Chapter 8
Design Space Exploration of Parallel Architectures

Carlos Kavka, Luka Onesti, Enrico Rigoni, Alessandro Turco, Sara Bocchio, Fabrizio Castro, Gianluca Palermo, Cristina Silvano, Vittorio Zaccaria, Giovanni Mariani, Fan Dongrui, Zhang Hao, and Tang Shibin

Abstract This chapter will present two significant applications of the MULTICUBE design space exploration framework. The first part will present the design space exploration of a low power processor developed by STMicroelectronics by using the modeFRONTIER tool to demonstrate the DSE benefits not only in terms of objective quality, but also in terms of impact on the design process within the corporate environment. The second part will describe the application of RSM models developed within MULTICUBE to a tiled, multiple-instruction, many-core architecture developed by ICT China. Overall, the results have showed that different models can present a trade-off of accuracy versus computational effort. In fact, throughout the evaluation, we observed that high accuracy models require high computational time (for both model construction time and prediction time); vice-versa low model construction and prediction time has led to low accuracy.

8.1 Introduction

The embedded systems industry faces today an always increasing demand to handle complexity in the design process, which together with the usual strong time-to-market pressure, make essential the use of an automatic tool-based design flow. The use of an automatic tool for Design Space Exploration (DSE) directly impacts on the entire design process within the corporate, with benefits that can be measurable (or tangible), like the reduction of the overall design process lead time, and also qualitative (or intangible) like the streamlining and the reduction of human error prone repetitive operations.

DSE tools for the automation of the embedded system design process do exist today. The MULTICUBE FP7 European Research project has contributed to the

C. Kavka (✉)
ESTECO, Trieste, Italy
e-mail: carlos.kavka@esteco.com

C. Silvano (eds.), *Multi-objective Design Space Exploration of Multiprocessor SoC Architectures*,
DOI 10.1007/978-1-4419-8837-9_8,

development of an Open Source tool and to the re-targeting of a proprietary state-of-the-art multi-disciplinary optimization tool that can be directly applied today to embedded systems design. One of these tools is M3Explorer [6], an open source tool developed from scratch in the MULTICUBE project, and the other is modeFRONTIER™ [2], an already existing proprietary tool widely used in multidisciplinary optimization, which has been re-targeted to the domain of embedded systems.

This chapter presents two case studies of design space exploration in embedded systems to illustrate the benefits of the use of automatic tools from an industrial perspective. Section 8.2 describes a design space exploration study performed on a low power processor design and Sect. 8.3 describes the application of response surface models on a many-core architecture design.

8.2 Design Case Study: Design Space Exploration of the STM Industrial SP2 Platform

This section describes the application of automatic design space exploration for the design of a low-power processor developed by STMicroelectronics, using the modeFRONTIER multidisciplinary optimization tool. The objective is to demonstrate the benefits of the introduction of an automatic design process not only by considering the final objective quality, but also its effects on the entire design process within the corporate. The experiment is an extension of the benchmark used in Chap. 3 to analyze the behavior of the optimization algorithms proposed in the MULTICUBE project [7].

8.2.1 Architectural Model Description

The SP2 processor from STMicroelectronics is a low power and high performance microprocessor offering comparable performance to entry-level desktop microprocessors. It is designed for both generic and mobile applications that need low power dissipation and high peak performance. The presence of a SIMD coprocessor makes it suitable for multimedia applications.

The main architectural features of the SP2 processor can be summarized as follows:

- MIPSISA32 release2 compatible instruction set architecture (ISA) and privilege resource architecture (PRA)
- 4-issue superscalar out-of-order execution
- Built-in 2 integer ALUs, 2 interleaved load-store units
- Split primary instruction and data cache
- 64-byte cache line, 4-way set associative primary caches
- 64-byte cache line, 8-way set associative unified exclusive secondary cache

- Configurable out-of-order parameters: renaming register number, instruction issue window width, reorder buffer depth, etc
- Configurable cache parameters: cache size
- Runtime resizable secondary cache

The experiments presented in this section were performed using Sp2sim, a register transfer level cycle accurate simulator, which runs its applications according to SP2's pipelines and configurations. Sp2sim models precisely the SP2 microprocessor design, which includes both the MIPSISA32r2 ISA and PRA [5]. It also embeds a model of external memory controller, which can be configured to measure the performance in different memory latencies and sizes. The interface between the processor model and the memory controller model is a 64-bit AXI interface. Sp2sim implements both an area and power consumption estimation algorithms. Sp2sim is written in C++ language so as to achieve high simulation speed. It runs on Linux x86 or x86_64 platforms.

8.2.2 Design Space and Application

The SP2 implementation provides eleven configurable parameters, which are classified into three categories: out-of-order execution engine, cache system and branch prediction. By adjusting these parameters, the user can get different processor implementations targeting to distinct application areas. The list of parameters together with a description and their possible values is presented in Table 8.1.

The SP2 simulator produces seven system metrics, which are grouped in three categories: performance, power dissipation and area occupation. Table 8.2 describe the system metrics.

The selected application for the experiments presented in this section is *gzip*, a popular data compression program written by Jean-Loup Gailly, which comes from SPEC CPU2000 benchmark suite [8]. It has been selected since its size is relatively small, and the program behavior is more regular than most other applications.

Table 8.1 The eleven configuration parameters

Category	Parameter	Description	Values
Out of order execution	rob_depth	Reorder buffer depth	32, 48, 64, 80, 96, 112, 128
	rmreg_cnt	Rename register number	16, 32, 48, 64
	iw_width	Instruction window width	4, 8, 16, 24, 32
Cache system	icache_size	Instruction cache size	16, 32, 64
	dcache_size	Data cache size	16, 32, 64
	scache_size	Secondary cache size	0, 256, 512, 1024
	lq_size	Load queue size	16, 24, 32
	sq_size	Store queue size	16, 24, 32
	msh_size	Miss holding register size	4, 8
Branch prediction	bht_size	Branch history table size	512, 1024, 2048, 4096
	btb_size	Branch target buffer size	16, 32, 64, 128

Table 8.2 The system metrics generated by the simulator

Category	Metric	Description
Performance	total_cycle	Total cycle number
	total_inst	Total instruction number
	ipc	Instruction per cycle
Power dissipation	total_energy	Total energy consumed
	power_dissipation	Average power dissipation
	peak_power_dissipation	Peak power dissipation
Area	area	Area occupied

8.2.3 Design Space Exploration

The aim of the experiment presented in this section is to determine the set of designs that minimize the total number of cycles, the power dissipation and the area occupation required to run the selected *gzip* application with the SP2 simulator. Since it is a multi-objective problem, and since the three objectives can potentially be contradictory, the result of the optimization process will not be a single design, but a Pareto front, which corresponds to the set of designs that represents a trade-off between the different objectives.

In real industrial applications, one of the hardest constraints that limits the exploration is determined by the computing resources available for simulation. In this study, the complete design space consists of 1,161,216 designs by considering all combinations of the configuration parameters. This space is clearly too large to be explored exhaustively, making essential to perform a well defined exploration strategy.

In this experiment, the time available for simulation limits the exploration to the execution of at most 8,134 evaluations of the simulator. This information will guide the selection of the algorithms for the different exploration phases. The rest of this section illustrates the design space exploration process, starting with the creation of the optimization workflow, the definition of the experiments, the optimization process and the results assessment.

8.2.3.1 The Optimization Workflow

The optimization tool used in this experiments is modeFRONTIER, a multidisciplinary multi-objective optimization and design tool which is used world-wide in many application fields like aerospace, appliances, pharmaceutics, civil engineering, manufacturing, marine multibody design, crash, structural, vibro-acoustics and turbo-machinery. In the FP7 MULTICUBE project, modeFRONTIER has been enhanced to support categorical discrete domain optimization problems, like typical problems faced in the SoC domain.

The design space exploration process starts with the definition of the optimization workflow, which is presented in Fig. 8.1. The workflow can be graphically defined or can be built automatically using the XML design space definition file (see Chap. 1).

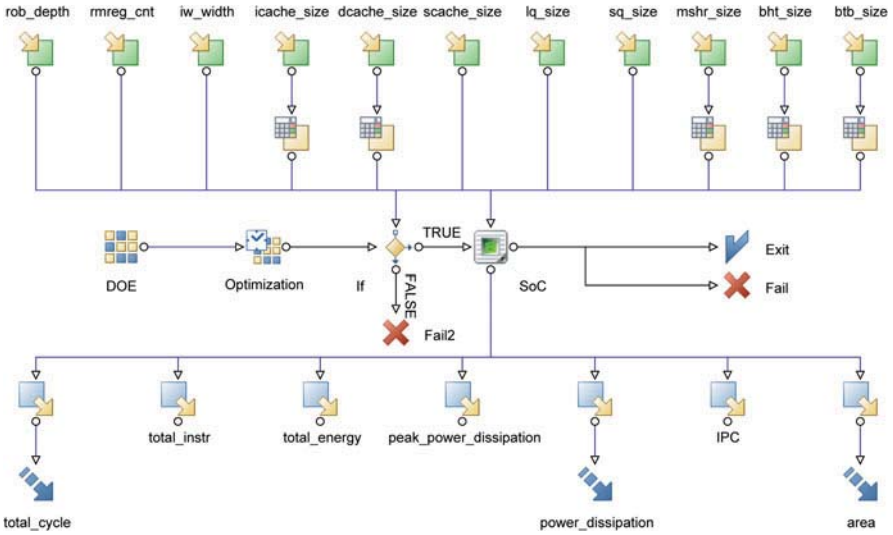

Fig. 8.1 The modeFRONTIER optimization workflow

The worklow specifies both a data path and a process path, which represent respectively, the data flow for the evaluation of a design and the optimization execution flow as described below.

The data path, which flows vertically from the top to the bottom, contains one input node for each configuration parameter and one output node for each metric produced by the simulator. In this optimization task there are eleven input nodes and seven output nodes. The System-on-Chip node (labeled as SoC) represents the interaction with the simulator, and takes care of the interface between the exploration tool and the simulator, passing values for the configuration parameters and getting back the values of the metrics as produced by the simulator. The three objectives to be minimized are represented with the arrow nodes.

The process path, which flows horizontally from left to right, starts with the Design of Experiments node (DoE), which generates the initial set of configurations to be evaluated. The optimization node (Optimization) guides the exploration of the design space using an optimization algorithm to generate the subsequent configurations that will be evaluated based on the performance of previous evaluations. The conditional node (If) pass only feasible designs for evaluation to the System-on-Chip node. Successful executions of the simulator will generate a valid design (Exit), and errors will generate failed designs (Fail).

8.2.3.2 Design of Experiments

The Design of Experiments (DoE) technique is used to generate the initial set of designs for evaluation. The aim is to define a limited number of test runs that allow to maximize the knowledge gained by eliminating redundant observations, in such a

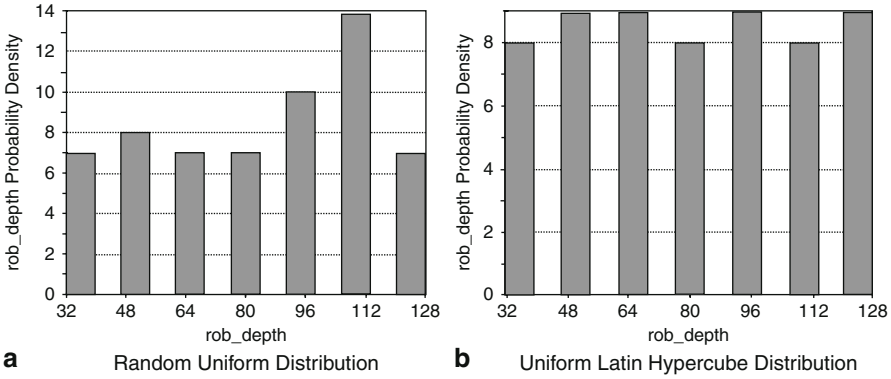

Fig. 8.2 The marginal distribution for the parameter rob_depth with a **a** "Uniform Random" and **b** "Uniform Latin Hypercube" Design of Experiments with 60 designs. It can be appreciated that the distribution of samples for each value is better in **b**, showing that the second approach has provided a better sampling of this discrete design space

way, that the time and resources needed to make the experiments is reduced as much as possible.

In this experiments, a Uniform Latin-Hypercube algorithm [4] is selected to generate 60 initial designs. The selection of the DoE algorithm and the number of designs is guided by the fact that only 8,134 designs can be evaluated and also, as described in Chap. 3, that this algorithm is preferred over the widely-used Uniform Random algorithm, particularly due to the better mapping of the marginal distribution functions with a small number of samples, as clearly shown in Fig. 8.2.

8.2.3.3 Optimization

The algorithm MFGA (Magnifying Front Genetic Algorithm) [7], described in Chap. 3, is selected by considering the characteristics of the optimization problem. This algorithm has been developed in the MULTICUBE project to handle categorical discrete optimization problems. A non-generational operation mode (steady state) makes it well suited for problems involving long simulation time. This algorithm has only two configuration parameters, which are set to default suggested values: crossover probability to 0.9 and mutation probability to 0.15.

The optimization is performed by using this algorithm till the allowed number of evaluations is reached without any manual intervention. Figure 8.3 shows the design space evaluation, highlighting the Pareto front as obtained at the end of the evolutionary process[1]. The values of the metrics (normalized due to confidentiality reasons) are plotted for each evaluated design with a bubble which size is directly

[1] Even if unusual, a bubble graph with the third dimension as the bubble size was preferred over a typical 3D graph since the large number of points produces a difficult to understand 3D cloud-like graph

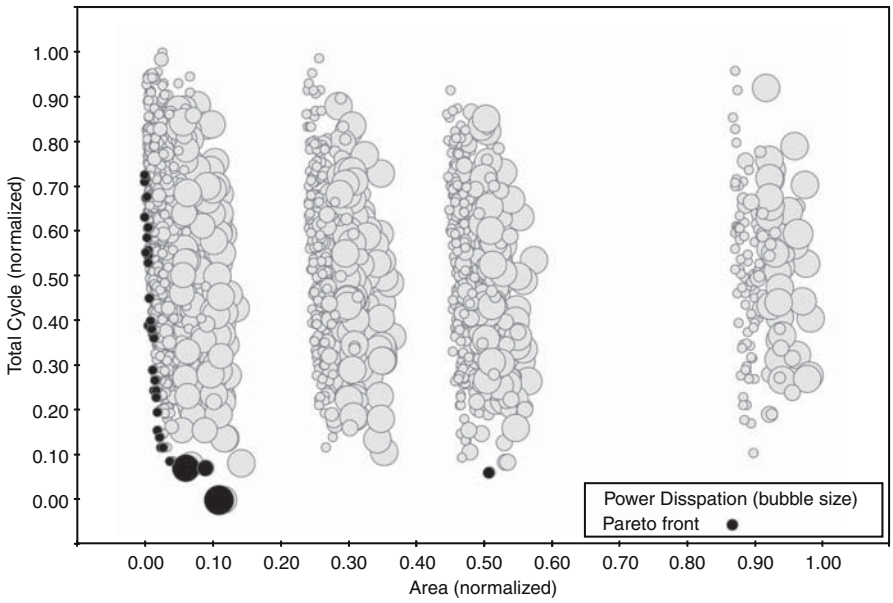

Fig. 8.3 The Pareto front in the design space

proportional to the Power Dissipation. The x and y axis correspond respectively to the Area and the Total Cycle. Note that designs with lower Occupation Area, lower Total Cycle and lower Power Dissipation are preferred. The set of Pareto designs is represented with black bubbles.

Figure 8.4 presents a comparison between the Pareto front obtained in this experiment and the Pareto front obtained by a semi-automatic approach guided by statistical analysis, which was performed as described in Sect. 3.4.2. As the figure clearly shows, the Pareto front obtained by the fully automatic procedure dominates all points generated by the semi-automatic approach. This is an important result since the automatic procedure was performed without any manual intervention, while the semi-automatic procedure was based on the designer ability and experience to assess the results and based on that, select the next instance of the model to be simulated.

Table 8.3 presents some quantitative results of the comparison between the Pareto fronts obtained with the automatic optimization and the semi-automatic approach. The enhancement on Area and Total Cycle metrics are displayed in percentage of the whole design space range grouped by their power dissipation characteristics. It can be appreciated that more than interesting enhancements were obtained by the automatic procedure.

8.2.3.4 Other Studies

Once a well organized experiment provides enough data, it is possible to study some properties of the design space. One of the most important analysis is the correlation

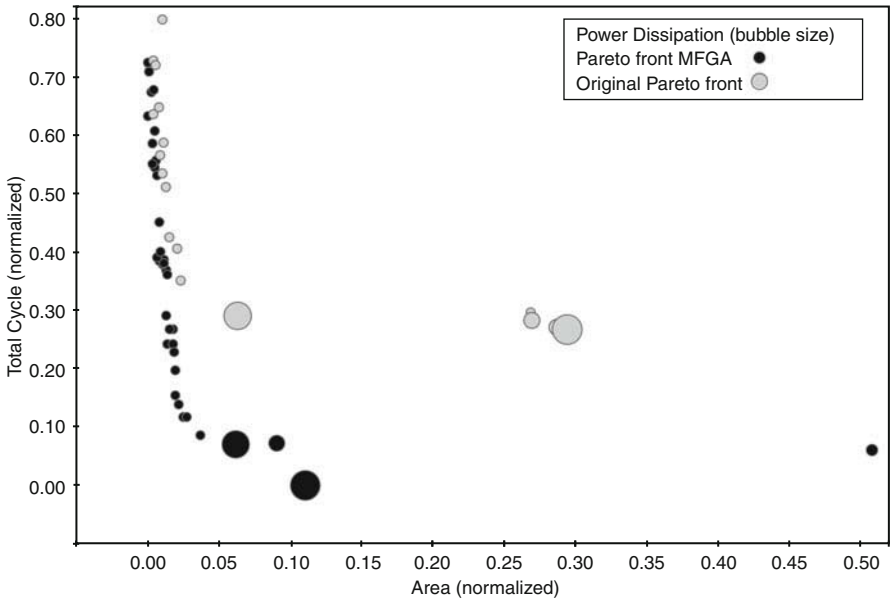

Fig. 8.4 A comparison between the Pareto front obtained with the automatic optimization and the semi-automatic process based on statistical analysis

Table 8.3 Enhancement on Area and Total cycle metrics when compared with the statistically-based approach grouping designs by Power dissipation

Power dissipation	Area	Total cycle %
Low power dissipation	Comparable	−7.71
Medium power dissipation	−17.35%	−18.42
High power dissipation	Comparable	−28.99

study, which allows to identify relations between parameters and metrics. There are many tools that can be used, however, the Self Organizing Map (SOM) [3] is particularly well suited for high dimensional spaces. Its main advantage compared with other methods is that it allows to identify not only the global correlations between the variables, but also highlights information about local correlations.The SOM is a 2D map where the parameters and metrics are automatically mapped during the learning process. Figure 8.5 shows the SOM obtained after the optimization process described in previous section.

Variables (parameters and metrics) that are correlated are mapped to nearby areas of the map. The spatial location of the variables total energy (total_energy), power dissipation (power_dissipation) and data cache size (dcache_size), which are all mapped in the lower-right corner of the map, allows to conclude that there is a strong correlation between them. There is also a large correlation between the occupation area (area) and the secondary cache size (scache_size), and between total cycle (total_cycle) and store queue size (sq_size).

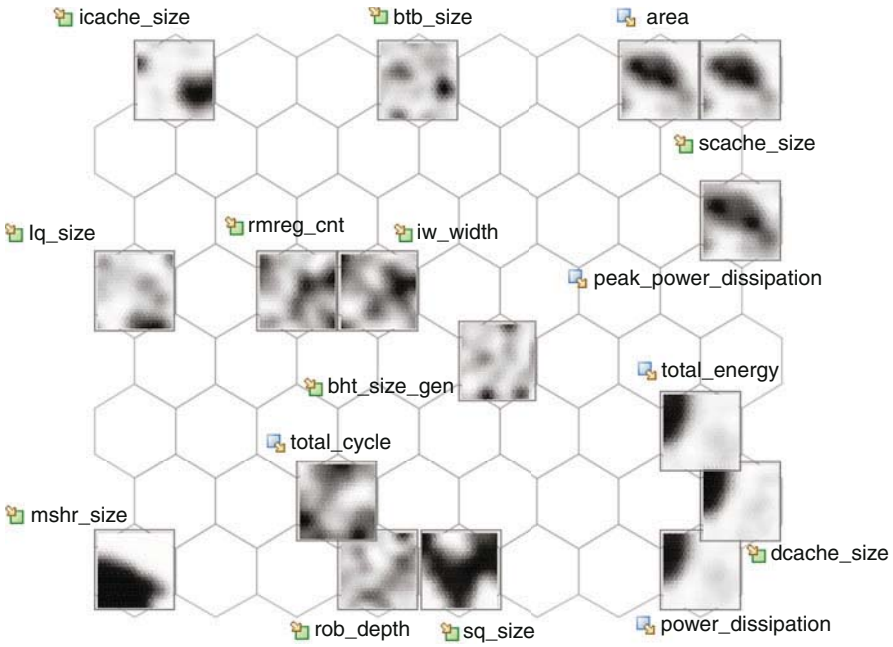

Fig. 8.5 The Self Organizing Map describing correlations between parameters and metrics. Some low correlations parameters and metrics have been removed from the diagram to enhance readability

The SOM provides also more information: the gray-scale patterns indicate the locations where the values of the variables are mapped, with light gray corresponding to low values and dark gray corresponding to large values. Just as an example, the darker area in the upper-left corner of the pattern of total energy (total_energy) indicates that higher values of this variable were mapped in the upper-left corner of the map. In this way, similar patterns in grouped variables describe a direct correlation, while inverted color patterns describe an inverse correlation. The analysis suggests there is a direct correlation between occupation area (area) and static cache size (scache_size), while there is an inverse correlation between total cycle (total_cycle) and store queue size (sq_size). Local correlations can also be identified by studying particular areas of the gray-scale patterns.

8.2.4 Conclusions

The automatic optimization process allowed to obtain a better Pareto front when compared with equivalent experiments performed with a semi-automatic approach guided by a statistical analysis. Not only the Pareto front was better at the end of the optimization, but after the evaluation of only 3,000 designs, the obtained Pareto

front was better than the front obtained by the statistical analysis guided optimization which consisted in 14,216 evaluations (see Chap. 3).

There are other advantages of the automatic exploration procedure. In the automatic exploration, all data concerning previous evaluations are always stored in a structured database. The designer, not only will not be stuck on repetitive operations, but can focus his/her attention (and profit from his/her experience) in analyzing the designs database in a statistical manner. In this way, analysis like clustering or correlations can be performed in a more systematic and organized way.

Moreover, the automatic exploration is driven by an optimization engine based on several optimization algorithms, whereas the manual exploration is based on designer ability and experience to assess the results and to move towards the next instance of the model to be simulated. In that particular case (see Chap. 3), wrong assumptions on the statistical distributions of data reduced the possibility to obtain better solutions.

8.3 Design Case Study: Analysis of Performance and Accuracy of Response Surface Models for the Many-Core Architecture Provided by ICT

This section describes an example application of response surface models on a many-core architecture by describing how the models are selected, trained and validated to provide the best accuracy without sacrificing performance.

8.3.1 Platform Description

The ICT many-core (named *Transformer*) is a tiled, multiple-instruction, multiple-data (MIMD) machine consisting of a 2D grid of homogeneous, general-purpose compute elements, called cores or tiles. Instead of using buses or rings to connect the many on-chip cores, the transformer architecture combines its processors using 2D mesh networks, which provide the transport medium for off-chip memory access and other communication activity.

By using mesh networks, the Transformer architecture can support anywhere from a few to many processors without modifications to the communication fabric. In fact, the amount of in-core (tile) communications infrastructure remains constant as the number of cores grows. Although the in-core resources do not grow as tiles are added, the bandwidth connecting the cores grows with the number of cores.

As Fig. 8.6 shows, the Transformer Processor Architecture consists of a configurable 2D grid of identical compute elements, called nodes. Each node is a powerful computing system that can independently run an entire application. The perimeters of the mesh networks in a Transformer Processor connect to memory controllers, which in turn connect to the respective off-chip DRAMs through the chips pins. Each node combines a processor and its associated cache hierarchy with a router,

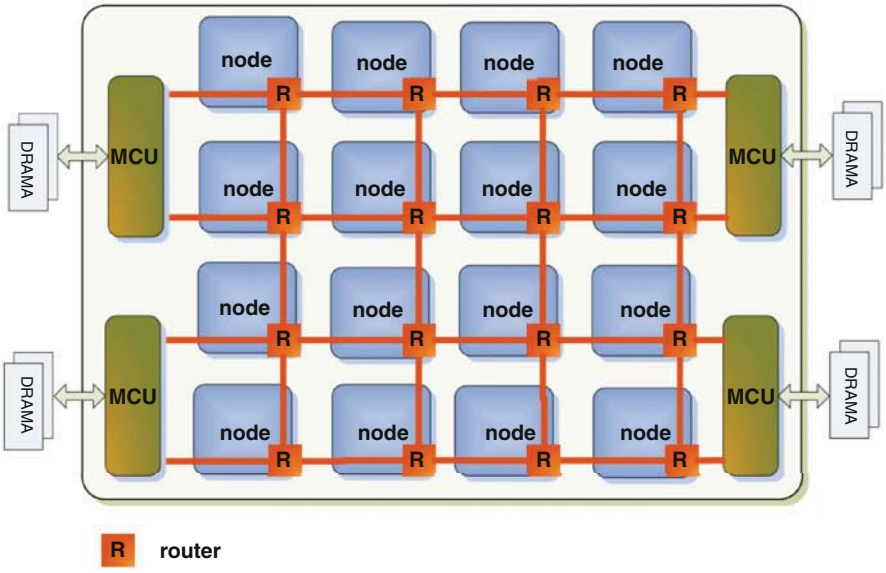

Fig. 8.6 The ICT Many-Core architecture (*transformer*)

which implements the interconnection network. Specifically, each node implements a two-issue out-of-order pipelined processor architecture with an independent program counter and a two-level cache hierarchy. The memory sub-system is *distributed* (thus not shared among cores) to simplify the architecture of the system.

8.3.2 Node Architecture and Instruction Set

For each node there is a separate level 1 instruction/data cache and a unified level 2 cache in each node. As there is need for 64-bit computing in multimedia applications, 64-bit MIPS-III ISA is chosen for the node. Some MAC (Multiply-Accumulate) instructions which need three source operands are also supported by node.

Each node implements a two-issue out-of-order pipelined processor architecture, as shown in Fig. 8.7. The pipeline is divided into 6 stages: fetch, decode, map, issue, execute and write back. For memory operations, there is one more stage *data cache* between *execute* and *write back*. In order to achieve reasonable performance and implement a low complexity node, a *scoreboard*-based out-of-order pipeline based is used. The scoreboard unit in each node is responsible for accepting decoded instructions from the *map* stage, and issuing them to the functional units (address generators, ALUs and FPUs) satisfying dependencies among the instructions. To achieve this goal the main element of the scoreboard is the instruction queue which holds decoded instructions and issues them once the resources they require are free and their dependencies have been cleared.

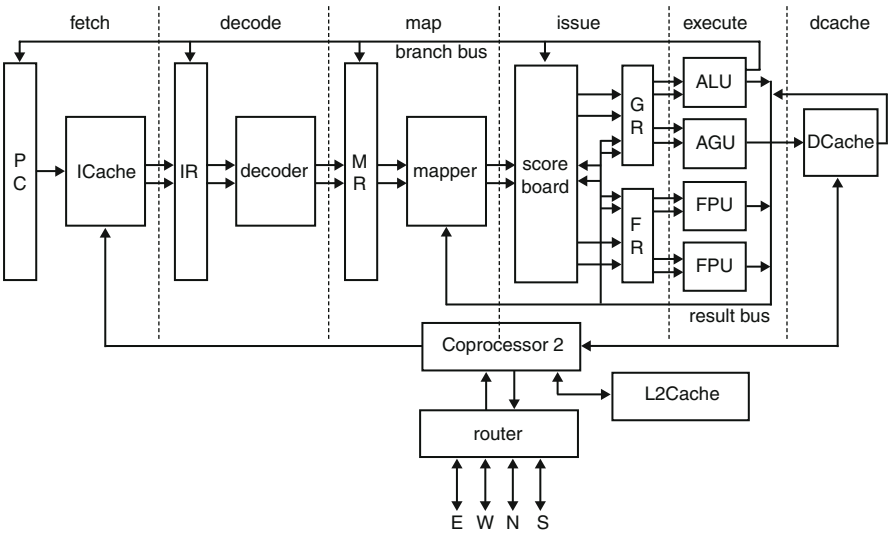

Fig. 8.7 The ICT Many-Core pipelined architecture of a node

8.3.3 The Simulator

The NoC based many-core design transformer is implemented in C++ as a cycle-accurate simulator. The simulator accepts the configuration as an input file, and generates the system metrics as an output file. Both input and output files are defined in XML format by following the rules described in the Design Space Definition file, as described in Chap. 1.

Figure 8.8 shows the scheme of a single core. A core defines an instruction queue, whose entry is allocated to the newly instructions fetched from instruction cache, and released after the instruction was committed. In detail, an instruction fetched from instruction cache (1) is allocated in the instruction queue (2), and at the same time, the instruction dependency is built (3). Then, if the required logic unit is idle and the necessary dependency is satisfied, the instruction is issued (4) to the corresponding logic unit, and the state of the logic unit is changed to *busy* in order to block the instructions using it (5). After the predefined latency and essential operations (e.g. accessing the register file, accessing the local data cache, and routing message in the mesh), the instruction is committed, and simultaneously those interrelated resources and dependencies are released. In above mentioned process, the timing information is accurately collected to implement a cycle-accurate simulator.

The power model is based on the The Princeton University's Wattch power model [1], with all parameters updated accordingly to a 0.13 μm process.

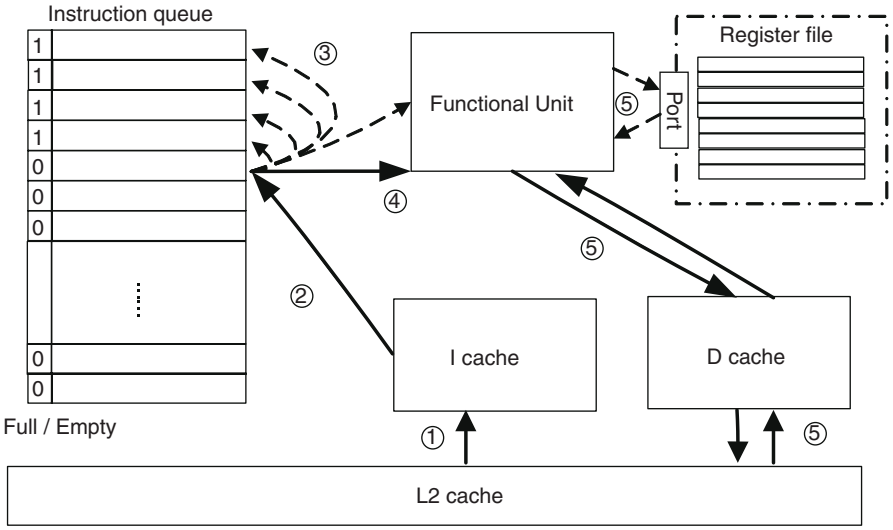

Fig. 8.8 The ICT Cycle-Accurate Simulator

8.3.4 Design Space and Application

One goal of the design of Transformer is to make a reconfigurable platform, allowing further tuning as the evaluation works goes on. According to the system metrics, the designers can tune the hardware arguments for higher performance. In the simulator, all the tunable arguments are collected in a configuration file in XML style. When the simulator is started, it read the arguments from this configuration file. The design space, which is shown in Table 8.4, is composed of 1,134 design points. As an additional rule in the design space, the associativity of the level-2 cache should be greater than the sum of the associativities of the level 1 instruction and data

Table 8.4 Parameter space for the ICT many-core platform

Parameter	Minimum	Maximum
Mesh_order	2	8
Cache_block_size	32	64
ICache_ways	2	16
Icache_entries	128	512
DCache_ways	2	16
DCache_entries	128	512
L2Cache_ways	4	32
L2Cache_entries	128	512
L2Cache_access_latency	3	10
Memory_size	8	32
Memory_access_latency	30	100
Router_buffer_entries	2	8

A00	A01	A03	A04
A10	A11	A12	A13
A20	A21	A22	A23
A30	A31	A32	A33

B00	B01	B02	B03
B10	B11	B12	B13
B20	B21	B22	B23
B30	B31	B32	B33

C00	C01	C02	C03
C10	C11	C12	C13
C20	C21	C22	C23
C30	C31	C32	C33

Fig. 8.9 Parallel multiplication of matrices

caches. The system metrics associated with the architecture are the following: cycles, instructions, power dissipation, peak power dissipation and area.

Application The application used for the following experimental results is a classic implementation of *partitioned* matrix multiplication. Each matrix (both sources and destination) is divided into an equal number of sub-matrices; each node works by multiplying the rows and columns needed for a single partition of the destination matrix. Figure 8.9 shows an example matrix multiplication ($C = A \times B$) for two squared matrices divided into 16 sub-matrices ($A_{i,j}$, $B_{i,j}$, $C_{i,j}$ where $i, j \in \{0, 3\}$).

8.3.5 Response Surface Modeling of Many-Cores

This section presents the results of the validation of RSMs described in Chap. 4. For the sake of synthesis, in this section the results related to the following RSM configurations are reported:

- *Linear regression*
  - Model order: *first*,
  - *Without any interaction between parameters*,
  - *excluding* the following design space parameters from metric estimation:
    - *ICache_ways*,
    - *DCache_ways*,
    - *L2Cache_ways*,
    - *L2Cache_access_latency*,
    - *Memory_size*,
    - *Memory_access_latency*.
- *Radial Basis Functions*
  - Distance function definition: *thin plate spline*.
- *Splines*
  - No parameters.
- *Kriging*

  - Variogram Type: *Gaussian*,
  - Autofitting Type: *Maximizing Likelihood*,
- *Evolutionary Design*
  - Crossover Depth: *10*,
  - Generations Number *1000*,
  - Population Size: *500*,
- *Neural Networks*
  - Network Size Policy: *Automatic*.

To enable the validation of the proposed RSMs, all the simulations associated with the complete design space (1,134 design points) have been performed. The resulting data has been used to train the RSMs and to validate the predictions. The chosen sizes for the training sets are: 100, 200, 300, 400, 500, 600, 700, 800, 900 and 1,000 design space points. For each training set size, 5 training sets have been generated with a pseudo-random process from the simulation data, hence the total number of training sets is 50. Given a trained RSM, the average normalized error of the predictions over the complete design space (of 1,134 points) is computed. For each RSM, the three values of the Box-Cox transform $\lambda = \{0, 0.5, 1\}$ were evaluated. Overall, for each RSM, 150 validation tests were performed.

Figure 8.10 reports ε versus the training set size, for all the experimented RSMs, where:

$$\varepsilon = \frac{\eta_1 + \eta_2 + \eta_3 + \eta_4 + \eta_5}{5} \tag{8.1}$$

and η_i, with $1 \leq i \leq 5$, is the *average normalized error* corresponding to the error observed training the RSM with the $i - th$ training set of the corresponding size. In Fig. 8.10, ε is the *average normalized error* on the y-axis, while the x-axis shows the training set size.

Linear Regression, Radial Basis Functions and Splines have been plotted with a scale from 0 to 0.7 (equivalent to 70% error); Kriging, Neural Networks and Evolutionary Design have a scale ranging from 0 to 0.045 (equivalent to 4.5% error).

Overall, there is evidence that the Neural Networks (Fig. 8.10f) allows for the best error for a given training set size (less than 0.2% error after 100 training samples). On the other hand, the Linear Regression RSM seems to be not appropriate for modeling such type of use case (Fig. 8.10a) since it provides the highest error while not scaling with the training set size.

As can be seen, the logarithmic box-cox transformation is crucial for reducing the variance of the data and improve the model prediction. We further analyze the behavior for this particular Box-Cox transform in Fig. 8.11. The Figure shows the statistical behavior (computed over 5 runs on the same training set size for each of the RSMs). As can be seen, Evolutionary Design, Kriging and Neural Networks provide a strong improvement with respect to Linear Regression and Splines (their scale has been reformatted accordingly), while Radial Basis Functions are overall halfway.

However, for the ICT use case, Evolutionary Design and Neural Networks have revealed the highest execution times (the validation for Evolutionary Design lasted for

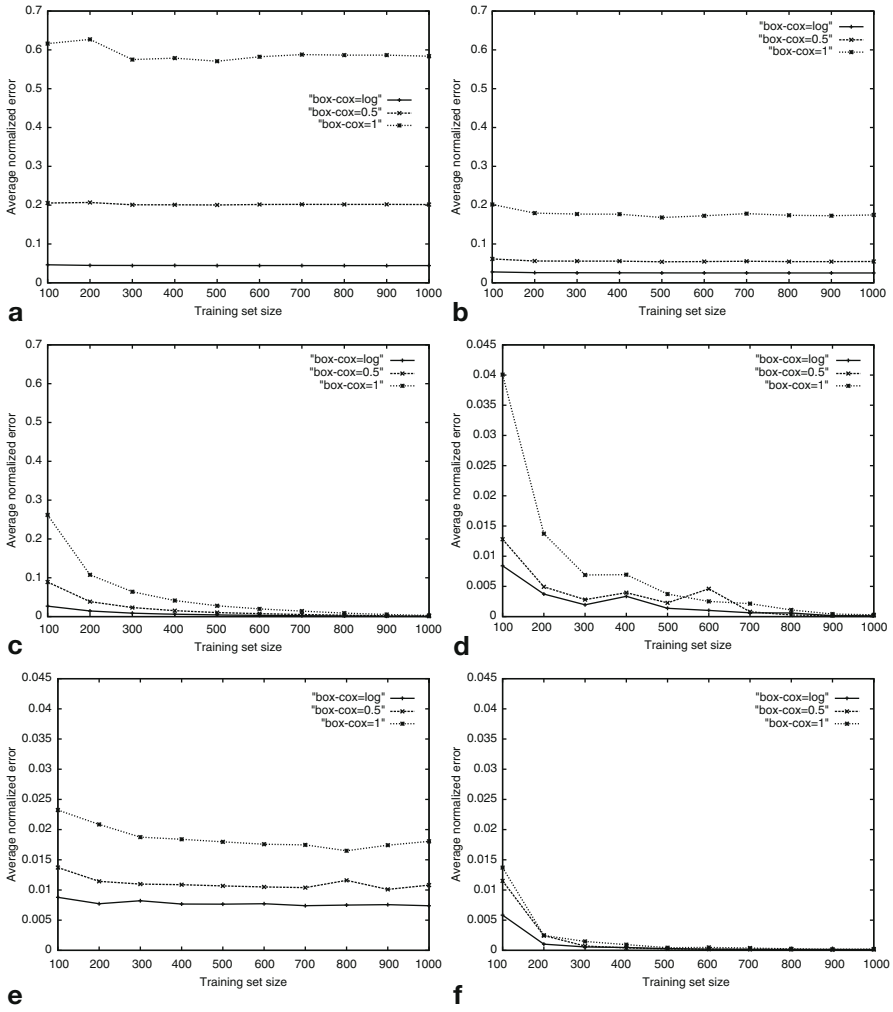

Fig. 8.10 Average normalized error versus training set size for the experimented RSMs. **a** Linear regression, **b** Splines, **c** Radial basis functions, **d** Kriging, **e** Evolutionary design, **f** Neural networks

several days, while Neural Network took few hours), Splines and Linear regression have been the fastest RSMs (the validation time for both of them was few minutes), while Radial Basis Functions and Kriging presented an intermediate behavior (the overall validation time was several minutes).

Overall, the results have showed that different models can present a trade-off of accuracy versus computational effort. In fact, throughout the evaluation, we observed that high accuracy models require high computational time (for both model construction time and prediction time); vice-versa low model construction and prediction time has led to low accuracy. We can sum up by observing that the best choice for a specific use case depends on the desired accuracy and the constraints on the

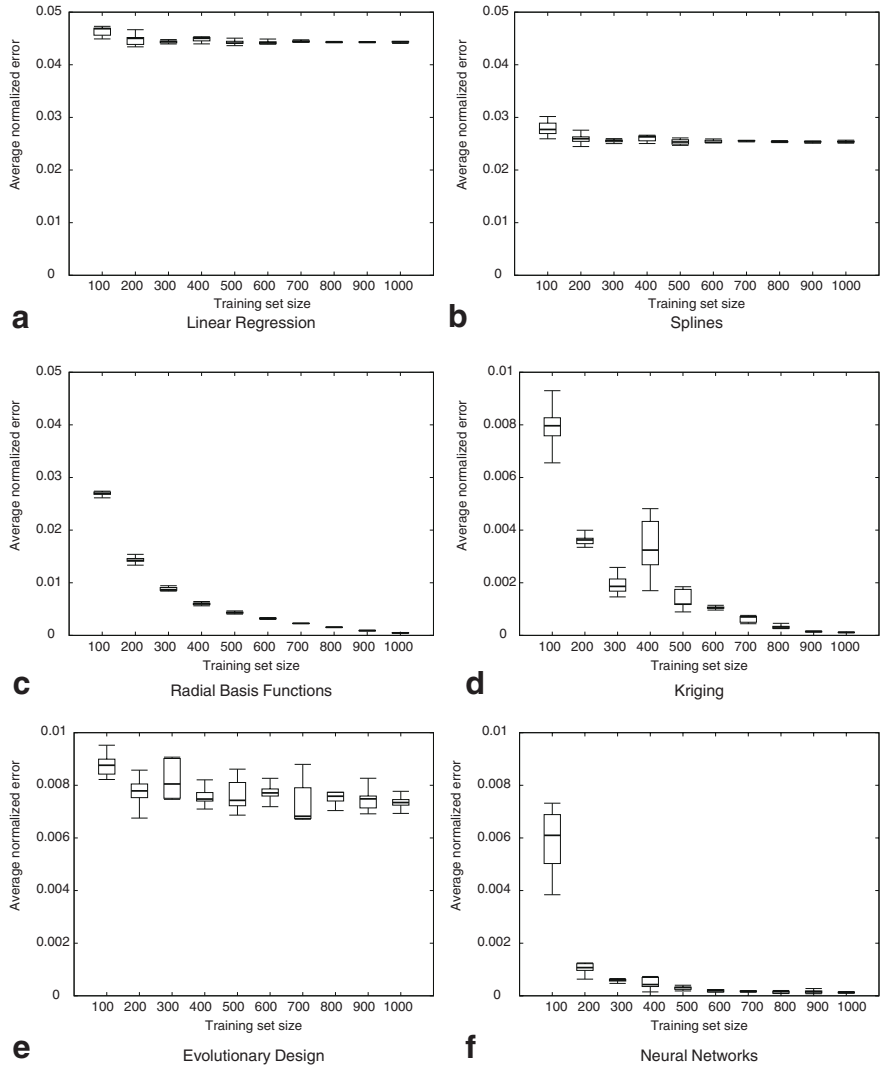

Fig. 8.11 Box plots of average normalized error versus training set size for logarithmic box-cox transformation. **a** Linear regression, **b** Splines, **c** Radial basis functions, **d** Kriging, **e** Evolutionary design, **f** Neural networks

prediction time. Referring to the use-case addressed in this section, however, we discovered an exception to the previous *empiric* rule when, for $\lambda = 0$, Kriging gave a better accuracy than Evolutionary Design for every training set size, paired with a lower computation time.

8.4 Conclusions

This chapter presented two significant applications of the MULTICUBE design space exploration framework. While in the first part has been presented the design space exploration of a low power processor developed by STMicroelectronics by using the modeFRONTIER tool, the second part has described the application of RSM models developed within MULTICUBE to a tiled, multiple-instruction, many-core architecture developed by ICT China. The presented analyses refer to a direct application the methodology implemented in the project to real case studies.

References

1. Brooks, D., Tiwari, V., Martonosi, M.: Wattch: a framework for architectural-level power analysis and optimizations. In: ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture, pp. 83–94. ACM, New York, NY, USA (2000)
2. ESTECO: modeFRONTIER, Multi-Objective Optimization and Design Environment Software. http://www.esteco.com
3. Kohonen, T.: Self-organized formation of topologically correct feature maps. Biological Cybernetics **43**(1), 59–69 (1982)
4. McKay M. D., C.W.J., J., B.R.: Latin hypercube sampling: A comparison of three methods for selection values of input variables in the analysis of output from a computer code. Technometrics **22**(2), 239–245 (1979)
5. MIPS Technologies: Architecture Programming Publications for MIPS32. http://www.mips.com/products/product-materials/processor/mips-architecture
6. MULTICUBE: Final Prototype of the Open-Source MULTICUBE Exploration Framework. Deliverable 3.1.2 (2009)
7. Turco, A., Kavka, C.: MFGA: a GA for complex real-world optimisation problems. International Journal of Innovative Computing and Applications **3**(1), 31–41 (2011)
8. Standard Performance Evaluation Corporation: SPEC CPU 2000. http://www.spec.org/cpu2000