ axis) vs. Execution Time $[\mu sec]$ (X axis). Dashed lines indicate the time constraints.



(d) Energy $[0..1](Y$ axis) vs. Variation Level $[standard\_deviation](X$ axis). The segments below the plots indicate different deadline miss rates at that point.

Figure 6.3: a)Variability injection b)Profiling c)Energy vs. Execution Time d)Energy vs. Variation Level

	Functional Yield [%]									
	MPEG2 Decoder					Integral Image				
STD	NOM	RF	RP	RN	LP+BP	NOM	RF	RP	RN	LP+BP
0.0	0	0	0	0	0	100	100	100	100	100
0.5	0	0	0	0	0	0	40	0	40	40
1.0	0	0	0	0	0	0	40	40	40	40
	Deadline Miss rate [0..1]									
	MPEG2 Decoder					Integral Image				
STD	NOM	RF	RP	RN	LP+BP	NOM	RF	RP	RN	LP+BP
0.0	0.40	0.40	0.40	0.40	0.40	0.00	0.00	0.00	0.00	0.00
0.5	0.47	0.37	0.40	0.39	0.37	1.00	0.60	1.00	0.60	0.60
1.0	0.47	0.37	0.37	0.37	0.37	1.00	0.60	0.60	0.60	0.60

Table 6.7: Normal distribution for parameter variations. 4 tasks.

	3 cores	6 cores	9 cores	12 cores	16 cores
LP closed form	135	268	411	567	793
BFD to solve BP (8 tasks)	4	14	43	73	112
BFD to solve BP (16 tasks)	9	25	43	75	106
BFD to solve BP (32 tasks)	24	57	102	160	281
BFD to solve BP (128 tasks)	181	286	451	728	1153

Table 6.8: LP closed form algorithm and BFD algorithm costs [μsec]. The algorithms are executed by ST231 clocked at 900MHz.

6.3.2 LP+BP policy execution time

The previous results were obtained by applying LP+BP at run-time. In particular, during the xPEs execution, GPE solves the combined LP+BP problem in shorter time. In this part we provide a detailed characterization of LP+BP execution time, highlighting that the algorithm can be solved in linear time with respect to the number of cores.

The LP-BP policy runs on the host core, which is an ST231 processor clocked at 900 MHz. We report in Table 6.8 the execution time of the policy for an increasing number of cores and for various task configurations. Overall, LP+BP overhead linearly increases with the number of cores. In all of the configurations the policy overhead is one order of magnitude lower than the execution time of the slice decoding or IDCT for the MPEG2 application (at least 7,000 μsec), then it can be transparently executed on the host while the slaves perform the decoding tasks. In case of Integral Image, which is a simpler algorithm, this is true (for the considered matrix size) up to 9 cores and 16 tasks. However, it must be noted that for larger matrices, for which large parallelism is meaningful, this break-even point likely increases.

As a final consideration, the estimated stall cycles did not impact the LP+BP effectiveness;

in particular the actual stall cycles always have been less than the estimated ones.

6.4 Summary

The algorithm we propose in this chapter, which implements the LP+BP policy, needs a linear computation time and therefore it can be applied on-line. We demonstrated the effectiveness of our approach through a comparison with state-of-the-art policies. In our experiments we used representative multimedia streaming applications. We focused on the xSTsim industrial multicore platform provided by STMicroelectronics conducting our experiments on the xSTream cycle-accurate simulator. LP+BP can save up to 33% of energy with respect to the state-of-the-art policies and 65% of energy with respect to one variability-un-aware task allocation policy while providing better QoS.

Chapter 7

High-reliability multicore platforms

7.1 Overview

In Chapters 5 and 6 we studied the effects caused by static variations in terms of energy and performance in MPSoCs. Here, we want to move our attention to the lifetime reliability of the systems.

To cope with process variations which cause performance uncertainty and unbalancing in MPSoCs, countermeasures at various levels have been developed, ranging from transistor level, architectural and system software level. Software approaches can be very effective because they can adapt to wear-out and temperature dependency. There are several hardware techniques that can be used to make software aware of chip degradation, namely sampling based detection [10, 24], periodic testing, error correction and detection circuitry [68]. Once this information is made available at the software level, a common purpose of various approaches recently proposed is to provide wanted performance and match real-time constraints through statistical scheduling [82] or learning algorithms [83].

The main challenge of these techniques in a multiprocessor systems is to cope with the non-uniform distribution of critical path delay variations. To handle this heterogeneous delay distribution, each core can be clocked with a different frequency, thus increasing the need of synchronization for intra-core communication. A more conservative approach is to run all the cores at the same clock frequency dictated by the slowest core [63]. In both cases the aging effect will deviate the system from the starting condition, affecting the expected lifetime and its distribution between the cores. In this scenario, some cores will have a lower lifetime expectation than others, thus decreasing reliability and predictability of the system.

The objective of the work presented in this chapter is to mitigate the impact on lifetime uncertainty and unbalancing among the cores. To this purpose, we developed an idleness distribution policy that increases core expected lifetimes by duty cycling their activity. The idleness is distributed to equalize the expected lifetime of each core to a target value, imposed by the system designer or by the user. Since the actual impact on performance depends on the task model running on the target multicore system, in this work we consider three representative task models, namely batch execution, playout and streaming, for which we evaluate the impact of the policy on the performance level. The proposed approach is based on variability information that can be provided at run time by variability monitors, that are likely to be embedded in next generation MPSoC designs.

Idleness distribution is conceptually similar to clock frequency scaling. Even if our implementation exploits idleness, the same strategy can be coupled with a frequency scaling approach. In both cases the core operates on average at a lower average speed and reduce overall switching activity with a positive effect on lifetime. However, idleness distribution is more profitable because it does not require separate frequency domains. Nevertheless, frequency scaling coupled with voltage scaling can provide considerable dynamic power savings. However, for this to be possible separate voltage domains with associated expensive level shifters are needed. Provided that voltage islands are present, also idleness distribution policy contributes to power reduction as long as the core allows to be power-gated when idle.

The contributions provided in this chapter can be summarized as follows. First, we propose an on-line adaptive strategy for increasing MPSoC tolerance to non-uniform wear-out due to variations. The methodology is innovative as it is focused on aging tolerance to improve system lifetime rather than on recovery of performance lost because of wear-out. Moreover, it is not based on static task characterization, but on on-line execution time and wear-out monitoring. Second, we propose an efficient implementation based on a look-up table that directly correlates target lifetime with idleness distribution. Third, we studied the impact of the aging tolerance policy on performance for various representative task models, demonstrating its negligible overhead and adaptation to different workload characteristics.

The rest of the chapter is organized as follows. Section 7.1.1 reviewes the recent works in this filed, Section 2 discusses the variability model considered in this work. Section 3 presents the hardware and software infrastructure. Section 4 describes the proposed policy and Section 5 presents experimental results.

7.1.1 Related work

In [81] a statistic scheduling approach is proposed to mitigate the impact of parameter variations in a multiprocessor platform. The strategy assumes that task executions are statistic rather than deterministic. A new metric is introduced called performance yield, defined as the probability of the assigned schedule meeting the timing constraints. This work demonstrates that using a statistical scheduling approach consistently improves the performance yield. Wear-out factors are not considered in this work. As a result, the proposed policy is based on a static estimation of task execution times and variability information.

Wear-out effects are considered in [63], where authors present a scheduling approach which is aimed at recovering the performance impact due to non-uniform chip degradation. They propose an integer linear programming method to determine an optimal scheduling for streaming applications. Differently from previous work, variability effects on interconnect and memories are also considered in the optimization problem. Moreover, task migration is also considered as solution to handle the time dependent effect of wear-out.

These papers state the effectiveness of software and system level approaches to variability issues and we want to complement previous techniques by presenting a fully on-line and workload adaptive strategy aimed at improving MPSoC aging tolerance instead of focusing only on performance. It is based on the on-line estimation of idleness and variability as well as wear-out conditions. As such, it does not exploit task pre-characterization. The proposed technique can be applied to workload based on a variable number of tasks. Because of these characteristics, our on-line approach to lifetime improvement could be applied with static techniques to achieve an effective performance vs. lifetime trade-off.

7.2 Idleness constraints

The relationship between the degradation of the critical path delay and actual lifetime for each core depends on two factors. First of all, an aging function which expresses the delay critical path degradation as a function of time. We refer to a per-core aging function modulated by the core activity. This function has been extrapolated by the NBTI model and is shown in Section 4.2.1. This means that the delay critical path does not degrade when the core is idle. Moreover we can increase the expected system core lifetime by putting it in some standby state when idle, which is a realistic assumption for state of the art SoCs. The second factor is the effectiveness of the error correction circuitry that is possibly embedded in the architecture. The wear-out effect causes more and more severe timing violations and an increasing number of paths violating them as the time elapse, thus increasing the percentage of corrected errors.

The error correction circuitry is able to correct up to a certain error rate. If this rate is reached, the core cannot be recovered and thus it fails. For this reason, the expected lifetime can be computed as the time to reach this maximum error rate. Error correction systems can be exploited as monitor of the aging process. Using an aging model, it is possible to determine the expected lifetime based on the amount of corrected errors. In this way, our policy can directly use the lifetime information to know how much idleness is needed to match a given target lifetime requirement. This opportunity is depicted in Figure 7.1. Starting from an initial expected lifetime (t_{max}) which is achieved with 100% core activity, by playing with idleness it is possible to increase the lifetime up to a target value t_{lf}. The dashed line represents the activity duty cycling performed by inserting idle periods between task executions. We assume that the system is required to match a lifetime requirement for the whole system and we play on idleness distribution of each core in order to increase the expected lifetime to match the target one.

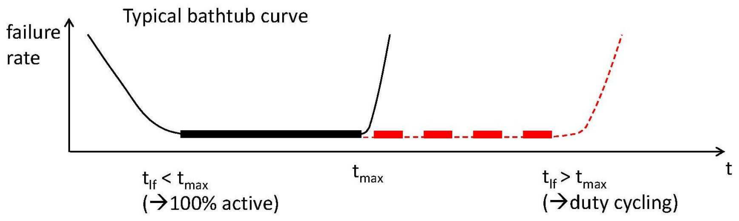


Figure 7.1: Relationship between idleness and core lifetime.

7.2.1 Platform model and software infrastructure

The platform we refer is xSTsim provided by STMicroelectronics and illustrated in Section 2.3. We used for our experiment the cycle-accurate simulator.

The software organization of our system is composed by support functions for task loading, data communication and synchronization, statistic collection. All the cores load the same program, following a SPMD approach, where each core executes a different portion of the program depending on its identifier. The accelerator code contains all the possible tasks to be executed. Currently, dynamic loading of tasks is not supported. As such, to execute a certain task, cores have to jump to the related code portion, which is identified by a pointer. To control the execution on the accelerators, the master core changes the pointer depending on which task the accelerator has to run. Shared memory is used to exchange data among cores.

Batch execution model. In the batch execution model, the master core spawns a number of N independent tasks on the accelerators exploiting a non-blocking round-robin algorithm. The performance metric associated with this task model is the execution time that in this case is defined as the time between the allocation of the first task and the completion of the last allocated task. Input and output data are stored in local memories of accelerators.

Output rate-constrained execution model (playout). This model is representative of playout activity performed by audio or video decoders. Also in this case the master allocates tasks on the accelerators. Accelerators read input data from their local memories. Output data items are stored in a common output queue allocated in shared memory with access regulated by semaphores. A consumer task runs in one dedicated core which periodically picks one data item from the output queue. The associated performance metric is the output throughput. When the output queue becomes empty, the consumer will experience a deadline miss. As such, the performance constraint is represented by the output rate.

Input-output rate-constrained execution model (streaming). While in the playout model input data for accelerators are available on local memories, in streaming task model data are provided to the accelerators by the master core. This is a typical model for a videoconferencing application where the input data are provided by a video camera and accelerators performs video encoding. Another example is a video decoder application receiving compressed frame from the network. An interprocessor communication queue is used as buffer between master and accelerators. As in the playout model, an output queue is used to synchronize data communication with the consumer core. The associated performance metric in this case is not only the output throughput, but also input throughput. If the input queue becomes full, this means that accelerators are not able to handle the input data rate. The constraint on the output still applies also in this task model.

7.3 Adaptive idleness distribution policy

The master core is responsible of allocating tasks on the accelerators. For this reason, it is the most suitable place where to implement the idleness distribution algorithm. Since the distribution algorithm depends on the reading of variability monitors of each core. Our target platform is equipped with a register accessible from the master and all the cores where the percentage of corrected errors (also called error rate) can be read for each core.

Our policy computes a required amount of idleness for each core. In order to make the policy implementation independent from the type of runtime information available, the policy takes as input a required idleness for each core. A conversion module fills up a table with the idleness values computed starting from error rate statistics for each core. An aging model as described in Section 4.2.1 is used to compute the time required to reach the max_error_rate value assuming zero idleness, that we call t^i_{max}, where i indicates the $i-th$ core. For each core, the target amount of idleness for a generic $i-th$ core is defined as:

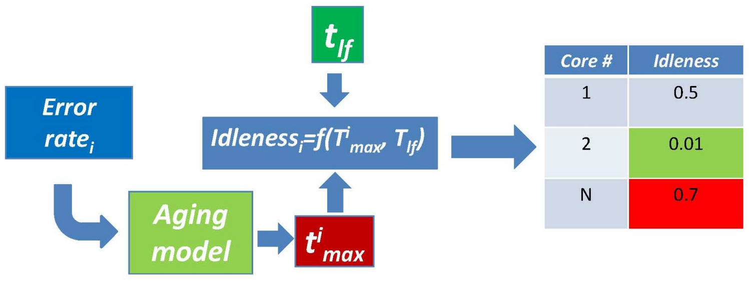


Figure 7.2: Implementation scheme of the adaptive idleness distribution policy.

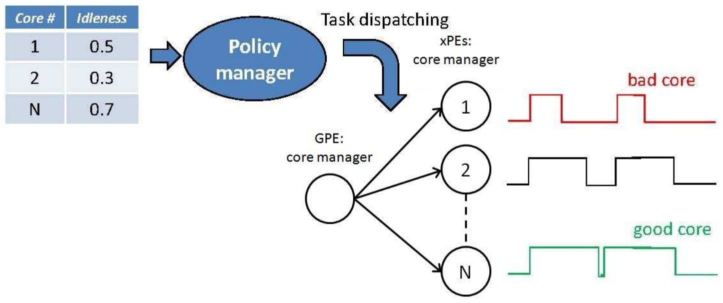


Figure 7.3: Adaptive idleness distribution policy description.

$$idleness = \begin{cases} 1 - \frac{t^i_{max}}{t_{lf}}, t_{lf} > t^i_{max}; \\ 0, t_{lf} < t^i_{max} \end{cases}$$

where t_{lf} is the system lifetime requirement and idleness is expressed as a number between 0 and 1, where 0 indicates full activity and 1 indicating no activity. Once the wanted average $idleness$ has been computed it is stored in a table as shown in Figure 7.2. Then, the master processor must perform the task allocation policy accordingly, as depicted in Figure 7.3. To achieve the wanted average idleness, our policy allocates idle periods between task executions for each accelerator. This implies that the wanted idleness is achieved on a time scale on the order of task execution times. This is reasonable as long as the expected lifetime is typically several orders of magnitude larger than task durations. Indeed, the implementation on a smaller timescale would imply pre-emption of tasks on the accelerators, introducing an unnecessary overhead. It must be noted that the proposed policy does not assume a specific aging model. The unique assumption is that additional idleness increases core lifetime.

As a result, the master core exploits hardware timers to update a data structure where task start and completion times are stored. After each task completes, its activity interval is computed. The idle period to be allocated is obtained by multiplication of the last activity period by the wanted idleness. After the idle period expires for a core, a new task is allocated to it.

It must be taken into account that cores must also perform task management (i.e. loading and completion notification) and synchronization operations (i.e. waiting on semaphores), as needed to implement a given task model. When computing the idle period to be allocated to each core, this additional activity is taken into account by our policy. This is possible because the master core has full visibility and monitoring capability of accelerator's activity. The idleness for each core is conservatively updated by the master core at each task completion, depending on monitor readings. However, frequency of updates can be configured. Experimental results show that the implementation overhead of this policy is negligible and that the wanted idleness is obtained with a very high accuracy.

7.4 Experimental results

The policy described in Section 7.3 requires software support mechanism for task activity monitoring and idleness computation, that could impact the accuracy of idleness distribution. For our experiments we considered two platform configurations, namely four and six accelerators. For each configuration, we considered three variability scenarios. Each variability scenario defines the number of cores affected by variability issues and the mapping of error rates on the cores. In our simulation platform, error rates are extracted from a Gaussian distribution. In our experiments we considered a static condition where monitor readings (i.e. variability conditions) are constant over time. However, we consider a worst case scenario where the master core reads the variability information at each task completion. The platform configurations and variability scenarios considered for our experiments are described in Figure 7.4.

It must be noted that minimum and maximum values of error rates are the same for the four and six core configurations. Benchmarks used for experiments are matrix multiplication kernels. To the purpose of characterization of idleness computation accuracy we measured the actual idleness and we compared it with the target one. The results we obtained about idleness accuracy, that are not shown here for space limitations, highlight that the maximum error in idleness assignment is within 0.1%, demonstrating the effectiveness of the proposed software infrastructure.

Batch Execution Results. The matrix multiplication benchmark $A \cdot B = C$ is composed by two phases. During the first one the matrix B is copied from shared memory to local memory, where A resides. In the second phase the actual matrix multiplication takes place. Results are

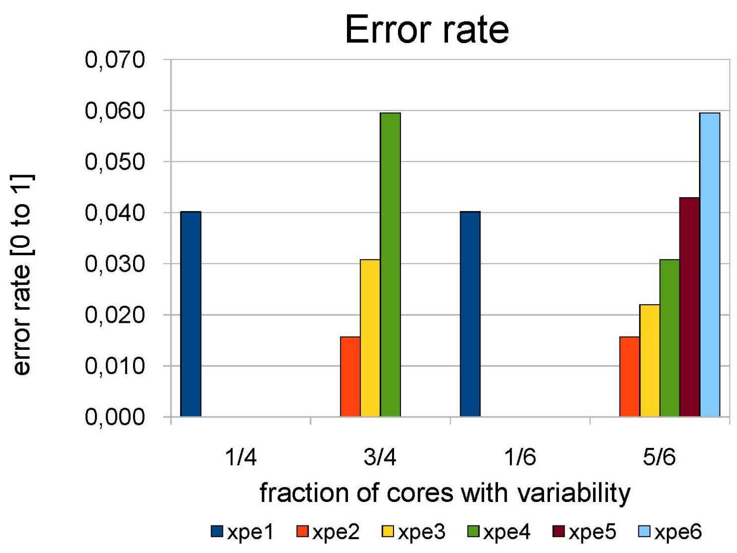
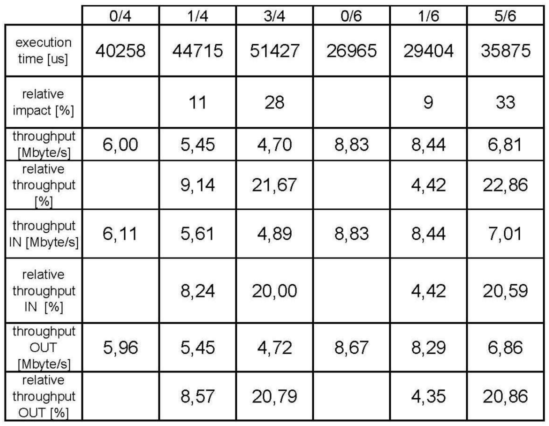


Figure 7.4: Variability scenarios. Error rates are mean values of a gaussian distribution.

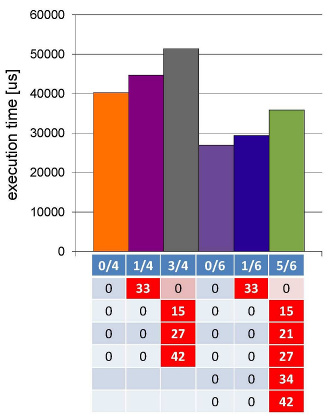
stored in the local matrix C.



	0/4	1/4	3/4	0/6	1/6	5/6
execution time [us]	40258	44715	51427	26965	29404	35875
relative impact [%]		11	28		9	33
throughput [Mbyte/s]	6,00	5,45	4,70	8,83	8,44	6,81
relative throughput [%]		9,14	21,67		4,42	22,86
throughput IN [Mbyte/s]	6,11	5,61	4,89	8,83	8,44	7,01
relative throughput IN [%]		8,24	20,00		4,42	20,59
throughput OUT [Mbyte/s]	5,96	5,45	4,72	8,67	8,29	6,86
relative throughput OUT [%]		8,57	20,79		4,35	20,86

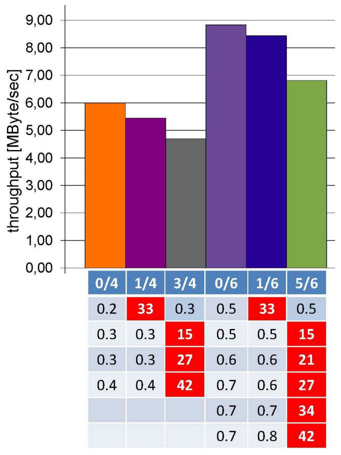
Figure 7.5: Relative impact of variability on performance for all the scenarios and configurations

Increasing the lifetime may have an impact on performance depending on the task model. For batch execution, performance hit lead to an increase of the overall execution time of N tasks, where N has been fixed to 60. Results are shown in Figure 7.6.a, where associated idleness values for each core are also reported for clarity. In Figure 7.5 the relative impact on execution time is shown. For each platform configuration (i.e. four vs. six cores), this has been computed using the scenario without variations as reference. By comparing the two platform configurations, it can be noted that the impact on execution time is proportional to the fraction



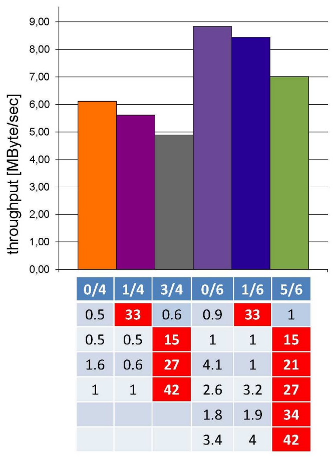
0/4	1/4	3/4	0/6	1/6	5/6
0	33	0	0	33	0
0	0	15	0	0	15
0	0	27	0	0	21
0	0	42	0	0	27
			0	0	34
			0	0	42

a)



0/4	1/4	3/4	0/6	1/6	5/6
0.2	33	0.3	0.5	33	0.5
0.3	0.3	15	0.5	0.5	15
0.3	0.3	27	0.6	0.6	21
0.4	0.4	42	0.7	0.6	27
			0.7	0.7	34
			0.7	0.8	42

b)



0/4	1/4	3/4	0/6	1/6	5/6
0.5	33	0.6	0.9	33	1
0.5	0.5	15	1	1	15
1.6	0.6	27	4.1	1	21
1	1	42	2.6	3.2	27
			1.8	1.9	34
			3.4	4	42

c)

Figure 7.6: Impact of variability on the output throughput for a) batch task model; b) playout model; c) streaming model.

of variability-affected cores. For instance, the execution time of the 5/6 configuration has a larger increase than for the 3/4 one. However, with the given error rate distribution this is not enough to make any of the four cores configuration more performing.

Output Rate-Constrained Processing Results. In this case the metric to be considered is the output throughput. In order to consider a worst case condition, we set the consumer frequency corresponding to the maximum throughput that can be delivered by the six core configuration, which is about 9MBytes/sec. As such, introducing idleness has an immediate impact on throughput, as it can be observed in Figure 7.6.b. Differently from the execution time for the previous task model, throughput degradation here is less sensitive to the fraction of variability affected cores. Indeed, in Figure 7.5 the 5/6 scenario has a throughput drop of 23% while the 3/4 scenario has a throughput drop of 22%. However, by comparing 1/4 and 1/6 scenarios, the relative throughput drop is 9.1% compared to 4.4%.

Input-Output Rate-Constrained Streaming Results. Both the input and output throughput are critical in this case. Figure 7.6.c shows variability effects on the input throughput. The same results have been obtained for the output throughput (not shown). Interestingly, also for the input throughput the relative performance drop for high throughput values is similar for 3/4 and 5/6, being around 20% in both cases (see Figure 7.5).

7.5 Summary

In this chapter we presented an adaptive idleness distribution policy aimed at reducing the impact of variations and aging on the lifetime of MPSoCs. The policy exploits variability monitors and on-line task execution statistics to determine the duration of idle intervals to be distributed to the cores to match a given lifetime requirement. The proposed strategy has been implemented on an industrial simulator of a next generation nanoscale multiprocessor platform.

Chapter 8

Using micro thermoelectric cooling in multicore processors *

8.1 Overview

Our contribute so far regarded MPSoCs that are characterized by larger numbers of small cores. As we explained DVFSs can cause penalties in area in these systems. In this chapter we want to give some insights with regard to general-purpose multicore processors which are characterized by few complex cores. In particular we focus on the problem of the reliability keeping attention at the performance preservation.

While it is possible to reduce the operating temperature through the use of dynamic voltage and frequency scaling (DVFS), this reduction comes at the expense of the performance and runtime of applications. Furthermore, the increase in runtime makes the extended lifetime of the processor less useful as applications will take longer to finish.

Micro thermoelectric cooling technology presents an approach that can supplement traditional air-based cooling techniques to reduce the temperatures of processors. Micro thermoelectric coolers are inserted between the processor's die and the processor's heat spreader as illustrated in Figure 8.1. A thermoelectric cooler pumps heat from the die side to the heat spreader side against a temperature gradient. This pumping uses electrical energy, and thus, thermoelectric cooling has to be exercised carefully. Micro thermoelectric coolers (TECs) are particularly attractive to use with multi-core processors, where each core can use its own TEC,

[0]*This research has been done in collaboration with SCALE laboratory at Brown University and it has been partially funded by NSF grant No. 5-26874.

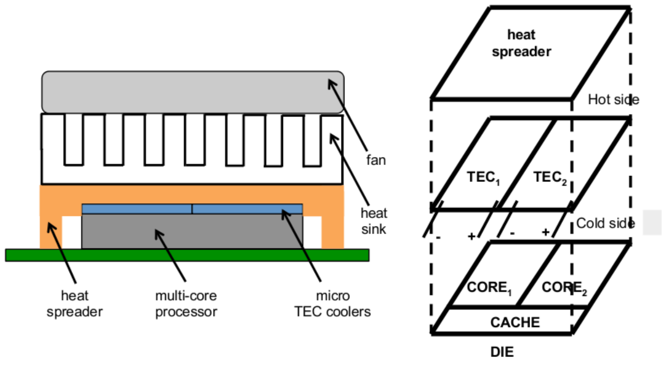


Figure 8.1: Processor heat removal system incorporating micro TECs. Thermal interface material is assumed at the interfaces between surfaces.

and thus cooling efforts can be focused directly on overheating cores.

In this chapter we explore the use of thermoelectric coolers to improve the reliability and performance of multi-core processors. Our contributions are as follows.

- We devise a reliability model for multi-core processors. Our model is driven by real measurements from a dual-core processors. We measure the power, temperature and voltage of each core. We then feed these measurements to a reliability model to estimate the expected mean-time-to-failure (MTTF) for each core and for the entire processor.
- We develop a thermal model for our dual-core processor with the TECs, and use this model to simulate the impact of using the TEC on the temperature of the processor and total power consumption.
- We devise a number of strategies for using TEC and DVFS to improve the reliability and performance of multi-core processors. Our strategies seek to maximize performance while using total power consumption and MTTF as constraints.
- Using measurement traces from a real dual-core processor based system, we quantify the impact of using our techniques on performance, power consumption, and the MTTF. We show that using TEC with DVFS offers a valuable trade-off operating point that improves MTTF and performance compared to pure DFVS.

In Chapter 4 we presented the causes of the lifetime degradation, in particular we described mechanisms of failure and the related models in Section 4.2.2. The rest of the chapter is organized as follows. Section 8.2 reviews related work in the literature. In Section 8.3, we propose a TEC thermal model for multi-core processors. In Section 8.4 we propose a number of strategies to control MTTF within power budgets. A number of comprehensive experiments are provided in Section 8.5 to illustrate the impact of using TEC and DVFS on reliability, performance and power. Section 8.6 summarizes the main conclusions of this chapter.

8.2 Related work

Higher power densities, die temperatures and smaller nanometer features have pushed failure as a major concern in modern processors [16, 70]. As a result, it is now valuable to incorporate reliability modeling and optimization into the design and operationation of processors. A popular architectural-level reliability modeling tool is RAMP [70]. RAMP uses floorplan information with power and thermal traces produced from instruction-level architectural simulators to estimate per-structure and system MTTF for every failure mechanism. RAMP-like tools have been used in a number of architectural-related papers to evaluate system-level design and runtime choices on the reliability of processors. At the design side, Atienza *et al.* propose design optimizations for the register file to improve its reliability [6]. At the system-level runtime side, Lu *et al.* propose DVFS-based techniques to extend the lifetime of the processor [49], and Coskun *et al.* use simultaneously DVFS and job scheduling methods to increase the lifetime [23]. RAMP-like models can be also adapted to evaluate the reliability of real processors from their actual measurements. For example, Mesa-Martinez *et al.* develops reliability models for a single core AMD processor from temperature, power, voltage of the processor [52].

To avoid degradation to performance, it is possible to adjust the cooling system to reduce the operating temperatures. Because most failures mechanisms depend strongly on temperature, small reductions in temperature can lead to large improvements in MTTF. Cooling systems need to be used judiciously due to their power consumption. Micro cooling (whether liquid based [20] or themoelectric based [44, 69]) can directly focus the cooling on the hot spots, which reduces the cooler power consumption. Recent advances in thermoelectric cooling have improved the heat removal capability of TECs, while bringing further miniaturization. It has been recently demonstrated the possibility of using micro TECs to track hot spots and adaptively cool them in a dual-core processor [5]. The design of the TEC itself naturally plays n important rule in its efficiency. Thus, Long *et al.* consider design optimization choices for thin film thermoelectric coolers [46].

Our work differs from previous work in a number of ways. First, we consider the impact of the cooling system directly on the reliability, rather than just the temperature, of the system. We also consider the simultaneous use of TECs and DVFS to optimize reliability and performance of multi-core processors.

8.3 Thermal modeling

Peltier-based TECs pump heat, Q, from the *cold* side of the TEC to the *hot* side of the TEC creating a difference of temperature, ΔT, between the two sides that is dependent on the electrical energy provided to the TEC. Figure 8.1 in Section 8.1 illustrates the embedding of two micro TECs between the processor die and the heat spreader. One TEC has an area equal to the half of the die and in particular will be located on top of one core and half the cache. Without the TECs, the die side is naturally hotter than the heat spreader side, and heat flows from the die to the heat spreader. However, when TECs are used, the cold side is the die side, and the hot side is the heat spreader side, and the TECs pump heat, Q from the processor against the thermal gradient ΔT. The relationship between Q and ΔT is given by

$$Q = ST_cI - K\Delta T - I^2R/2, \tag{8.1}$$

where S is the Seebeck coefficient; K is the TEC thermal conductance; R is the TEC electrical resistance, I is the TEC electrical current and T_c the temperature at the cold side [69]. K and R are constant parameters that depend on the TEC construction.

If we assume a particular desired cool side temperature for the die (e.g., $T_c = 35$celsius), then the relationship between Q and ΔT is linear for a fixed I as given by Equation (8.1). Figure 8.2 gives this relationship for various values of current supply I. The plots illustrate some typical TEC characteristics. For a fixed I, increasing the amount of pumped heat, Q, decreases the temperature difference ΔT. The maximum amount of heat that can be pumped is reached when $\Delta T = 0$. The maximum amount of heat, $Q_{\max}$, that can be pumped at the highest current setting, I_{max}, is one of the most important parameters of a TEC. A mismatch between the power dissipated by the processor and the $Q_{\max}$ of its TECs can lead to thermal runaway. The maximum temperature difference, $\Delta T_{\max}$, obtained at $I_{\max}$ obtained when no heat is pumped is another important parameter.

The voltage of TEC as a function of the applied current, I, and the temperature difference, ΔT, and it is given by $V_{TEC} = S\Delta T + IR$. Thus the power consumption of the TEC, P_{TEC}, is equal to $V_{TEC}I$. This consumed power has to be dissipated at the hot side of the TEC. This extra power consumption is the drawback of using TECs; furthermore, the heat rejected at the hot side of the TEC, which is the sum of Q and P_{TEC}, increases the temperature of the heat spreader compared to the case when no TEC is used. Thus, to model ΔT and the exact temperature of the cold side, it is necessary to develop a thermal model for the TECs with the processor.

Figure 8.3 shows the thermal circuit of the simpler case of a single-core processor connected to the heat sink through one TEC. In the figure, the power of the processor is modeled

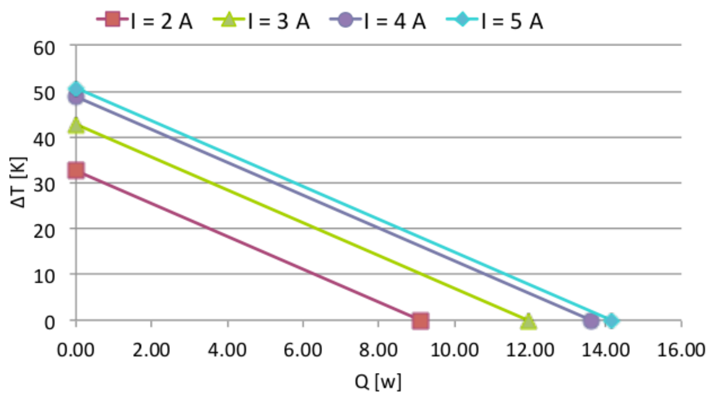


Figure 8.2: Relation between Q and ΔT for various values of current I.

by the current source Q; the power of the TEC is modeled by the current source, $V_{TEC}I$; and ΔT is modeled by a supply source. The lumped thermal resistance of the spreader, sink, and fan assembly is modeled by R_s. The TEC also introduced its own thermal resistance with is determined by it thermal conductance and its physical dimensions. Note that the heat sink assembly has to dissipate the sum of the TEC power and the processor power. The temperature on the hot side, T_h, is equal to

$$\begin{aligned} T_h &= (IV_{TEC} + P_{core})R_s + T_{amb} \\ &= (I(S\Delta T + IR) + P_{core})R_s + T_{amb}, \end{aligned} \tag{8.2}$$

where T_{amb} is the ambient temperature. Thus, the temperature of the cold side is equal to $T_c = T_h - \Delta T$. To develop a lumped thermal model for a dual-core processor, we have

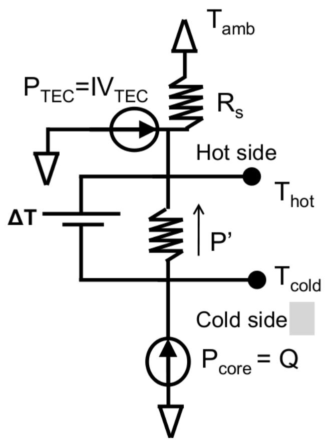


Figure 8.3: Thermal circuit for one TEC used with single-core processor.

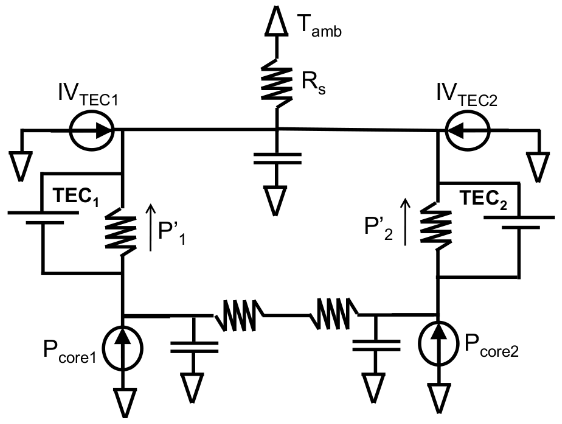


Figure 8.4: Thermal circuit modeling for two TECs attached to a dual-core processor as illustrated in Figure 8.1.

to consider the mutual thermal dependency between the cores as shown in Figure 8.4. The temperatures at the cold sides of the TECs have to be found by solving the linear equations that represent the thermal model.

8.4 Strategies for improving reliability and performance

TECs can be use to keep down core temperatures but since they consume power it is to devise strategies that use them intelligently. In this section we propose a number of strategies to engage TECs to improve the MTTF. As described in Section 4.2.2, MTTF depends on temperature, power, and voltage but the most important parameter is the temperature. Reducing the temperature also has the additional benefit of reducing leakage power. In addition to improving reliability, we also focus on preserving the highest possible performance. In contrast to adaptive DVFS techniques that improve reliability at the expense of performance, we want to identify strategies that meet or improve the Reliability with minimum loss in performance. We also want to take into account the power consumption of the TECs over time and put some constraints on the TEC power consumption if necessary. We propose two strategies.

Strategy I: Maximize MTTF for a Given TEC Power Budget. In this strategy we seek to provide a solution of the problem of maximizing MTTF under TEC power constraints. Power constraints on TECs really means that the TEC power should not be substantial in comparison to the core power. Thus, we consider the TEC power budget as a maximum *ratio* between the TEC power and the core power that we have to meet at any time. Given the power ratio, we search for the minimum T_c that the core can reach at every moment in time. Decreasing

	2.80 GHz at Vdd 0.996 V															
	SYS	CORE1							CORE2							SYS
	GOPS	Power max[W]	Tmax [C]	EM [Y]	SM [Y]	TDDB [Y]	TC [Y]	NBTI [Y]	Power max[W]	Tmax [C]	EM [Y]	SM [Y]	TDDB [Y]	TC [Y]	NBTI [Y]	MTTF [Y]
perlbench - gromacs	7.091	21	51	32	40	25	33	25	19	43	75	84	32	64	34	14
bzip2 - tonto	6.980	23	51	41	43	25	35	26	30	44	55	83	31	63	34	15
gcc - hmmer	7.182	37	51	46	48	26	38	27	35	43	63	92	32	70	35	15
gobmk - h264ref	7.676	18	50	50	42	25	34	26	28	42	50	89	34	69	35	15
hmmer - povray	9.060	20	51	34	36	24	30	24	29	45	38	71	30	54	32	13
sjeng - calculix	9.227	25	51	21	35	23	29	24	21	45	65	67	29	51	31	12
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
AVERAGE	7.125	23	49	42	46	26	37	27	23	44	82	81	31	62	33	14
	2.13 GHz at Vdd 0.884 V															
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
AVERAGE	5.978	15	43	108	92	17041	72	35	14	43	136	108	18326	85	38	24
	1.60 GHz Vdd 0.804 =V															
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
AVERAGE	4.393	9	37	277	175	3,37E+06	159	46	9	37	201	108	2,70E+06	85	38	29

Table 8.1: Summary of characterization results of the SPEC CPU 06 on our dual-core processor. For space limitations we report results on only a few pairs of benchmarks. Averages are, however, computed across all pairs of benchmarks.

the temperature to the minimum possible value maximizes the MTTF, while keeping the same performance of the system since we are not scaling frequency. To find such minimum temperature, we incrementally increase the TEC current and solve Equation (8.1) and Equation (8.2) to compute T_c. We stop when the minimum T_c is found, while keeping all parameters within the TEC specification.

Strategy II: Meeting Required MTTF using Minimum TEC Power and minimum Performance Degradation. In this strategy we seek to ensure a minimum acceptable MTTF while using the least amount of TEC power consumption. To identify the minimum power required to get to the required MTTF, we incrementally increase the power consumption of the TEC and use Equation (8.1) and Equation (8.2) to identify the cold side temperature at every moment of time. The cold side temperature is then fed to the failure models to compute the system MTTF. If the system MTTF reaches the required value, then the algorithm stops. In some extreme conditions, a situation might arise where the core power is higher than the $Q_{\max}$ of the TEC. In this case, we are forced to use DVFS as a means to reduce the core's power. We refer to this strategy by *adaptive TEC*.

8.5 Experimental Results

Our experimental system is equipped with an Intel Core 2 Duo E4700 processor and 4 GB of DRAM. The processor has three DVFS states: 2.8 GHz at 0.996 V, 2.13 GHz at 0.884 V, and 1.6 GHz at 0.804 V. We intercept the power supply lines to the processor and measure the current consumption using an Agilent 34410A multimeter. We measure the temperature of

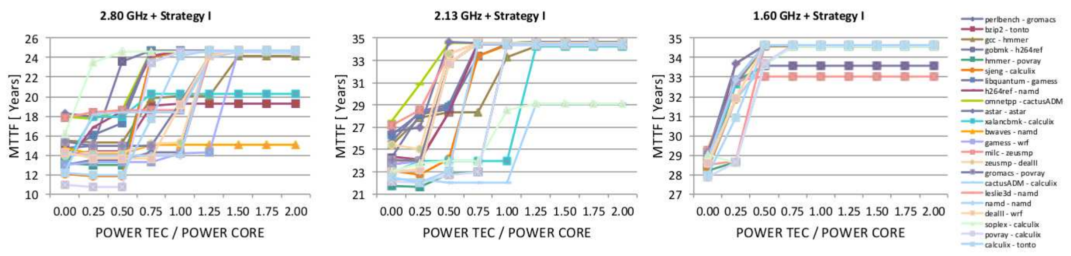


Figure 8.5: MTTF as a function of different TEC power ratios at various DVFS settings.

each core from the embedded sensors using the `lmsensors` package. We also measure the performance counters of the processor using the `pfmon` package. Measuring the performance counters enables us to calculate the throughput of the processor and to estimate the individual power of the cores from the total power consumption of the processor as discussed in Section 4.2.2. We use a sampling interval of 200 ms for all measurements. The measured power, temperature and voltage traces of each core are fed to our RAMP-like model to estimate the FIT/MTTF for the five failure mechanisms and the MTTF of the entire processor as discussed in Section 4.2.2.

Our TEC model is based on TEC-microsystems model number 1MDL06-052-03. We assume two TECs are embedded between the processor die and the heat spreader. The physical dimensions of each core match the physical dimensions of the individual cores of the dual-core processor. The parameters of our TEC model at 300 K are: $\Delta T_{max} = 67$ K, $Q_{max} = 18.7$ W, $I_{\max} = 5.3$ A, $R = 0.87\ \Omega$, and $V_{\max} = 6.3$ V.

In the first experiment we characterize the impact of workload variations on the power, voltage, temperature, throughput, and MTTF of our dual-core processor under different DVFS settings. We use the SPEC CPU 2006 benchmarks, where a pair of benchmarks are executed, with one benchmark per core. We ran every possible combination of the 29 SPEC CPU 2006 benchmarks at every possible frequency-voltage setting for 120 seconds. Table 8.1 gives the average total throughput of the processor in Giga Operations Per Second (GOPS) and the system MTTF in years. We also report in the table the maximum power, temperature, and MTTF of every failure mechanism for each of the two cores. Due to space limitations, we report results for a few pairs of benchmarks. The average values are, however, computed across all pairs of benchmarks. All failure mechanisms are directly affected by temperature; in addition, EM is affected by power and TDDB is affected by voltage. The MTTFs for EM and TDDB are very high at 1.60 GHz and 2.13 GHz. Thus, for these frequencies the system's MTTF is largely determined by NBTI, TC, and SM, which are largely determined by temperature.

In the second experiment we evaluate the improvement in MTTF of the dual-core processor as a function of the power consumption of the TECs at every DVFS setting. We vary the *ratio* of the power consumption of TECs to its core, and for each setting, we identify the largest possible improvements in the core temperatures and the system MTTF using Strategy I proposed in Section 8.4. We plot the results in Figure 5 for a number of application traces at the three different frequency-voltage settings. In Figure 5, the x-axis gives the ratio between the TEC power consumption and the core power consumption, and the y-axis gives the MTTF for the different pairs of benchmarks. Increasing the TEC power consumption reduces the core temperatures and improves the system MTTF. Because many failure mechanisms depend exponentially on the temperature, small reductions in temperature can result in large improvements in MTTF. For example, at 2.8 GHz, engaging the TEC can double the MTTF for sjeng-calculix at the expense of an additional 60% increase in power consumption. At 1.6 GHz, MTTF can improve by 26% at the expense of an additional 35% increase in power consumption. Note that the curves exhibit two "flat" regions at low TEC power ratios and at high TEC power ratios. At low TEC power ratios, the electrical power supplied to the TEC might not be sufficient to pump the power dissipated by the processor, resulting in no thermal or MTTF improvements. At high TEC power ratios, MTTF improvements saturate at a point when the power consumption of a TEC reaches its maximum power rating ($V_{\max}I_{\max}$). At such stage $Q_{\max}$ is being pumped from each core by its TEC. We also observe a trend where smaller TEC power ratios are required to reach saturation at smaller frequencies. This result is expected since cores consume less power at lower frequencies and the TECs have to pump less heat producing larger ΔT.

In the third experiment we evaluate the impact of using TEC and DVFS on the performance, MTTF, and power consumption of the processor. Our goal is to control the MTTF during runtime with little or no impact to performance using Strategy II developed in Section 8.4, which we will call *adaptive TEC*. To mimic real-world settings and generate sufficient MTTF variations, we sequentially execute different pairs of benchmarks. Each pair is executed for 100 billion operations, before the next pair is brought into the system. Table 8.2 gives two different benchmark combinations that will be analyzed in this experiment. Each combination involves six pairs for a total of 600 billion operations. In Figure 6, we plot the MTTF (blue solid line) when the frequency is statically held at the highest setting 2.80 GHz. The dashed gray line gives the MTTF from using adaptive DVFS, while the dotted red line gives the MTTF from using adaptive TEC. Table 8.3 gives the average MTTF for these strategies, where it is clear that both adaptive TEC and DVFS give larger improvements in MTTF (from 16 years to 25 years). The table and figure show that adaptive TEC finishes executing all the operations in less runtime compared to adaptive DVFS for an average improvement of about 17%. However, adaptive TEC uses higher power consumption than adaptive DVFS. Our results demonstrate that none of the evaluated three strategies dominate any of the two others. Each strategy gives

	Combination 1		Combination 2	
GOPS	CPU1	CPU2	CPU1	CPU2
[0 : 100]	bwaves	namd	xalancbmk	calculix
[100 : 200]	perlbench	gromacs	perlbench	gromacs
[200 : 300]	libquantum	gamess	povray	calculix
[300 : 400]	gcc	hmmer	dealII	wrf
[400 : 500]	libquantum	gamess	gromacs	povray
[500 : 600]	perlbench	gromacs	zeusmp	dealII

Table 8.2: Combinations where pairs of benchmarks that are executed in sequence. Each pair of applications is executed for 100 GOP.

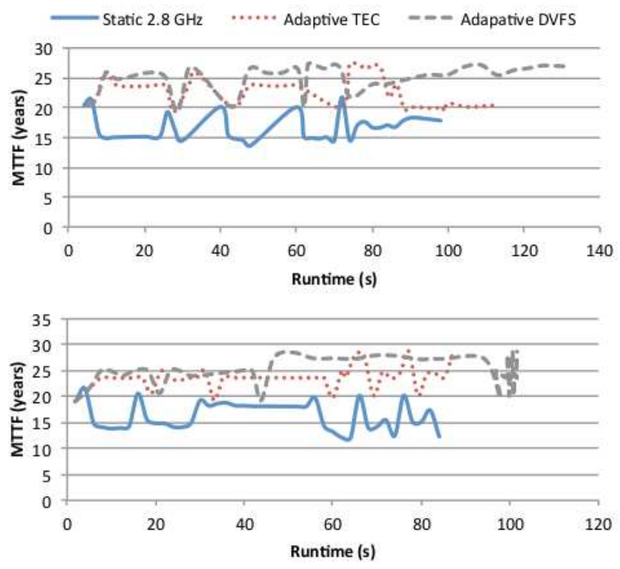


Figure 8.6: MTTF over the time for different benchmark combinations using different strategies.

a valuable trade-off among MTTF, performance, and power consumption. Depending on the computing system objectives, the right strategy should be engaged.

While leakage is included in our initial traces, we do not model the reduction in leakage due to the change in temperature arising from the use of TECs. Thus, the real total power consumption when TECs are used will be less than our conservative estimates.

8.6 Summary

In this chapter we investigated the use of thermoelectric cooling to improve the reliability and performance of multi-core processors. We devised a reliability model to characterize the MTTF as a function of operating temperatures, power and voltages derived from measurements

Combination 1

metric	static 2.80 GHz	adaptive TEC	adaptive DVFS
MTTF (years)	16	23	25
GOPS	6.47	6.16	4.97
Average power (W)	30	30	22
runtime (s)	99	104	129

Combination 2

metric	static 2.80 GHz	adaptive TEC	adaptive DVFS
MTTF (years)	16	24	25
GOPS	6.11	5.33	4.49
Average power (W)	30	20	19
runtime (s)	83	95	113

Table 8.3: Summary of results.

on a real dual-core processor. We also developed a thermal model to evaluate the impact of using TECs on the die temperature and power consumption. We then proposed a number of strategies to use adaptive TEC and DVFS to improve the reliability and performance with minimum power increases. In our experiments, we explored comprehensively the trade-off among reliability, power, and performance under a number of strategies such as static frequency assignments, adaptive DVFS, and adaptive TEC. We demonstrated that TECs offer a valuable operational point that delivers improved reliability without the performance degradation of pure DVFS techniques and with a reasonable increased power budget.

Chapter 9

Conclusions

Realizing multicore platforms in a single chip is becoming an unavoidable choice to obtain a comparable increase between power and performance in recent CMOS technologies. The miniaturization of the components produces undesired post-fabrication variations on the technological parameters; the cores of the platforms can differ in terms of power and speed from the nominal values. Moreover several mechanisms depending on temperature, supply voltage, and stress of the components, create speed degradation over time that can also generate soft and hard errors if not well controlled.

Multicore platforms are used for large application domains to meet tight requirements in terms of energy saving, performance, and lifetime. Hardware techniques at design time are not sufficient to reach all these targets, and then adaptive software strategies are needed. In particular the aim of this thesis was to devise runtime mechanisms able to manage the actual degradation status among the cores known by using monitors placed on the chip and meet the given requirements for the current workload. Many solutions were proposed in literature; in particular we wanted to improve the solutions aimed at minimizing the energy consumption while meeting a time constraint in multimedia multicore platforms. We firstly discovered a technique to find the optimal solution by formulating the problem through an Integer Linear Problem Formulation (ILP), then, since the algorithm of such method is time-demanding and cannot be applied on-line, we studied a sub-optimal solution based on two steps, namely a Linear Programming (LP) and a Bin Packing (BP). We proved that since the LP part meets some key properties its solution can be calculated in closed-form. We devised a simple algorithm characterized by a linear cost with respect to the number of the cores that can be applied on-line and which solves the overall problem LP+BP. We demonstrated its efficiency by comparing it against ILP, state-of-the-art policies, and variability-agnostic strategies by running real multimedia applications on the virtual prototype of an industrial next-generation multicore platform. LP+BP can save up to 33% energy with respect to state-of-the-art policies and 65% energy with respect to variability-agnostic task allocation policies while providing better QoS.

Furthermore we faced the problem of meeting a given lifetime requirement in multicore multimedia platforms. We presented an adaptive idleness distribution policy aimed at reducing the impact of variations and aging on the lifetime. The policy exploits variability monitors and on-line task execution statistics to determine the duration of idle intervals to be distributed to the cores to match a given lifetime requirement. We evaluated the impact on performance for different degradation scenarios.

Finally we investigated the use of micro thermoelectrical coolers to control the temperature of the cores at runtime with the objective of meeting lifetime constraints without performance loss. We showed that using only DVFS-based techniques the recovered lifetime implies performance degradations.

An adaptive technique to control performance, power, and lifetime all together needs further research. However, the techniques we proposed in this thesis - if correctly handled - can be used together.

In fact, not all the applications are performance-hungry. This may be the case when the user wants to watch a movie or manage some pictures, while for the rest of the time the applications that are running are not particularly demanding. This suggests two observations.

The first one is that we have to activate task allocation techniques to deliver high performance only when the platform is under stress, while in the other cases the runtime can manage the idleness to meet lifetime constraints.

The other observation is that the strategy for lifetime preservation can exploit the intrinsic idle time that the cores experience during runtime. If the policy is based on statistical information about the scheduling of the applications related to the typical usage of the user, it can enforce the lifetime preservation when the computations do not have stringent time constraints. This allows to push the hardware to its peak performance when necessary. Also thermoelectrical coolers can be adopted only for the strictly necessary time frame; in this manner the power they require will be well-amortized.

Appendix A - Proof of Proposition 1

Here we prove Proposition 1.

Proof. Consider LP 6.7 and its LP dual:

$$\max_{\alpha,\beta,\gamma} K\alpha - T\gamma$$
$$\begin{cases} f_{cki}\alpha - \beta_i \leq p_i & \forall i : 1 \ldots N \\ \sum_{i=1}^{N} \beta_i - \gamma \leq q & \\ \beta_i, \gamma \geq 0 & \forall i : 1 \ldots N \end{cases} \tag{9.1}$$

where α is the dual variable associated with constraint $\sum_{i=1}^{N} f_{cki}x_i = K$, β_i the dual variable associated with constraint $t \geq x_i$, and γ the dual variable associated with constraint $T \geq t$. By the weak LP duality theorem, given a feasible solution x^*, t^* of (12) and a feasible solution $\alpha^*, \beta^*, \gamma^*$ of (9.1) having the same value, i.e. such that $\sum_{i=1}^{N} p_i x_i^* + qt^* = K\alpha^* - T\gamma^*$, both solutions are optimal. Accordingly, the proof is based on showing the optimal dual solution associated with (a) and (b) in the statement.

First, suppose $\sum_{i=1}^{s} f_{cki}T \geq K$, in which case (a) is immediately checked to be feasible for (12) (in particular, $t^* \leq T$). Consider the following solution of (9.1): $\alpha^* = \frac{\sum_{i=1}^{s} p_i + q}{\sum_{i=1}^{s} f_{cki}}$; $\beta_i^* = f_{cki}\alpha^* - p_i$ for $i = 1, \ldots, s$; $\beta_i^* = 0$ for $i = s+1, \ldots, N$; $\gamma^* = 0$. Elementary calculations show that this solution is feasible and has the same value as x^*, t^*.

Second, suppose $\sum_{i=1}^{s} f_{cki}T < K$, and consider the feasible solution (b) to (12). Consider the following solution of (9.1): $\alpha^* = \frac{p_r}{f_{ckr}}$; $\beta_i^* = \frac{p_r}{f_{ckr}} \cdot f_{cki} - p_i$ for $i = 1, \ldots, r-1$; $\beta_i^* = 0$ for $i = r, \ldots, N$; $\gamma^* = \frac{p_r}{f_{ckr}} \cdot \sum_{i=1}^{r-1} f_{cki} - q - \sum_{i=1}^{r-1} p_i$. Also in this case, elementary calculations show that this solution is feasible, in particular that $\gamma^* \geq 0$, and has the same value as x^*, t^*. $\square$

9. APPENDIX A - PROOF OF PROPOSITION 1

Appendix B - Published Papers

Several publications on international journals have been obtained during the development of this thesis, and several works have been presented at international conferences such as Design, Automation and Test in Europe (DATE 2009 and 2011), Embedded Systems for Real-Time Multimedia (ESTIMEDIA 2009), Computing Frontiers (CF 2010), and System on Chip (SoC 2011).

We list below these publications by dividing the papers published on international journals and the papers published on the proceedings of international conferences.

Journal Papers:

1. **F Paterna**, A Acquaviva, A Caprara, F Papariello, G Desoli, L Benini. "Variability-aware Task Allocation for Energy-Efficient Quality of Service Provisioning in Embedded Streaming Multimedia Applications". It will appear on *Computers, Transactions on.* IEEE. The preprint version is available online at `http://ieeexplore.ieee.org/`.

2. **F Paterna**, A Acquaviva, F Papariello, G Desoli, L Benini. "Variability-tolerant workload allocation for mpsoc energy minimization under real-time constraints". It will appear on *Embedded Computing Systems, Transactions on.* ACM.

Conference Papers:

3. D Bortolotti, **F Paterna**, C Pinto, A Marongiu, M Ruggiero, L Benini. "Exploring Instruction Caching Strategies for Tightly-coupled Shared-memory Clusters". *System on Chip, Proceedings of the Conference on*, 34-31. IEEE, 2011.

4. **F Paterna**, A Acquaviva, A Caprara, F Papariello, G Desoli, L Benini. "An Efficient On-line Task Allocation Algorithm for QoS and Energy Efficiency in Multicore Multimedia

Platforms". *Design, Automation and Test in Europe, Proceedings of the Conference on*, 1-6. IEEE, 2011.

5. **F Paterna**, A Acquaviva, A Caprara, F Papariello, G Desoli, L Benini. "Variability-tolerant Run-time Workload Allocation for MPSoC Energy Minimization under Real-time Constraints". *Computing Frontiers, Proceedings of the Conference on*, 109-110. ACM, 2010.

6. **F Paterna**, A Acquaviva, F Papariello, G Desoli, L Benini. "Variability-tolerant Workload Allocation for MPSoC Energy Minimization under Real-time Constraints". *Embedded Systems for Real-Time Multimedia, Proceedings of the Workshop on*, 134-142. IEEE/ACM, 2009.

7. **F Paterna**, A Acquaviva, F Papariello, G Desoli, M Olivieri, L Benini. "Adaptive Idleness Distribution for Non-uniform Aging Tolerance in Multiprocessor Systems-on-chip". *Design, Automation and Test in Europe, Proceedings of the Conference on*, 906-909. IEEE, 2009.

References

[1] ACCELLERA SYSTEMS INITIATIVE *The open systemc initiative website.* http://www.systemc.org. 11

[2] Tutorial 2: Leakage issues in ic design: Trends, estimation, and avoidance. In *Proceedings of the Conference on Computer-aided design*, pages 1–11. IEEE, 2003. 24

[3] M. AGARWAL, B. PAUL, M. ZHANG, AND S. MITRA. Circuit failure prediction and its application to transistor aging. In *Proceedings of the Symposium on the VLSI Test*, pages 277–286. IEEE, 2007. 4

[4] M. ALAM AND S. MAHAPATRA. A comprehensive model of pmos nbti degradation. *Microelectronics Reliability*, **45**(1):71–81, ELSEVIER, 2005. 25

[5] R. ALLEY, M. SOTO, L. KWARK, P. CROCCO, AND D. KOESTER. Modeling and validation of on-die cooling of dual-core cpu using embedded thermoelectric devices. In *Proceedings of the Symposium on Semiconductor Thermal Measurement and Management*, pages 77–82. IEEE, 2008. 85

[6] D. ATIENZA, G. DE MICHELI, L. BENINI, J. AYALA, P. DEL VALLE, M. DEBOLE AND V. NARAYANAN Reliability-aware design for nanometer-scale devices. In *Proceedings of the Conference on Asia and South Pacific Design Automation*, pages 549–554. IEEE, 2008. 85

[7] K. BERNSTEIN, D. FRANK, A. GATTIKER, W. HAENSCH, B. JI, S. NASSIF, E. NOWAK, D. PEARSON, AND N. ROHRER. High-performance cmos variability in the 65-nm regime and beyond. *Journal of Research and Development*, **50**:433–449, IBM, 2006. 23, 24

[8] B. BILGIC, B. HORN, AND I. MASAKI. Efficient integral image computation on the gpu. In *Proceedings of the Symposium on Intelligent Vehicles*, pages 528–533. IEEE, 2010. 19

[9] G. BLAKE, R. DRESLINSKI, AND T. MUDGE. A survey of multicore processors. *Signal Processing Magazine*, **26**(6):26–37, IEEE, 2009. 9, 10

[10] J. BLOME, S. FENG, S. GUPTA, AND S. MAHLKE. Online timing analysis for wearout detection. In *Workshop on Architectural Reliability held in conjunction with International Symposium on Microarchitecture*, 2006. 73

[11] S. BORKAR. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *Micro*, **25**(6):10 – 16, IEEE, nov.-dec. 2005. 4

[12] S. BORKAR. Thousand core chips: a technology perspective. In *Proceedings of the Conference on Design Automation*, pages 746–749. ACM, 2007. 2

[13] S. BORKAR AND A. CHIEN. The future of microprocessors. *Communication*, **54**:67–77, ACM, May 2011. 3

[14] S. BORKAR, T. KARNIK, S. NARENDRA, J. TSCHANZ, A. KESHAVARZI, AND V. DE. Parameter variations and impact on circuits and microarchitecture. In *Proceedings of the Conference on Design Automation*, pages 338–342. ACM, 2003. 24

[15] K. Bowman, A. Alameldeen, S. Srinivasan, and C. Wilkerson. Impact of die-to-die and within-die parameter variations on the throughput distribution of multi-core processors. In *Proceedings of the Conference on International Symposium on Low Power Electronics and Design*, pages 50–55. ACM, 2007. 23

[16] D. Brooks, R. Dick, R. Joseph, and L. Shang. Power, Thermal, and Reliability Modeling in Nanometer-Scale Microprocessors. *Micro*, **27**(3):49 – 62, IEEE, 2007. 27, 85

[17] J. Butts and G. Sohi. A static power model for architects. In *Proceedings of the Symposium on Microarchitecture*, pages 191–201. ACM, 2000. 3

[18] R. Buyya. *High Performance Cluster Computing: Programming and applications*. High Performance Cluster Computing. Prentice Hall PTR, 1999. 16, 19

[19] Y. Cao and C. McAndrew. Mosfet modeling for 45nm and beyond. In *Proceedings of the Conference on International Conference on Computer-Aided Design*, pages 638–643. IEEE, 2007. 24

[20] K. Chakrabarty, P. Paik, and V. Pamula. *Adaptive Cooling of Integrated Circuits Using Digital Microfluidics*. Artech House Publishers, first edition, 2007. 85

[21] NVIDIA Corp. Nvidia cuda: Compute unified device architecture. 2008. 8

[22] Tilera Corp. Tilepro64 processor. 2008. 10

[23] A. Coskun, R. Strong, D. Tullsen, and T. Rosing. Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors. In *Proceedings of the Conference on SIGMETRICS/Performance*, pages 169–180. ACM, 2009. 85

[24] S. Das, S. Pant, D. Roberts, S. Lee, D. Blaauw, T. Austin, T. Mudge, and K. Flautner. A self-tuning dvs processor using delay-error detection and correction. In *Proceedings of the Symposium on VLSI Circuits*, pages 258–261, Digest of Technical Papers, 2005 73

[25] V. De and S. Borkar. Technology and design challenges for low power and high performance microprocessors. In *Proceedings of the Symposium on Low Power Electronics and Design*, pages 163–168, IEEE, 1999. 3

[26] A. Drake, R. Senger, H. Singh, G. Carpenter, and N. James. Dynamic measurement of critical-path timing. In *Proceedings of the Conference on Integrated Circuit Design and Technology and Tutorial*, pages 249–252. IEEE, 2008. 35

[27] R. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge. Near-threshold computing: Reclaiming moores law through energy efficient integrated circuits. In *Proceedings of the IEEE*, **98**(2):253–266, IEEE, 2010. 3

[28] P. Faraboschi, G. Brown, J. Fisher, G. Desoli, and F. Homewood. Lx: a technology platform for customizable vliw embedded processing. In *Proceedings of the Conference on Computer Architecture*, pages 203–213, IEEE, 2000. 12, 20

[29] E. Flamand. Strategic directions toward multicore application specific computing. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1266–1266. IEEE, 2009. 23

[30] G. Fox. Parallel computing comes of age: Supercomputer level parallel computations at caltech. *Concurrency - Practice and Experience*, pages 63–103, John Wiley & Sons Ltd, 1989 17

[31] D. Geer. Chip makers turn to multicore processors. *Computer*, **38**(5):11–13, IEEE, may 2005. 2

[32] R. Gonzalez, B. Gordon, and M. Horowitz. Supply and threshold voltage scaling for low power cmos. *Journal of solid-State Circuits*, **32**:1210–1216, IEEE 1997. 3

[33] P. Hansen. Model programs for computational science: A programming methodology for multicomputers. *Concurrency - Practice and Experience*, pages 407–423, John Wiley & Sons Ltd, 1993 17

[34] B. Haskell, A. Puri, and A. Netravali. *Digital Video: An introduction to MPEG-2*. Chapman & Hall Ltd., 1996. 20

[35] K. Hazelwood and D. Brooks. Eliminating voltage emergencies via microarchitectural voltage control feedback and dynamic optimization. *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 326–331, IEEE, 2004. 35

[36] S. Herbert and D. Marculescu. Characterizing chip-multiprocessor variability-tolerance. In *ACM, Proceedings of the Conference on Design Automation Conference*, pages 313–318, ACM, 2008. 24

[37] M. Hill and M. Marty. Amdahl's law in the multicore era. *Computer*, **41**:33–38, IEEE, July 2008. 3

[38] S. Hong, S. Narayanan, and M. Kandemir. Process variation aware thread mapping for chip multiprocessors. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 821–826, IEEE, 2009. 36

[39] L. Huang, F. Yuan, and Q. Xu. Lifetime reliability-aware task allocation and scheduling for mpsoc platforms. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 51–56, IEEE, 2009. 36

[40] E. Humenay, D. Tarjan, and K. Skadron. Impact of process variations on multicore performance symmetry. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1653–1658, EDA Consortium, 2007. 23

[41] ILOG. Ilog solver. 2009. 6, 39

[42] Advanced Micro Devices Inc. 9

[43] R. Joseph, D. Brooks, and M. Martonosi. Control techniques to eliminate voltage emergencies in high performance processors. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, pages 79–91, IEEE, 2003 35

[44] D. Koester, R. Venkatasubramanian, B. Conner, and G. Snyder. Embedded thermoelectric coolers for semiconductor hot spot cooling. In *International Conference on Thermal and Thermomechanical Phenomena in Electronics Systems*, pages 491–496, IEEE, 2006. 85

[45] T. LeBlanc and E. Markatos. Shared memory vs. message passing in shared-memory multiprocessors. In *Proceedings Symposium on Parallel and Distributed Processing*, pages 254–263. IEEE, 1992. 16, 17

[46] J. Long, S. Memik, and M. Grayson. Optimization of an On-Chip Active Cooling System Based on Thin-Film Thermoelectric Coolers. In *Design, Automation and Test in Europe*, pages 117–122, IEEE, 2010. 85

[47] ARM Ltd. The arm cortex-a9 processors. 2007. 8, 10

[48] Plurality ltd. Hypercore processor architecture. 2010. 9

[49] Z. Lu, J. Lach, M. Stan, and K. Skadron. Improved Thermal Management with Reliability Banking. In *Micro*, **25**(6):40–49, IEEE, 2005. 85

[50] B. Flachs, M. Hopkins, Y. Watanabe, M. Gschwind, H. Hofstee and T. Yamazaki. Synergistic processing in cells multicore architecture. *Micro*, **26**(2):10–24, IEEE,2006. 9, 10

[51] K. Meng, F. Huebbers, R. Joseph, and Y. Ismail. Modeling and characterizing power variability in multicore architectures. In *Proceedings of the Symposium on Performance Analysis of Systems Software*, pages 146 –157, IEEE, 2007. 3

[52] F. Mesa-Martinez, E. Ardestani, and J. Renau. Characterizing Processor Thermal Behavior. In *Architectural Support for Programming Languages and Operating System*, pages 193–204, ACM, 2010. 27, 85

[53] ST MICROELECTRONICS AND CEA. Platform 2012: A many-core programmable accelerator for ultra-efficient embedded computing in nanometer technology. 2010. 9

[54] J. MITCHELL, W. PENNEBAKER, C. FOGG, AND D. LEGALL, *MPEG Video Compression Standard*. Chapman & Hall Ltd., 1996. 20

[55] G. MOORE. Cramming more components onto integrated circuits. *Electronics*, **38**(8), April 1965. 2

[56] S. NASSIF. Modeling and analysis of manufacturing variations. In *Proceeding of the Conference on Custom Integrated Circuits*, pages 223–228, IEEE, 2001. 3

[57] P. NDAI, S. BHUNIA, A. AGARWAL, AND K. ROY. Within-die variation-aware scheduling in superscalar processors for improved throughput. *Transactions on Computers*, **57**(7):940–951, IEEE, 2008. 24

[58] A. PAPANICOLAOU, M. MIRANDA, P. MARCHAL, B. DIERICKX, AND F. CATTHOOR. At tape-out: Can system yield in terms of timing/energy specifications be predicted? *Proceedings of the Conference on Custom Integrated Circuits Conference*, pages 773–778, IEEE, 2007. 29

[59] K. POPOVICI1 AND A. JERRAYA. Virtual platforms in system-on-chip design., The MathWorks Inc. 12

[60] R. RAO, A. SRIVASTAVA, D. BLAAUW, AND D. SYLVESTER. Statistical analysis of subthreshold leakage current for vlsi circuits. *Transactions on Very Large Scale Integration Systems*, **12**(2):131 –139, IEEE, feb. 2004. 3

[61] B. REBAUD, M. BELLEVILLE, E. BEIGNE, M. ROBERT, P. MAURINE, AND N. AZEMARD. An innovative timing slack monitor for variation tolerant circuits. In *Proceedings of the Conference on IC Design and Technology*, pages 215–218, IEEE, 2009. 35

[62] V. REDDI, M. GUPTA, G. HOLLOWAY, G. WEI, M. SMITH, AND D. BROOKS. Voltage emergency prediction: Using signatures to reduce operating margins. In *IProceedings of the nternational Symposium on High Performance Computer Architecture*, pages 18–21. IEEE, 2009. 35

[63] D. ROBERTS, R. DRESLINSKI, E. KARL, T. MUDGE, D. SYLVESTER, AND D. BLAAUW. When homogeneous becomes heterogeneous. In *Workshop on Operating Systems for Heterogeneous Multiprocessor Architectures*, 2007. 24, 73, 75

[64] S. SAHA. Modeling process variability in scaled cmos technology. *Design Test of Computers*, **PP**(99):1, IEEE, 2010. 3

[65] T. SATO AND T. FUNAKI. Dependability, power, and performance trade-off on a multicore processor. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 714–719. IEEE, 2008. 2

[66] G. SEMERARO, G. MAGKLIS, R. BALASUBRAMONIAN, D. ALBONESI, S. DWARKADAS, H. DWARKADAS, AND M. SCOTT. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, pages 29–40. IEEE, 2002. 3

[67] T. SIDDIQUA AND S. GURUMURTHI. A multi-level approach to reduce the impact of nbti on processor functional units. In *Proceedings of the Symposium on Great lakes symposium on VLSI*, pages 67–72. ACM, 2010. 4

[68] J. SMOLENS, B. GOLD, J. HOE, B. FALSAFI, AND K. MAI. Detecting emerging wearout faults. In *Proceedings of the Workshop on Silicon Errors in Logic*. IEEE, 2007. 73

[69] G. SNYDER, M. SOTO, R. ALLEY, D. KOESTER, AND B. CONNER. Hot spot cooling using embedded thermoelectric coolers. In *Proceedings of the Symposium on Semiconductor Thermal Measurement and Management*, pages 135–143. IEEE, 2006. 85, 86

[70] J. SRINIVASAN, S. ADVE, P. BOSE, AND J. RIVERS. Lifetime Reliability: Toward an Architectural Solution. *Micro*, **25**(3):70–80, IEEE, 2005. 4, 24, 27, 85

[71] J. Srinivasan, S. Adve, P. Bose, and J. Rivers. The case for lifetime reliability-aware microprocessors. In *Proceedings of the International Symposium on Computer Architecture*, pages 276–287, ACM, 2004. 4

[72] J. Srinivasan, S. Adve, P. Bose, and J. Rivers. Lifetime reliability: Toward an architectural solution. *Micro*, **25**(3):70–80, IEEE, 2005. 2, 28

[73] D. Sylvester, K. Agarwal, and S. Shah. Invited paper: Variability in nanometer cmos: Impact, analysis, and minimization. *INTEGRATION, the VLSI journal*, **41**:319–339, ELSEVIER, May 2008. 23

[74] D. Sylvester, D. Blaauw, and E. Karl. Elastic: An adaptive self-healing architecture for unpredictable silicon. *Design and Test of Computers*, **23**:484–490, IEEE, 2006. 23

[75] R. Teodorescu and J. Torrellas. Variation-aware application scheduling and power management for chip multiprocessors. *SIGARCH Computer Architecture News*, **36**(3):363–374, ACM, 2008. 35, 36, 37, 44, 54, 55, 56

[76] Inc. Texas Instruments. Tms320dm6467: Digital media system-on-chip, 2008. 10

[77] A. Tiwari and J. Torrellas. Facelift: Hiding and slowing down aging in multicores. *Proceedings of the Symposium on Microarchitecture*, pages 129–140, IEEE/ACM, 2008. 2, 23, 24, 25

[78] IMEC *VAM variability aware modeling* http://www.imec.be/ScientificReport/SR2007/html/1384291.html 29

[79] ARM Ltd. *Latest ARM OVP Fast Processor Models and Platforms Available.* http://www.ovpworld.org/download_ARM.php 12

[80] Tensilica *The XTensa Modeling Protocol and XTensa SystemC Modeling for Fast System Modeling and Simulation.* http://www.tensilica.com/products/hw-sw-dev-tools/for-software-developers/system-modeling-3/ 12

[81] F. Wang, C. Nicopoulos, X. Wu, Y. Xie, and N. Vijaykrishnan. Variation-aware task allocation and scheduling for mpsoc. In *Proceedings of the Conference on Computer-Aided Design*, pages 598–603, IEEE, 2007. 36, 74

[82] H. Wang, W. Wang, D. Peng, and H. Sharif. A route-oriented sleep approach in wireless sensor networks. In *Proceedings of the Conference on Communication Systems*, pages 1–5, IEEE, 2007. 73

[83] J. Winter and D. Albonesi. Scheduling algorithms for unpredictably heterogeneous cmp architectures. In *Proceedings of the Conference on Dependable Systems and Networks*, pages 42–51, IEEE, 2008. 36, 73

[84] T. Yamauchi, L. Hammond, and K. Olukotun. The hierarchical multi-bank dram: A high-performance architecture for memory integrated with processors. In *Proceedings of the Conference on Advanced Research in VLSI* , pages 303–320. IEEE, 1997. 9

[85] Y. Yi, W. Han, X. Zhao, A. Erdogan, and T. Arslan. An ilp formulation for task mapping and scheduling on multi-core architectures. In *Proceedings of the Conference on Design, Automation, and Test in Europe*, pages 33–38, IEEE, 2009. 36

[86] L. Zhang, L. Bai, R. Dick, L. Shang, and R. Joseph. Process variation characterization of chip-level multiprocessors. In *Proceedings of the Conference on Design Automation Conference*, pages 694–697, ACM/IEEE, 2009. 36, 37