

Static Task Mapping for Tiled Chip Multiprocessors with Multiple Voltage Islands

Nikita Nikitin and Jordi Cortadella

Universitat Politècnica de Catalunya, Barcelona, Spain

Abstract. The complexity of large Chip Multiprocessors (CMP) makes design reuse a practical approach to reduce the manufacturing and design cost of high-performance systems. This paper proposes techniques for static task mapping onto general-purpose CMPs with multiple pre-defined voltage islands for power management. The CMPs are assumed to contain different classes of processing elements with multiple voltage/frequency execution modes to better cover a large range of applications. Task mapping is performed with awareness of both on-chip and off-chip memory traffic, and communication constraints such as the link and memory bandwidth. A novel mapping approach based on *Extremal Optimization* is proposed for large-scale CMPs. This new combinatorial optimization method has delivered very good results in quality and computational cost when compared to the classical simulated annealing.

Keywords: Chip Multiprocessing, Task Mapping, Power Management, Extremal Optimization

1 Introduction

Chip-multiprocessing (CMP) is becoming a major trend to take advantage of Moore's law under the power consumption limitations dictated by the heat dissipation problems in high performance computing systems. Commercial and prototype implementations have shown the performance gains achieved by CMPs having up to a hundred cores [1–3]. As we move down to deep nanometric technologies, the design complexity of such systems increases significantly. Manufacturing costs and time-to-market compromise the viability of new products that are customized for specific applications.

Design reuse is a pragmatic solution to this problem, in both CMP design and deployment. For an effective reuse during deployment, CMPs are designed general-purpose, to support a variety of applications. Hence, a methodology for efficient mapping of applications onto CMPs is essential. Many approaches have been proposed to solve the mapping problem for application-specific and multiprocessor on-chip systems (SoCs) [4]. However, there are significant differences between the SoCs and CMPs, that are of the great importance for the mapping problem. To understand these differences we have to consider two aspects of CMPs: the *tiled architecture* and organization of *power management*.

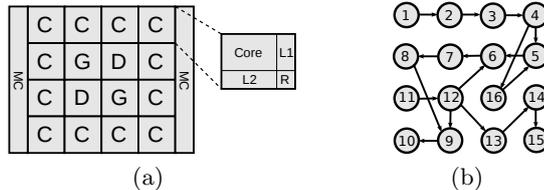


Fig. 1: (a) Tiled CMP; (b) task graph to be mapped onto CMP.

Tile replication was proposed for many-core CMPs in order to reduce the design time [2, 3]. This led to the concept of *tiled architecture*, characterized by regular structures of homogeneous or heterogeneous tiles [1, 5, 6]. Such systems may include specialized processors (e.g., graphics, DSP) or different implementations of the same architecture (e.g., in-order/out-of-order) with varied power-performance trade-offs. Figure 1(a) depicts a CMP with three classes of tiles: general-purpose cores (C), cores with graphics units (G) and DSPs (D). Each tile also incorporates cache memories of two levels (L1, L2) and an on-chip router (R), connecting it to the interconnection fabric. Two memory controllers (MC) are placed at the periphery to provide communication with the off-chip memory.

To assure the performance and thermal properties of the system, CMPs are designed to operate under a certain power budget. One of the most effective ways to manage power is to floorplan various voltage islands and assign the best voltage and frequency for each core [7]. Unfortunately, the number of voltage islands is constrained by the design of the power delivery network and costly implementation of voltage regulators [8]. It is therefore realistic to consider that future CMPs will have many cores and voltage islands with several cores (e.g., 4 or 8). This fact imposes an additional constraint in the task mapping problem: even though some cores could possibly run at lower voltages and frequencies, sharing the island with other cores may prevent from having this flexibility.

Up to now, the research on power-aware mapping has assumed that the voltage islands are defined pre-silicon during task mapping in SoCs, often disregarding the cost of their implementation. A broad overview of the related work on SoC application mapping and island planning can be found in [4]. The approach in [9] considers performance constraints, but does not account for the communication power. A more realistic approach is proposed in [10] in which computation and communication are optimized taking into account a component related to voltage shifters. The work in [11] emphasizes the importance of optimizing both static and dynamic CMP power and describes efficient heuristics for scheduling of streaming applications. Thermal-aware island creation via evolutionary algorithms was proposed in [12]. The distinction of processor classes was introduced in [13], but assuming that every processor runs at independent voltage level.

1.1 Task Mapping for Tiled CMPs

The mapping problem we want to address differs from previous ones in that the CMP is assumed to be already manufactured and reused for several applications (similarly to FPGA reconfiguration). Therefore, the voltage islands have been

already planned and the maximum bandwidth of the links is also known a priori. Another peculiarity of CMP mapping (as opposed to SoCs), captured by this work, is the presence of traffic to the off-chip memory. Finally, the proposed method is scalable to handle systems with hundreds of cores.

The work in [14] offers a framework for accurate compiler-level mapping of applications onto homogeneous mesh CMPs through analysis of the instructions and allocation of data. The approach presented in this paper differs by considering the variety of processing units, offered by heterogeneous CMPs.

The examined problem consists of *statically mapping* a set of parallel tasks onto a many-core CMP and selecting the voltages of islands so that the total system power is minimized. Every task is considered as an infinite process (encountered in control, automotive and robotics systems) and has an associated throughput constraint that guarantees the required QoS for that task. A variety of processor classes is supported, each one characterized by a set of voltage/performance/power parameters used to find the best performance/power trade-off for each task. It is assumed that the cores are organized in a mesh with XY-routing [15]. The main contributions of this work can be summarized as:

- Mathematical formulation of a mixed-integer linear programming problem (MILP) delivering optimal mapping solutions for the examples of small size.
- Heuristical mapping optimization by Simulated Annealing (SA).
- Scalable approach based on *Extremal Optimization* (EO) [16], shown to outperform the optimization by SA, both in quality and computational cost.

Due to the lack of space and to provide the reader with intuition on the EO-based approach, we only focus on this algorithm in the paper. The MILP and SA mapping are presented in detail in the technical report [17].

The structure of the paper is as follows. Next section presents an overview of the problem by considering a small example. Section 3 presents the definition of the problem. The proposed solution technique based on Extremal Optimization is explained in Section 4. Section 5 discusses the experimental results.

2 Problem Overview

This section illustrates the task mapping problem using a small example. Let us assume a task graph with four tasks (Fig. 2(a)). There are three flows between the tasks, with the bandwidths specified in the arcs of the graph (in Gbps). Figure 2(b) depicts a CMP with four processors. There are two classes of processors: C_1 (light) and C_2 (dark). The task graph must be mapped onto the CMP.

Communication-optimal mapping. Figure 2(c) shows a mapping that optimizes the communication metric, i.e. the product of bandwidth and hop-count. Assuming the distance between the neighboring processors is one hop, the cost of the mapping is $CCost_1 = 1.0 \cdot 1 + 1.0 \cdot 1 + 0.5 \cdot 2 = 3.0$ (Gbps).

Throughput-feasible mapping. Now let us take into account the processor parameters and consider the throughput requirements of the tasks. Figure 2(d) describes these parameters. Processors can operate at two voltages, 1.0 and 0.8V. The corresponding frequency (F , in GHz) and power (P , in W) for each voltage

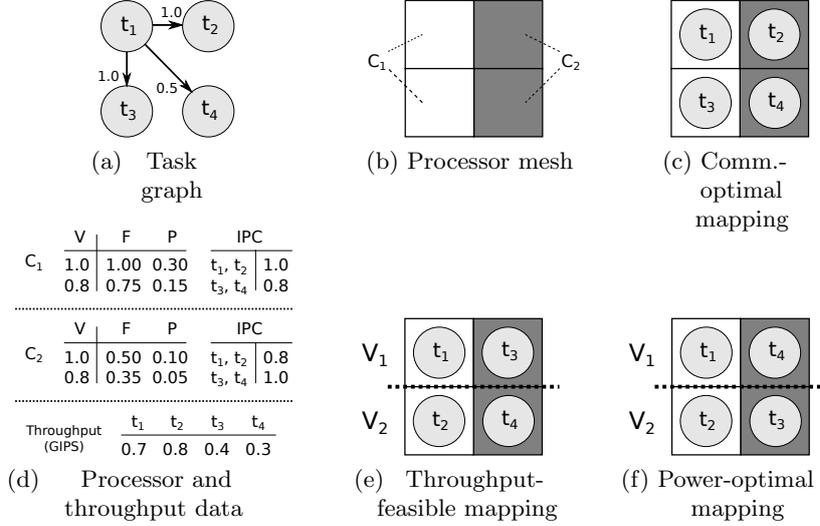


Fig. 2: Task mapping example.

is shown in the tables. Tasks may be executed with a different performance (Instructions Per Cycle, IPC) in each class of processors. Each task may also require a specific throughput (given in giga-*IPS* in Fig. 2(d)).

The mapping in Fig. 2(c) is infeasible with the introduction of the throughput constraints. Consider task t_2 assigned to a C_2 -processor. The maximum performance that C_2 can provide for t_2 is $IPC(t_2) \cdot F(1.0V) = 0.8 \cdot 0.5 = 0.4$ *GIPS*, while the throughput requirement for t_2 is 0.8 *GIPS*. To satisfy the requirements, tasks t_2 and t_3 are swapped (Fig. 2(e)). This mapping satisfies the throughput constraints and still keeps the optimal value for the communication metrics.

Power-optimal mapping. As a final step, let us consider the partitioning of the CMP into voltage islands. Let us assume the CMP has two islands, separated by the bold dotted line, as shown in Fig. 2(e). Processors in the same island must operate at the same voltage level, that is the minimal voltage required to satisfy all the throughput constraints for the tasks mapped to this island.

For the mapping in Fig. 2(e), the upper island has to operate at 1.0V dictated by the throughput constraint of t_3 . The lower island also has to run at 1.0V, because of t_2 . Thus, the computation power, calculated using the data from Fig. 2(d), is $P_{comp} = 0.30 + 0.10 + 0.30 + 0.10 = 0.80$ W. Let the energy to transfer one bit for one hop be $E_{bit} = 0.1nJ/bit$. Then the communication power $P_{comm} = CCost_2 \cdot E_{bit} = 3.0Gbps \cdot 0.1nJ/bit = 0.3$ W, and the total power $P = P_{comp} + P_{comm} = 1.10$ W. Notice that if tasks t_3 and t_4 are swapped (Fig. 2(f)), the upper island can lower the voltage to 0.8V without violating the throughput. The new computation power is $P_{comp} = 0.15 + 0.05 + 0.30 + 0.10 = 0.60$ W. The communication cost is increased: $CCost_3 = 1.0 \cdot 1 + 1.0 \cdot 2 + 0.5 \cdot 1 = 3.5$ (*Gbps-hop*), so the communication power becomes $P_{comm} = CCost_3 \cdot E_{bit} = 3.5 \cdot 0.1 = 0.35$ W. However, the total power $P = 0.95$ W decreases, making the assignment in Fig. 2(f) the best one in terms of total power efficiency.

3 Problem Definition

This section defines the mapping problem. A *task graph* $TG(\mathcal{T}, \mathcal{F})$ is a directed graph with vertices representing the tasks $t_i \in \mathcal{T}$. Each arc represents a communication flow $f_{sd} \in \mathcal{F}$ from task t_s to t_d , with the minimum required bandwidth B_{sd} . Every task t_i has a throughput constraint $\text{IPS}(t_i)$, that is the minimum number of instructions per second required to provide the service delivered by the task. $\Lambda(t_i)$ defines the total traffic rate between t_i and the memory controller. The ratio between the traffic to and from the controller is specified by the parameter ρ . Note that the $\Lambda(t_i)$ value can be approximated, given the amount of data, operated by the task (i.e. the working set), and the size and miss ratio of the tile cache.

A CMP is represented by a *mesh of processors* $PM(\mathbb{P}, \mathcal{L})$ with $W \cdot H$ cells, where \mathbb{P} is the set of processors and \mathcal{L} is the set of communication links with capacity Cap . The performance of p_j to execute task t_i , measured in instructions per cycle, is specified by the function $\text{IPC}(t_i, p_j)$. The processors may operate at different voltages. We assume a set of voltages $\mathcal{V} = \{v_1, \dots, v_V\}$ available for all processors. The frequency and power of p_j operating at voltage v_k are defined by the functions $F(p_j, v_k)$ and $P(p_j, v_k)$, respectively. Every p_j belongs to some voltage island ι_n , as defined by the island map \mathcal{I} .

A CMP has a set of controllers to access the off-chip memory. We assume that every processor p_j is associated with *one* controller. This assumption can be eliminated by definition of the probabilities to access different controllers for p_j . Function $\text{McDist}(p_j)$ returns the hop-count distance from p_j to the related controller. The McBw parameter sets the maximum controller bandwidth to guarantee the performance of memory access.

The *mapping problem* is to assign all tasks in \mathcal{T} to the processors in \mathbb{P} , minimizing total power consumption and satisfying a set of design and performance constraints. The assignment constraint restricts every processor to hold one task at most. The *voltage selection* constraints force all processors in the same island to run at one voltage. *Throughput* constraints assure that for every task t_i running on processor p_j at voltage v_k the throughput requirement is met, that is $\text{IPC}(t_i, p_j) \cdot F(p_j, v_k) \geq \text{IPS}(t_i)$. The *link capacity* and *memory bandwidth* constraints restrain total traffic through links and memory controllers, evaluated under the assumption of XY-routing.

Let the binary variables a_{ijk} define whether task t_i is mapped onto processor p_j operating at voltage v_k . The computation power can be written as:

$$P_{comp} = \sum_{t_i \in \mathcal{T}} \sum_{p_j \in \mathbb{P}} \sum_{v_k \in \mathcal{V}} a_{ijk} \cdot P(p_j, v_k).$$

The communication power has two terms: the on-chip communication, defined by the flows between the tasks, and the off-chip communication, due to the traffic to and from memory controllers. It can be defined as:

$$P_{comm} = \sum_{f_{sd} \in \mathcal{F}} B_{sd} \cdot h_{sd} \cdot E_{bit} + \sum_{t_i \in \mathcal{T}} \sum_{p_j \in \mathbb{P}} \sum_{v_k \in \mathcal{V}} a_{ijk} \cdot \Lambda(t_i) \cdot \text{McDist}(p_j) \cdot E_{bit},$$

where h_{sd} represents the hop-count of flow f_{sd} , and E_{bit} is the estimated energy for transmitting one bit over a link. The objective of mapping becomes:

$$\min : P = P_{comp} + P_{comm}. \quad (1)$$

The formal definition of the problem via MILP is available in the report [17].

4 Task Mapping with Extremal Optimization

This section proposes the *Extremal Optimization* (EO) [16] metaheuristic in application to the task mapping problem. As *Simulated Annealing* (SA) [18], it is inspired by equilibrium statistical physics. SA has been successfully applied in many EDA problems, mostly related to layout synthesis. Recently EO has emerged as a competitive alternative delivering superior results in quality and computational cost. The SA algorithm is outlined in [17]. This section shows how EO can be customized to effectively solve the task mapping problem.

Extremal optimization is guided by the principle of evolution in ecosystems, which were observed to evolve by selecting against its worst components. For every possible solution, EO evaluates the *fitness* of each component in the system. A high fitness value indicates that the component has a comfortable low-cost status in the system. EO focuses on improving the status of components with low fitness. At each iteration, some of the worst-fit components are replaced by other components that contribute to improve their fitness. Local optima are avoided by randomizing the selection process. The components are ranked according to their fitness in ascending order, and are randomly selected by some probability distribution biased towards the ones with lowest fitness values. The power-law distribution is a typical one for EO. For example, if the system has N components ranked from 1 to N in ascending order of their fitness, the index i of the selected component can be calculated as follows:

$$i = \lceil N \cdot p^\tau \rceil \quad (2)$$

where p is a random number obtained from a uniform distribution in the interval $[0, 1]$ and τ is the exponent of the power law.

For the task mapping problem, at each iteration EO selects a pair of tasks to be swapped: an unfavorable task (t_u) and a replacement task (t_r). EO uses information about the system cost when selecting the swapped tasks, that results into fast progress towards the final solution. In addition, EO accepts new solutions unconditionally, thus making the algorithm easier to tune.

The mapping problem can be considered as a multiobjective optimization problem, since the P_{comp} and P_{comm} terms of the cost function (1) depend on weakly related voltage level and hop-count values. It was observed in [19] that the multiobjective EO operates better by interleaving the optimization of individual objectives in time, rather than trying to optimize all of them simultaneously. This suggests to introduce different fitness functions for the optimization of the power components and alternate them at odd and even iterations of the algorithm.

Procedure 1 EXTREMALOPTIMIZATION

```

1:  $CurSol \leftarrow BestSol \leftarrow$  "Some initial solution"
2: while some improvement in the last  $k$  iterations do
3:   if even iteration then /* improve communication cost */
4:     sort all tasks in ascending order of  $\Phi_u^{comm}$ 
5:     select  $t_u$  according to equation (2)
6:     sort all tasks  $t_i \neq t_u$  in ascending order of  $\Phi_r^{comm}$ 
7:     select  $t_r$  according to equation (2)
8:   else /* improve computation cost */
9:     sort all tasks in ascending order of  $\Phi_u^{comp}$ 
10:    select  $t_u$  according to equation (2)
11:    sort all tasks  $t_i \neq t_u$  in ascending order of  $\Phi_r^{comp}$ 
12:    select  $t_r$  according to equation (2)
13:    swap tasks  $t_u$  and  $t_r$  in  $CurSol$ 
14:    if  $Cost(CurSol) < Cost(BestSol)$  then
15:       $BestSol \leftarrow CurSol$ 
16:  end while
17: return  $BestSol$ 

```

The EO algorithm is outlined in procedure 1. The initial mapping for EO is obtained by greedily placing the tasks with highest throughput to the fastest processors. The bandwidth constraints may be violated in the initial mapping and will be handled during the optimization process. After the definition of an initial solution, the execution is continued until no further improvement is observed during a certain number of iterations.

To evaluate a configuration, two functions are used. $Cost()$ returns the cost of a configuration, calculated as the total system power according to equation (1). $CapP()$ calculates the penalty for link and memory bandwidth violations:

$$CapP = \sum_{l \in \mathcal{L}} \max\left(\frac{B_l - Cap}{Cap}, 0\right) + \sum_{mc \in \mathcal{MC}} \max\left(\frac{B_{mc} - McBw}{McBw}, 0\right),$$

where B_l is the total bandwidth of flows routed through link l and B_{mc} is the bandwidth of controller mc . If all constraints are satisfied, then $CapP = 0$.

The core of the algorithm focuses on selecting the pair of tasks that must be swapped. The fitness functions alternate depending on the iteration number. In one case, fitness is oriented to improve the power generated by inter-task and memory communication, considering the hop-counts and bandwidth parameters. In the second case the power generated by computations is addressed.

The first task, t_u , is selected by using the Φ_u fitness function and sorting the tasks according to the fitness value. The second task, t_r , is selected by ranking the task according to the improvement in cost that the swap would produce (Φ_r functions). The power law described by equation (2) is used to select the tasks randomly. Finally the locations of tasks of t_u and t_r are swapped unconditionally and $BestSol$ is updated if the cost is better than any solution visited so far.

4.1 Fitness Functions

To model the fitness for the power consumption generated by the inter-task and memory traffic on the mesh, Φ_u^{comm} ranks the tasks according to the product of total traffic and the square of hop-count of the involved flows:

$$\Phi_u^{comm}(t_i) = - \sum_{f_{sd}: (t_s=t_i) \vee (t_d=t_i)} B_{sd} \cdot (h_{sd}^x + h_{sd}^y)^2 - \Lambda(t_i) \cdot \text{McDist}(p_j)^2.$$

The square of hop-count penalizes tasks with longer flows, rather than those with high bandwidth, since B_{sd} and $\Lambda(t_i)$ are constants that cannot be changed. The selection of the ranked tasks tends to pick tasks with high communication cost. The negative sign allows to rank the tasks in ascending order of fitness.

The fitness function for the replacement task t_r aims at selecting a task that, when swapped with t_u , would mostly decrease the communication cost and contribute to reduce the violations of maximum bandwidth:

$$\Phi_r^{comm}(t_i) = \text{Cost}(\text{NewSol}) \cdot (1 + \text{CapP}(\text{NewSol})),$$

where *NewSol* is the solution obtained by swapping t_i and t_u .

The computation-oriented fitness functions aim at finding power-efficient solutions by smoothing the *voltage spillover* in the voltage islands. Let us call V_i^{\min} the minimum voltage required to guarantee the throughput of task t_i assigned to a processor in some voltage island ι_n . Since task is living in the same island with other tasks, it may not be possible to assign V_i^{\min} to it, as other tasks may require a higher voltage.

We define the *voltage spillover* of t_i as $\text{Spillover}_i = V_i^{\min} - \bar{V}$, where \bar{V} is the average minimal voltage of all tasks allocated in the same voltage island. The dispersion of island ι_n is defined as

$$\text{Dispersion}_n = \sum_{t_i \in \iota_n} (\text{Spillover}_i)^2.$$

and measures the voltage imbalance for the island. High dispersions imply less power-efficient solutions, as more processors operate at voltages higher than required. Computational fitnesses aim at decreasing the voltage dispersion of the system. The unfavorable component is selected from the tasks with the high spillover value:

$$\Phi_u^{comp}(t_i) = -\text{Spillover}_i.$$

The replacement task is selected to maximize the product of the cost improvement with the dispersion, penalizing solutions with large capacity violations:

$$\Phi_r^{comp}(t_i) = \frac{1 + \text{CapP}(\text{NewSol})}{\Delta \text{Cost} \cdot \Delta \text{Dispersion}}.$$

5 Experimental Results

This section considers optimal mappings obtained by MILP for small examples and compares the quality and speed of the SA and EO heuristics. The latter is shown to outperform in both metrics for a vast space of configurations.

Every test case is characterized by an application graph and a target CMP, as described by Table 1. The number of tasks and flows are reported in the second and third columns. The fourth and fifth columns show the mesh dimensions and the number of memory controllers. First group of examples is inspired by the applications, widely used in the SoC research (e.g. [10]): *Multi-Window Displayer (MWD)*, *MPEG4 decoder (MPEG4)* and *Object Plane Decoder (OPD)*. To explore the EO scalability, we generate large examples for mapping onto 8×8 to 20×20 -tile CMPs (test cases *64T-400T*). The graphs for these configurations are obtained by combining instances of *MWD*, *MPEG4* and *OPD*. To avoid disconnected clusters of tasks, few random flows were added between the cores.

Table 1: Testcase configurations.

Name	# of tasks	# of flows	Grid size	# of MC
<i>MWD</i>	12	11	4×3	2
<i>MPEG4</i>	12	13	4×3	2
<i>OPD</i>	16	17	4×4	2
<i>64T</i>	64	90	8×8	4
<i>144T</i>	144	200	12×12	4
<i>256T</i>	256	380	16×16	8
<i>400T</i>	400	595	20×20	8

We consider three processor classes with different frequency and power parameters operating at three voltage levels: 1.2V, 1.0V and 0.8V. The parameters are reported in Table 2. The distribution of tiles in the CMP is as follows: 20% of the tiles have C1 processors, 30% - C2 processors and 50% - C3 processors. Without loss of generality, we assume that all islands have the same size S_{vi} .

Table 2: Parameters of the processor classes.

Class	1.2V	1.0V	0.8V
C1	1000MHz, 260mW	800MHz, 150mW	600MHz, 70mW
C2	450MHz, 200mW	350MHz, 120mW	250MHz, 60mW
C3	160MHz, 55mW	130MHz, 30mW	100MHz, 15mW

Every task has a throughput requirement (IPS) and a different performance when executed at each class of processor (IPC). These values are defined randomly, with IPC values in the interval $[0.5, 2.0]$. This randomization contributes to have an unbiased tuning of the metaheuristics. The traffic $\lambda(t_i)$ between the task t_i and memory controller is estimated as 20% of total traffic between t_i and other tasks. The ratio between the request and reply traffic is set to $\rho = 0.2$.

SA is parametrized by two values: the initial temperature T_{init} and the cooling factor α . We define $T_{init} = 10^4$ and only vary the α value. Given that a large range of α is explored for each experiment, the solution quality is not dependent on T_{init} . The only parameter of EO is τ , the exponent of the power-law.

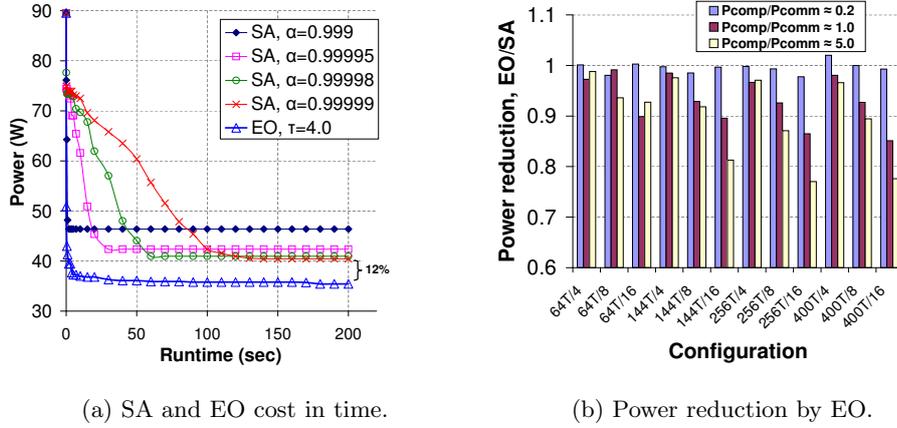


Fig. 3: Power optimization by metaheuristics.

5.1 Simulated Annealing and Extremal Optimization: Comparison

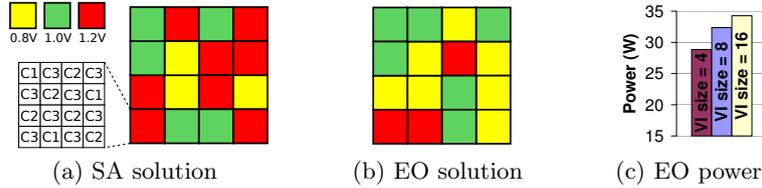
MILP formulation allows obtaining optimal solutions for the mapping problem, however, it is computationally affordable only for small examples. Table 3 displays the times required to solve the MILP problem by CPLEX [20] and to find optimal solutions by SA and EO, for the examples of the first group. This comparison affirms the fact, that both metaheuristics perform very well for the small examples with known optimum.

Table 3: Time to reach the optimal solution (sec).

Name	MILP	SA	EO
<i>MWD</i>	85.25	0.01	0.01
<i>MPEG4</i>	120.17	0.02	0.01
<i>OPD</i>	4594.40	1.17	0.08

Further we compare the SA and EO metaheuristics for the task mapping problem, using the *256T* example with $S_{vi} = 16$. The timeout for execution is set to 200 seconds, as no significant improvements are observed after that time for both methods. Figure 3(a) depicts the evolution of the cost function value for SA with various α and EO with $\tau = 4.0$. SA traces corresponding to higher values of α drop slower, but achieve better solutions in the long run. The resulting cost discovered by EO upon timeout outperforms any of the SA solutions by 12%. Another important fact is that EO solution cost drops rapidly to the value obtained in the long run. This makes EO useful when fast estimation of the cost is required, e.g. in exploration loops.

In this work we perform multiple SA runs with different α values and select the best results. The aim is to show that EO is a better alternative even with a very good tuning of SA. It was also observed that EO is much less sensitive to τ and to the size of the problem. Hence, we always define $\tau = 4.0$, the value found to deliver good results for all test cases.

Fig. 4: Voltage distribution and power for $256T$ example.

5.2 Power Optimization with EO

The goal of this section is to study the reduction in power that EO delivers in comparison with SA upon the timeout of 200 seconds. Examples of different size are considered ($64T$ to $400T$) with voltage islands varied among 4, 8 and 16 processors. Another important parameter is the ratio between the computation and communication power, as it reflects the ability of the approach to give priority to one power component or improve both simultaneously. Inspired by the results in [13], three values for P_{comp}/P_{comm} are explored: 0.2, 1.0 and 5.0.

Figure 3(b) reports the power (equation (1)) of the EO solution with respect to the best value obtained by SA. For each configuration, denoted as $testcase/S_{vi}$ along the X-axis, three values for different P_{comp}/P_{comm} are shown. For the majority of configurations EO outperformed the results of SA, with a maximum gain in power of 22.5%. Only for 3 of 36 explored configurations EO was slightly worse than SA. The difference in this case did not exceed 2.0%.

EO tends to perform better than SA at higher P_{comp} as well as for larger values of S_{vi} . In other words, EO better minimizes the voltage of islands, due to the consideration of *voltage spillover*. As the island size grows, the amount of tasks, required to be swapped in order to improve the voltage, also increases. This is one of the important features of EO, since it can model the fitness of each component in the system. Modeling the voltage spillover in SA is difficult, since only a global cost is considered in the acceptance of moves.

As an example, Fig. 4 shows the voltage distributions for the $256T/16$ test case. Every island contains a mixture of C1, C2 and C3 processors, as shown in Fig. 4(a). The final voltage assignment is represented by the three colors in the figure. The SA solution has 8 islands at 1.2V, 5 at 1.0V and 3 at 0.8V. The EO solution has 3 islands at 1.2V, 6 at 1.0V and 7 at 0.8V. The estimated power consumption of the EO solution is 12% lower. Another intuitive result is that the total power grows with the size of voltage islands (Fig 4(c)), since larger islands imply less mapping flexibility for individual tiles to reduce voltage.

6 Conclusions

Design reuse becomes a major paradigm for engineering many-core systems. This paper has addressed the problem of static task mapping for tiled CMPs with multiple voltage islands, as an approach to reduce design cost and time-to-market. The problem formulation considers task throughput requirements, on-chip and off-chip memory traffic, and bandwidth constraints. Extremal optimization has shown to be an efficient method for solving this combinatorial problem.

7 Acknowledgement

This research has been funded by project CICYT TIN2007-66523, FPI grant BES-2008-004612, and grants from Intel Corporation and Catalan Government (SGR 2009-1137).

References

1. Pham, D. et al.: Overview of the architecture, circuit design, and physical implementation of a first-generation cell processor. *J. Solid-State Circuits* **41** (2006)
2. Bell, S. et al.: Tile64 - processor: A 64-core SoC with mesh interconnect. In: *Solid-State Circuits Conference*. (2008)
3. Vangal, S. et al.: An 80-tile 1.28Tflops network-on-chip in 65nm CMOS. In: *Solid-State Circuits Conference*. (2007)
4. Marculescu, R., Ogras, U.Y., Peh, L.S., Jerger, N.E., Hoskote, Y.: Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives. *IEEE Trans. on Computer-Aided Design of Integrated Circuits* **28** (2009)
5. Azimi, M. et al.: Integration Challenges and Tradeoffs for Tera-scale Architectures. *Intel Technology Journal* (2007)
6. Balakrishnan, S., Rajwar, R., Upton, M., Lai, K.: The impact of performance asymmetry in emerging multicore architectures. In: *International Symposium on Computer Architecture*. (2005)
7. Lackey, D. et al.: Managing power and performance for System-on-Chip designs using voltage islands. In: *Int. Conf. Computer-Aided Design*. (2002)
8. Kim, W., Gupta, M., Wei, G.Y., Brooks, D.: System level analysis of fast, per-core DVFS using on-chip switching regulators. In: *International Symposium on High Performance Computer Architecture*. (2008)
9. Mak, W.K., Chen, J.W.: Voltage island generation under performance requirement for SoC designs. In: *Asia and South Pacific Design Automation Conference*. (2007)
10. Ghosh, P., Sen, A.: Energy efficient mapping and voltage islanding for regular NoC under design constraints. *J. High Perform. Syst. Archit.* **2** (2010)
11. Xu, R., Melhem, R., Mosse, D.: Energy-aware scheduling for streaming applications on chip multiprocessors. In: *Int. Symp. Real-Time Systems*. (2007)
12. Hung, W.-L. et al.: Temperature-aware voltage islands architecting in System-on-Chip design. In: *Int. Conf. Computer Design*. (2005)
13. Varatkar, G., Marculescu, R.: Communication-aware task scheduling and voltage selection for total systems energy minimization. In: *Int. Conf. Computer-Aided Design*. (2003)
14. Chen, G., Li, F., Son, S., Kandemir, M.: Application mapping for chip multiprocessors. In: *Design Automation Conference*. (2008)
15. Dally, W., Towles, B.: *Principles and Practices of Interconnection Networks*. (2003)
16. Boettcher, S., Percus, A.G.: Extremal optimization: Methods derived from co-evolution. In: *Genetic and Evolutionary Computation Conf.* (1999)
17. Nikitin, N., Cortadella, J.: Static task mapping for tiled chip multiprocessors with multiple voltage islands. Technical Report. <http://www.lsi.upc.edu/~techreps/files/R11-13.zip> (2011)
18. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. *Science* **220** (1983)
19. De Falco, I., Della Cioppa, A., Maisto, D., Scafuri, U., Tarantino, E.: A multiobjective extremal optimization algorithm for efficient mapping in grids. **58** (2009)
20. CPLEX. (<http://www.ilog.com/products/cplex>)