Static Task Mapping for Tiled Chip Multiprocessors with Multiple Voltage Islands

Nikita Nikitin
Universitat Politècnica de Catalunya
Barcelona, Spain

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. Besides proposing a linear programming model for small systems, 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.**

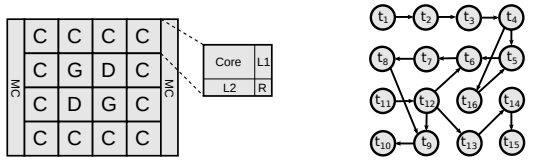
I. Introduction

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]–[5]. 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) [6]. 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*.

A. Tiled CMP architecture

To reduce the design time, *tile replication* was shown to be an efficient reuse methodology for many-core CMPs [4], [5]. This led to the concept of *tiled architecture*, characterized by regular structures of homogeneous tiles, each one consisting of a processing core, a cache memory and a router. Further research in this area inspired designs with *heterogeneous tiles*, preserving the regularity of the structure, but introducing several classes of tiles [1], [7], [8]. Such systems may include some specialized processors (e.g., graphics, DSP) or different implementations of the same architecture (e.g., in-order/out-of-order, multi-threading) with varied power-performance trade-offs. Figure 1(a) depicts a tiled 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.



(a) (b)

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

B. Power management

CMPs are designed to operate under a certain power budget that assures the performance and thermal properties of the system. 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 [9].

Unfortunately, voltage islands have a high design cost. Firstly, the floorplanning of the system is constrained by the design of the power delivery network and the location of the level shifters. Secondly, and more important, power management requires different voltage regulators for each power supply. Off-chip regulators need extra area on the PCB that may be unacceptable if the system has a large amount of power domains. On-chip regulators involve a significant area overhead and power consumption due to the large inductances and switching capacitors required to provide a stable supply voltage [10].

It is therefore realistic to consider that future CMPs will have many cores (hundreds) 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 to take advantage of this flexibility. Hence, it is convenient to allocate tasks in a way that cores within the same voltage islands can share similar voltage/frequency parameters.

Up to now, the research on power-aware mapping has assumed that the voltage islands are defined pre-silicon during task mapping in application-specific SoCs, often disregarding the cost of implementing the voltage islands. A broad overview of the related work on SoC application mapping and island planning can be found in [6]. The approach in [11] considers performance constraints, but does not account for the communication component of power. A more realistic approach is proposed in [12] in which computation and communication are both optimized taking into account a third component related to voltage shifters. Thermal-aware island partitioning via evolutionary algorithms was proposed in [13]. The distinction of different processor classes was introduced in [14], but assuming that every processor can run at an independent voltage level.

C. 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. Therefore, the voltage islands have been already floorplanned and the maximum bandwidth of the links between cores 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, as well as the limited bandwidth of the memory controllers (MCs). Finally, the methods proposed for task mapping must be scalable and suitable to handle systems with hundreds of cores. Hence, scalability becomes a major concern of this work.

The work in [15] proposed a framework for accurate compiler-level mapping of applications onto homogeneous mesh CMPs through detailed 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. It also demonstrates better scalability, due to the higher-level abstraction of application with a task graph.

The examined problem consists of *statically mapping* a set of parallel tasks onto a many-core CMP and selecting the voltages of the CMP islands so that the total communication and computation power is minimized. The application to be mapped is represented as a graph of parallel tasks (Fig. 1(b)) with specified average communication requirements between the tasks, that is a common assumption for mapping onto the on-chip systems [6]. The partitioning of application into parallel tasks can be obtained by profiling [15].

Every task has an associated throughput constraint (instructions per second) 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 that can be selected to find the best performance/power trade-off for each task. However, this flexibility is constrained by an important limitation: all the cores in the same island must work with the same voltage.

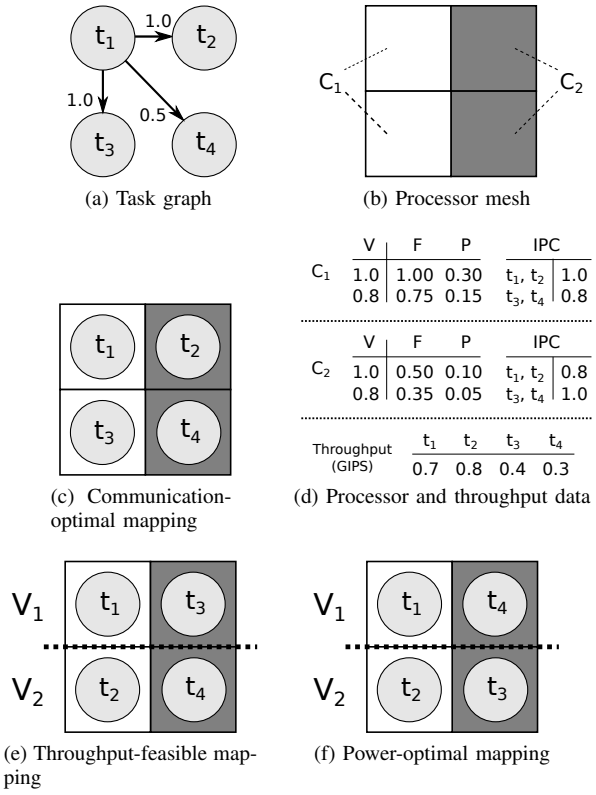
This work will also assume that the cores are organized in a mesh with XY-routing [16]. The task mapping must satisfy the link and memory controller bandwidth constraints defined a priori to avoid a saturation of the communication fabrics.

The main contributions of this work can be summarized as follows:

- Specification of the problem for power-aware task mapping onto manufactured CMP with several tile classes, subject to throughput constraints.
- Mathematical formulation of the problem as a mixed-integer linear programming problem (MILP) capable of delivering optimal solutions for examples of small size.
- Scalable approach based on *Extremal Optimization* (EO) [17], shown to outperform the optimization by simulated annealing, both in quality of results and computational cost. The scalability of the method is proved by examples with hundreds of cores.

We would like to emphasize the fact that Extremal Optimization is a combinatorial optimization technique mostly unknown in the EDA community. The results obtained for this problem have been surprisingly good and very competitive with regard to Simulated Annealing. We believe these results may encourage further research in other areas related to layout synthesis. A related work, although in a different context, can be found in [18].

The structure of the paper is as follows. Next section presents an overview of the mapping problem by considering a small example. Section III proposes an MILP formulation of the problem. The metaheustistic techniques are explained in Section IV. Section V discusses the experimental results.



(a) Task graph (b) Processor mesh

(c) Communication-optimal mapping (d) Processor and throughput data

(e) Throughput-feasible mapping (f) Power-optimal mapping

Fig. 2: Task mapping example.

II. Problem overview

This section discusses 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.

1) Communication-optimal mapping: Figure 2(c) shows a task mapping that optimizes the communication metric, that is the product of bandwidth and hop-count. Assuming the distance between the neighboring processors is one hop, the communication cost of this mapping is

$$CCost_1 = 1.0 \cdot 1 + 1.0 \cdot 1 + 0.5 \cdot 2 = 3.0 \ (Gbps).$$

2) Throughput-feasible mapping: Now let us take into account the processor parameters and consider the throughput requirements of the tasks. Figure 2(d) describes the processor parameters. They can operate at two voltages, 1.0 and $0.8V$. The corresponding frequency (F, in *GHz*) and power (P, in W) for each voltage is shown in the tables. Due to the nature of the tasks and the implementation of each processor, each task may be executed with a different performance (Instructions Per Cycle (IPC)) in each class of processors. Finally, each task may require a specific throughput (given in giga-*IPS* in Fig. 2(d)).

The mapping in Fig. 2(c) is infeasible with 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 $\texttt{IPC}(t_2) \cdot F(1.0V) = 0.8 \cdot 0.5 = 0.4$ *GIPS*, while the throughput requirement for t_2 is 0.8 *GIPS*.

To satisfy the requirements, tasks t_2 and t_3 are swapped (see Fig. 2(e)). This mapping satisfies the throughput constraints and still keeps the optimal value for the communication metrics.

3) 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 is

$$P_{comm} = CCost_2 \cdot E_{bit} = 3.0Gbps \cdot 0.1nJ/bit = 0.3\ W,$$

and the total power $P = P_{comp} + P_{comm} = 1.10\ W$.

Notice that if we swap tasks t_3 and t_4 (Fig. 2(f)), the upper island can lower the voltage to $0.8V$ without violating the throughput constraints. The new computation power is $P_{comp} = 0.15 + 0.05 + 0.30 + 0.10 = 0.60\ W$. The communication cost is increased: $CCost_3 = 1.0 \cdot 1 + 1.0 \cdot 2 + 0.5 \cdot 1 = 3.5$ ($Gbps \cdot hop$), 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.

The previous example demonstrates the importance of the task mapping problem when trying to minimize power consumption in a CMP with multiple classes of processors and voltage islands. The next section shows how optimal solutions for small instances of the problem can be found based on an MILP formulation.

III. A MATHEMATICAL MODEL

This section gives a formal definition of the problem via a Mixed-Integer Linear Programming model. This model will be later used as the basis of a heuristic method for large-scale systems based on Extremal Optimization.

A. Parameters of the problem

The parameters of the problem are summarized in Table I. The variables of the MILP formulation are outlined in Table II.

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 flow $f_{sd} \in \mathcal{F}$ that defines the communication from task t_s to t_d. Every flow has a minimum required bandwidth B_{sd}. Every task t_i has a throughput constraint $\texttt{IPS}(t_i)$, 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 $\Lambda(t_i)$ 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 dimensions of $W \cdot H$ cells, where $\mathbb{P}$ is the set of processors and $\mathcal{L}$ is the set of communication links. Links are organized into an on-chip network with regular mesh topology [19]. The communication capacity between the neighboring cells is determined by the global parameter `Cap` (all links are assumed to have the same capacity). Every cell represents a processor p_j, belonging to one of the processor classes in $\mathbb{C} = \{c_1, .., c_C\}$. Different classes of processors have distinct performance executing each task. The performance of p_j to execute task t_i, measured in instructions per cycle, is specified by the function $\texttt{IPC}(t_i, p_j)$.

Task parameters	
TG	Task graph with tasks t_i and flows f_{sd}
B_{sd}	Bandwidth requirement for flow f_{sd}
$\texttt{IPS}(t_i)$	Throughput requirement for task t_i (instr./sec.)
$\Lambda(t_i)$	Traffic of task t_i to the memory controller
ρ	Ratio of traffic rates to and from controller
Processor grid parameters	
$PM(\mathbb{P}, \mathcal{L})$	Mesh of processors ($\mathbb{P}$) with communication links ($\mathcal{L}$)
`Cap`	Maximum capacity of the communication links
$\texttt{IPC}(t_i, p_j)$	Performance of p_j executing t_i (instr./cycle)
$\mathcal{V}$	Set of available operating voltages v_k
$F(p_j, v_k)$	Frequency of processor p_j at voltage v_k
$P(p_j, v_k)$	Power of processor p_j at voltage v_k
$\texttt{MC}(p_j)$	Memory controller associated with p_j
$\texttt{McDist}(p_j)$	Distance from p_j to associated controller
`McBw`	Maximum bandwidth of memory controllers
Voltage island parameters	
$\{\iota_n\}$	Set of voltage islands
$\mathcal{I}$	Map from processors to voltage islands

TABLE I: Input parameters of the problem.

Variable	Type	Description
a_{ijk}	Binary	task t_i is assigned to processor p_j with voltage v_k
v_k^n		voltage island ι_n operates at voltage v_k
r_{sd}^l		link l belongs to the route of flow f_{sd}
m_{sd}^l	Real	mapping indicator for the terminals of f_{sd}
h_{sd}^x, h_{sd}^y		hop-count (x and y) of route f_{sd}

TABLE II: Variables of the MILP formulation.

The processors may operate at different voltages. We assume a set of voltages $\mathcal{V} = \{v_1, .., v_V\}$ 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 $\mathcal{I}$. The voltage of an island can be adjusted independently of the other islands, however, all processors in an island must operate at the same voltage.

A CMP has a set of controllers to access the off-chip memory. Guided by the existing implementations [4], [5], in this work we assume controller placement at the periphery of processor mesh. However, this does not limit the proposed approach from having the controllers placed inside the mesh, that was demonstrated beneficial by the recent research [20]. Another assumption we make is that every processor p_j is associated with *one* controller, as defined by the function $\texttt{MC}(p_j)$. This assumption can be eliminated by specifying the probabilities of accessing different controllers for p_j. Function $\texttt{McDist}(p_j)$ returns the hop-count distance from p_j to the related controller. The `McBw` parameter sets the maximum controller bandwidth to guarantee performance of memory access.

B. Cost function

The goal of the model is to minimize power consumption under a set of design and performance constraints.

The binary variables a_{ijk} define whether task t_i is mapped onto processor p_j operating at voltage v_k. The total power consumption for computation can be defined as follows:

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

The power consumption for communication has two terms: the on-chip communication, defined by the flows between the tasks and

the off-chip communication, defined by the traffic to the memory controllers. To model the first term, we introduce the variables h^x_{sd} and h^y_{sd} that represent the hop-count of flow f_{sd} in the x- and y-directions, respectively. The power consumption for inter-task communication can be defined as

$$P^t_{comm} = \sum_{f_{sd} \in \mathcal{F}} B_{sd} \cdot (h^x_{sd} + h^y_{sd}) \cdot E_{bit},$$

and the term related to communication with memory controllers

$$P^{mc}_{comm} = \sum_{t_i \in \mathcal{T}} \sum_{p_j \in \mathbb{P}} \sum_{v_k \in \mathcal{V}} a_{ijk} \cdot \Lambda(t_i) \cdot \texttt{McDist}(p_j) \cdot E_{bit}.$$

where E_{bit} is the estimated energy for transmitting one bit over a link. The objective of the problem is to minimize the total power:

$$\min: \quad P = P_{comp} + P_{comm} = P_{comp} + P^t_{comm} + P^{mc}_{comm}. \tag{1}$$

C. Constraints

The first two constraints are the classical requirements for an assignment problem. Every task t_i has to be assigned to some processor p_j and every processor can hold one task at most:

$$\forall t_i \in \mathcal{T}: \quad \sum_{p_j \in \mathbb{P}} \sum_{v_k \in \mathcal{V}} a_{ijk} = 1. \tag{2}$$

$$\forall p_j \in \mathbb{P}: \quad \sum_{t_i \in \mathcal{T}} \sum_{v_k \in \mathcal{V}} a_{ijk} \leq 1. \tag{3}$$

The next step is to model the communication component of the power. A set of constraints is introduced to calculate the hop-count of each flow assuming an XY-routing. Each processor p_j is located in a tile at column x_j and row y_j of the mesh (Fig. 3a). The coordinates are uniquely defined by the index j: $x_j = j \bmod W$ and $y_j = \lfloor j/W \rfloor$, where W is the width of the mesh. For any task t_i, we define (x_i, y_i) as the location of the processor assigned to the task. Then, the location is specified by the expressions over the task assignment variables:

$$\begin{aligned} x_i &= \sum_{p_j \in \mathbb{P}} (j \bmod W) \sum_{v_k \in \mathcal{V}} a_{ijk} \\ y_i &= \sum_{p_j \in \mathbb{P}} (\lfloor j/W \rfloor) \sum_{v_k \in \mathcal{V}} a_{ijk}. \end{aligned} \tag{4}$$

For every flow f_{sd}, the source and destination tasks, t_s and t_d, are mapped onto processors p_s and p_d, with coordinates (x_s, y_s) and (x_d, y_d), respectively, defined by (4). The horizontal hop-count, $h^x_{sd} = |x_s - x_d|$, and the vertical hop-count, $h^y_{sd} = |y_s - y_d|$ are defined by the following constraints[1]:

$$\begin{aligned} x_s - x_d \leq h^x_{sd}, \qquad & x_d - x_s \leq h^x_{sd} \\ y_s - y_d \leq h^y_{sd}, \qquad & y_d - y_s \leq h^y_{sd}. \end{aligned} \tag{5}$$

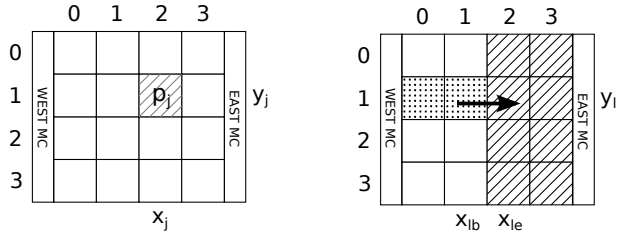
The next group of constraints defines the relations between voltage islands and throughput. Let the binary variable v^n_k represent the fact that the voltage island ι_n operates at voltage v_k. First, for every island ι_n only one voltage has to be selected:

$$\forall \iota_n \in \mathcal{I}: \quad \sum_{v_k \in \mathcal{V}} v^n_k = 1. \tag{6}$$

To enforce that all processors in the same voltage island work with the same voltage, the following constraint is added:

$$\forall \iota_n \in \mathcal{I},\ \forall v_k \in \mathcal{V}: \quad \sum_{t_i \in \mathcal{T}} \sum_{p_j \in \iota_n} a_{ijk} \leq \texttt{Num}(a_{ijk}) \cdot v^n_k, \tag{7}$$

[1]the pair of inequalities and the fact that the h variables are implicitly minimized with the cost function (since this implies minimization of power), guarantee the equality with the absolute value.



(a) Processor p_j located in the cell (x_j, y_j) of the mesh.

(b) East link from cell (x_{lb}, y_l) to cell (x_{le}, y_l).

Fig. 3: Definition of the processor and link location in mesh.

where $\texttt{Num}(a_{ijk})$ is the number of a_{ijk} variables in the LHS of the inequality. Expression (7) in combination with (6) guarantees that only the assignment variables for the selected voltage may take non-zero values.

The throughput constraint should guarantee that for each task t_i executed on processor p_j the product of $\texttt{IPC}(t_i, p_j)$ and the processor frequency $F(p_j, v_k)$ defined by the current voltage, is not less than the required throughput $\texttt{IPS}(t_i)$. Hence, the following relation is specified for each $t_i \in \mathcal{T}$:

$$\sum_{p_j \in \mathbb{P}} \sum_{v_k \in \mathcal{V}} a_{ijk} \cdot \texttt{IPC}(t_i, p_j) \cdot F(p_j, v_k) \geq \texttt{IPS}(t_i). \tag{8}$$

The last group of constraints aims at satisfying the requirements for link capacity and memory controller bandwidth, under the assumption of XY-routing. We start by considering the link capacity. The total link bandwidth can be expressed as the sum of the bandwidths of all flows that pass through the link. There are two terms that contribute to link bandwidth, related to the inter-task and memory controller traffic, hence the constraint can be written as

$$\forall l \in \mathcal{L}: \quad \texttt{TaskTerm}(l) + \texttt{McTerm}(l) \leq \texttt{Cap}. \tag{9}$$

Let us consider the task term first. In XY-routing, the data is always sent in the X-direction first and the Y-direction afterward. Hence, the route of a flow will pass through a link, only in case the source and destination tasks are mapped to a specific subset of processor locations. Thus, for every link l and flow f_{sd} we define the binary properties, $\texttt{MapSrc}(l, f_{sd})$ and $\texttt{MapDst}(l, f_{sd})$, that indicate whether the source and destination tasks are mapped onto the locations that imply link l to be on the flow route.

To guarantee that l is on the route of f_{sd}, both properties should be asserted, i.e., $\texttt{MapSrc}(l, f_{sd}) \cdot \texttt{MapDst}(l, f_{sd}) = 1$. This is a non-linear constraint that we linearize by introducing the real variables m^l_{sd}:

$$\texttt{MapSrc}(l, f_{sd}) + \texttt{MapDst}(l, f_{sd}) = m^l_{sd}. \tag{10}$$

Since the mapping properties can only take binary values, the m^l_{sd} variable can only take three values: 0, 1, or 2. We use another scaling of m^l_{sd} to the binary variables r^l_{sd}, that take non-zero values only when m^l_{sd} = 2, i.e. both mapping properties are true:

$$r^l_{sd} \geq m^l_{sd} - 1, \qquad 2 \cdot r^l_{sd} \leq m^l_{sd}. \tag{11}$$

The variables r^l_{sd} are equal to 1 if the route of flow f_{sd} goes through link l. Now the task term for link l is written as:

$$\texttt{TaskTerm}(l) = \sum_{f_{sd} \in \mathcal{F}} B_{sd} \cdot r^l_{sd}. \tag{12}$$

Next we explain how to represent the mapping properties $\texttt{MapSrc}(l, f_{sd})$ and $\texttt{MapDst}(l, f_{sd})$. Consider the horizontal east link of a cell with coordinates (x_{lb}, y_l) to a cell (x_{le}, y_l) (Fig. 3b).

The XY-route of flow f_{sd} can only pass through the link in case the source task t_s is mapped onto one of the two dotted processor cells. Indeed, the processor p_s should be located on the same row and in a column that is lower than or equal to the origin of the link: $(x_s \leq x_{lb}) \wedge (y_s = y_l)$. Hence, the source mapping property for an east link is:

$$\texttt{MapSrc}(l, f_{sd}) = \sum_{p_j:(x_j \leq x_{lb}) \wedge (y_j = y_l)} \sum_{v_k \in \mathcal{V}} a_{sjk}. \quad (13)$$

For the destination task t_d the requirement is to be located in a column that is greater than or equal to the link endpoint x_{le} (striped cells). The destination mapping property becomes:

$$\texttt{MapDst}(l, f_{sd}) = \sum_{p_j:(x_j \geq x_{le})} \sum_{v_k \in \mathcal{V}} a_{djk}. \quad (14)$$

The mapping properties for south, west and north links are derived in a similar manner.

Now consider the term related to the memory controller traffic. For link l let us denote $\texttt{Req}(l)$ the set of processors that send *requests* to their controllers through link l. Similarly, $\texttt{Rep}(l)$ is the set of processors that receive *replies* from controller through l. Hence, traffic to and from controllers through l is defined by the rate of tasks mapped to the sets $\texttt{Req}(l)$ and $\texttt{Rep}(l)$:

$$\texttt{McTerm}(l) = \sum_{t_i \in \mathcal{T}} \sum_{p_j \in Req(l)} \sum_{v_k \in \mathcal{V}} a_{ijk} \cdot \rho \cdot \Lambda(t_i) + \sum_{t_i \in \mathcal{T}} \sum_{p_j \in Rep(l)} \sum_{v_k \in \mathcal{V}} a_{ijk} \cdot (1 - \rho) \cdot \Lambda(t_i). \quad (15)$$

The sets $\texttt{Req}(l)$ and $\texttt{Rep}(l)$ can be expressed similarly to the MapSrc and MapDst properties for links. As an example, let us consider the same east link (Fig. 3b). Assuming XY-routing, p_j sends requests in the direction of the associated controller. Hence, the set of processors sending requests through the link is limited by those, associated with the *EAST* controller and located on the same row in the column that is lower or equal to the link origin:

$$\texttt{Req}(l) = \{p_j : (\texttt{MC}(p_j) = \textit{EAST}) \wedge (x_j \leq x_{lb}) \wedge (y_j = y_l)\}.$$

The sets for other links are derived similarly. The inequalities (9) together with the scaling relations (10), (11) and definitions (12)-(15) guarantee that the link capacity constraints are met.

The last step is to specify the bandwidth constraints of the memory controllers. The bandwidth of controller $mc_\kappa \in \texttt{MC}$ is defined by the rates of tasks mapped onto processors, associated with mc_κ:

$$\forall mc_\kappa \in \texttt{MC}: \sum_{t_i \in \mathcal{T}} \sum_{p_j:\texttt{MC}(p_j) = \kappa} \sum_{v_k \in \mathcal{V}} a_{ijk} \cdot \Lambda(t_i) \leq \texttt{McBw}. \quad (16)$$

D. Problem formulation

The problem can now be formulated as follows.

***Minimize*:** power consumption (1)

***subject to*:**

assignment constraints and hop-count definition (2)-(5),
voltage selection constraints (6), (7),
throughput constraints (8),
link capacity constraints (9)-(15)
and *memory bandwidth* constraints (16).

IV. Simulated Annealing and Extremal Optimization

This section discusses two metaheuristics commonly used to solve complex combinatorial problems: *Simulated Annealing* (SA) [21] and *Extremal Optimization* (EO) [17]. Both methods are inspired by equilibrium statistical physics. SA has been successfully applied in many EDA problems, mostly related to layout synthesis. However, EO has emerged as a very competitive alternative that can give superior results in quality and computational cost. This section shows how EO can be customized to effectively solve the task mapping problem. The results prove the superiority with regard to SA.

Both metaheuristics start from an initial mapping obtained by greedily placing the tasks with highest throughput to the fastest processors. It is assumed that the system is not highly throughput-constrained and that a feasible initial assignment can be achieved by a greedy heuristic. The bandwidth constraints may be violated in the initial mapping and will be handled during the optimization process.

A. Simulated annealing

The general algorithm for SA is described in procedure 1. To evaluate every 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 capacity and memory bandwidth violations:

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

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 $\textit{CapP} = 0$.

The SA algorithm implements a conventional annealing schedule. Given the initial temperature T_{init} and the cooling factor α, a new solution (*NewSol*) is generated (line 4) and may be accepted probabilistically, depending on the current temperature T_{cur} (line 5). The value of T_{cur} decreases as the system evolves in time (line 8). For every temperature, a number of moves that depends on the size of the system (P, that is the number of cells) is generated.

$NewSol$ is obtained by swapping a pair of random tasks and is accepted probabilistically according to Procedure 2. To penalize capacity violations, the cost is calculated as shown in lines 1-2, where $\lambda = T_{init}/T_{cur}$ is the weight for penalization, that grows in time. This decreases the probability to accept infeasible solutions as the simulation advances.

Solutions with better cost are always accepted (line 3), whereas worse-cost solutions are accepted with probability P_a:

$$P_a = \left(1 - \frac{|\Delta \textit{Cost}|}{\textit{CurCost}}\right) \cdot e^{-2 \cdot \frac{T_{init}}{T_{cur}}},$$

with $\Delta \textit{Cost} = \textit{NewCost} - \textit{CurCost}$. This probability depends on two factors. The former avoids the acceptance of solutions with high cost degradation. The latter reduces the probability of hill climbing as the temperature cools down.

B. Extremal optimization

Extremal optimization, inspired by the principle of evolution in ecosystems, is metaheuristic for complex combinatorial problems. Ecosystems 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.

Procedure 1 SIMULATEDANNEALING

$T_{cur} = T_{init}$
while improvement in last k iterations **do**
  for P iterations **do**
    generate *NewSol*
    if ACCEPT(*CurSol*, *NewSol*) **then** *CurSol*← *NewSol*
    if Cost(*NewSol*) < Cost(*BestSol*) **then** *BestSol*← *NewSol*
  end for
  $T_{cur} = \alpha \cdot T_{cur}$
end while
return *BestSol*

Procedure 2 ACCEPT(*CurSol*, *NewSol*)

CurCost← Cost(*CurSol*) + λ CapP(*CurSol*)
NewCost← Cost(*NewSol*) + λ CapP(*NewSol*)
if *NewCost* < *CurCost* **then return** true
else return true with probability P_a

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 (the worst components have lower indices in the rank). The components 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:

$$i = \lceil N \cdot p^{\tau} \rceil \tag{17}$$

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. In our experiments, we used values of τ in the interval $[3, 4]$.

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). Unlike SA, EO uses information about the system cost when selecting the swapped tasks. This results into a faster progress towards the final solution. In addition, EO accepts new solutions unconditionally without depending on any temperature cooling schedule, thus making the algorithm easier to tune.

The mapping problem is a multiobjective optimization problem, since the P_{comp}, P^t_{comm} and P^{mc}_{comm} terms of the cost function (1) depend on weakly related voltage level and hop-count values. It was observed in [18] (and proved by our experiments) 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 three power components and alternate them at different iterations of the algorithm.

The EO algorithm is outlined in procedure 3. After the definition of an initial solution (greedily), the execution is continued until no further improvement is observed during a certain number of iterations.

The algorithm used in this work is a variation of EO called *Continuous Extremal Optimization* [22]. This variant combines EO with a local search at the beginning of each iteration, that is performed by sequentially swapping P random pairs of tasks and accepting only those that improve the cost. This variant contributed to improve the cost of the final solution and the speed of the algorithm.

Procedure 3 EXTREMALOPTIMIZATION

CurSol← *BestSol*← "Some initial solution"
while some improvement in the last k iterations **do**
  Local search: swap P randomly selected pairs sequentially and
    accept only those that improve the cost of *CurSol*
  if (iter *mod* 3) = 1 **then** /* improve task comm. cost */
    sort all tasks in ascending order of $\Phi^{comm}_{u,t}$
    select t_u according to equation (17)
    sort all tasks $t_i \neq t_u$ in ascending order of Φ^{comm}_{r}
    select t_r according to equation (17)
  else if (iter *mod* 3) = 2 **then** /* improve mc comm. cost */
    sort all tasks in ascending order of $\Phi^{comm}_{u,mc}$
    select t_u according to equation (17)
    sort all tasks $t_i \neq t_u$ in ascending order of Φ^{comm}_{r}
    select t_r according to equation (17)
  else /* improve comp. cost */
    sort all tasks in ascending order of Φ^{comp}_{u}
    select t_u according to equation (17)
    sort all tasks $t_i \neq t_u$ in ascending order of Φ^{comp}_{r}
    select t_r according to equation (17)
  swap tasks t_u and t_r in *CurSol*
  if Cost(*CurSol*) < Cost(*BestSol*) **then**
    BestSol← *CurSol*
end while
return $BestSol$

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 consumption generated by inter-task communication, considering the hop-counts and bandwidth parameters. In the second case, the power of communication with memory controllers is optimized. The last case addresses the power generated by computations.

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 (17) 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 other solution visited so far.

C. Fitness functions

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

$$\Phi^{comm}_{u,t}(t_i) = - \sum_{f_{sd}:(t_s=t_i)\vee(t_d=t_i)} B_{sd} \cdot (h^x_{sd} + h^y_{sd})^2.$$

The square of hop-count penalizes tasks with longer flows, rather than those with high bandwidth, since B_{sd} is a constant parameter 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.

Similar fitness is used to select unfavorable task t_i mapped to processor p_j for the controller-related term of power:

$$\Phi^{comm}_{u,mc}(t_i) = -\Lambda(t_i) \cdot \texttt{McDist}(p_j)^2.$$

Although the fitness functions selected for both terms of communication power look similar, we consider them as individual candidates for multiproduct optimization. The intrinsic difference between the two types of communication is that an inter-task flow depends on mapping of both, source *and* destination tasks, while the memory controller flow depends on one, either source *or* destination task.

The fitness function for the replacement task t_r is the same for both types of communication. It 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:

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

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 $\textit{Spillover}_i = V_i^{\min} - \overline{V}$, where $\overline{V}$ is the average minimal voltage of all tasks allocated in the same voltage island. The dispersion of island ι_n is defined as

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

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:

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

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

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

V. Experimental results

The results presented in this section have three primary objectives. Firstly, optimal solutions are obtained for small examples by solving the MILP model. It is shown that metaheuristics can also find the optimum for these examples, and in much shorter time. Secondly, the quality and speed of SA and EO are compared. The latter is demonstrated to outperform in both metrics for a vast space of solutions. Thirdly, the impact of the link capacity and memory bandwidth constraints is discussed.

A. Examples and experimental setup

Every test case for the mapping problem is characterized by an application task graph and a target CMP. The parameters of the test cases are presented in Table III. The number of tasks and flows are reported in the second and third columns. The fourth column shows the dimensions of the mesh for the target CMP. The last column displays the number of memory controllers in each test case.

The first group of examples is inspired by the realistic applications, widely used in the SoC research domain (e.g. [12], [23]): *Multi-Window Displayer (MWD)*, *MPEG4 decoder (MPEG4)* and *Object Plane Decoder (OPD)*. To explore the scalability of the proposed technique, we generate a group of large examples for mapping onto 8×8, 12×12, 16×16 and 20×20-tile CMPs (test cases *64T* to *400T*). The task graphs for these configurations are obtained by combining instances of *MWD*, *MPEG4* and *OPD*. For instance, the task graph for *400T* consists of 30 small applications, 10 instances of each type. To avoid having totally disconnected clusters of tasks, few random flows were added between the components. The third column of Table III displays the resulting number of flows in graphs.

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

TABLE III: Testcase configurations.

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

TABLE IV: Parameters of the processor classes.

For the experiments, we have considered three processor classes (C1, C2 and C3) with different frequency and power parameters operating at three different voltages: 1.2V, 1.0V and 0.8V. The parameters are reported in Table IV. The distribution of tiles in the CMP is as follows: 20% of the tiles have C1-processors, 30% have C2-processors and 50% have C3-processors. The classes are distributed uniformly in such a way that all voltage islands have a similar mixture of classes. Without loss of generality, we assume that all islands have the same size S_{vi} (number of tiles). Different values have been used in the experiments.

Every task has a different throughput requirement (IPS) and a different performance when executed at each class of processor (IPC). All these values are defined randomly, with IPC values in the interval $[0.5, 2.0]$ and guaranteeing that a feasible mapping exists for the assigned performance and throughput requirements. This randomization contributes to explore a larger set of configurations and to have an unbiased tuning of the metaheuristics.

The traffic $\Lambda(t_i)$ between the task t_i and memory controller was estimated as 20% of total traffic between t_i and all other tasks. The ratio between the request and reply traffic was set to $\rho = 0.2$.

SA is parametrized by two values: the initial temperature T_{init} and the cooling factor α. In our experiments we define $T_{init} = 10^4$ and only vary the α value. Given that a large range of α values have been explored for each experiment, the quality of the solutions is not dependent on T_{init} . The only parameter of EO is τ, i.e., the exponent of the power-law (eq. (17)). Section V-C discusses the strategy used to explore the values of α and τ.

B. Comparison with the optimal solution

The MILP formulation allows obtaining optimal solutions for the mapping problem. However, the search of the optimum is computationally affordable only for small examples. We used CPLEX [24] to solve the MILP problem for the test cases of the first group: *MWD*, *MPEG4* and *OPD*. The size of voltage islands S_{vi} was set to four. The time required to find the optimum is displayed in the "MILP" column of Table V. One can observe the two-order increase in runtime for

Name	MILP	SA	EO
MWD	85.25	0.01	0.01
MPEG4	120.17	0.02	0.01
OPD	4594.40	1.17	0.08

TABLE V: Time to reach the optimal solution (sec).

a 16-tile example (*OPD*) in comparison with the 12-tile examples (*MWD*, *MPEG4*).

The metaheuristics are able to achieve the optimal solution for the same examples in much shorter time (columns "SA" and "EO" of Table V). In this experiment the SA and EO algorithms were run for a variety of parameters (α and τ), and the best runtime values were selected. This comparison affirms the fact, that both metaheuristics perform very well for the small examples with known optimum.

C. Simulated Annealing and Extremal Optimization: comparison

This section tries to give an apple-to-apple comparison of both metaheuristics for the task mapping problem. The comparison is illustrated using the *256T* example with $S_{vi} = 16$ and represents a typical behavior of the two algorithms for the explored test cases.

The timeout for execution was set to 200 seconds, since no significant improvements were observed after that time for both methods. Figure 4 depicts the evolution of the cost function value obtained by SA with various α and by EO with $\tau = 4.0$. The traces corresponding to higher values of α drop slower, but achieve better solutions in the long run.

Let us now consider the EO trace. At every moment in time the current solution found by EO is better than any of the SA solutions, obtained with different α values. 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 (0.1-2.5 seconds, depending on the test case), to the 10% of accuracy, with respect to the value obtained in the long run. This makes EO useful to apply when fast estimation of the cost is required, e.g. in exploration loops.

SA requires a careful tuning to eliminate the dependency of the α parameter on the problem size. Otherwise, small changes in α may lead to an important degradation in quality. In this work we do not aim at tuning the SA method. Rather, we perform multiple 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.

In the experiments, it was also observed that EO is much less sensitive to τ and to the size of the problem. This simplifies the tuning of the algorithm. Note that some variation of τ may provide slightly better results for certain examples. However, we do not aim at demonstrating the highest improvement for all test cases. We prefer to emphasize that even having τ *fixed*, EO is able to outperform SA with *any* α. Guided by this reasoning, in the following experiments we always define $\tau = 4.0$. This value was found to deliver good results for all test cases.

D. Power optimization with EO

In this section we analyze the final solutions obtained with a timeout of 200 seconds. The goal is to study the reduction in power that EO delivers in comparison with SA for broad set of configurations. It is important to indicate that the results obtained by SA were selected by taking the best solution from all the α values, thus making the analysis independent of the cooling factor.

Three parameters are explored to obtain a comprehensive collection of test cases. Firstly, examples of different size are considered. These include the *64T*, *144T*, *256T* and *400T* configurations from Table III. Secondly, for every test case the size of the voltage islands is varied among 4, 8 and 16 processors. Thirdly, different ratios between the computation power P_{comp} and communication power P_{comm} are considered. This is an important parameter, as it reflects the ability of the approach to give priority to one power component or improve both simultaneously. Three values for P_{comp}/P_{comm} are explored: 0.2, 1.0 and 5.0. They and inspired by the results presented in [14].

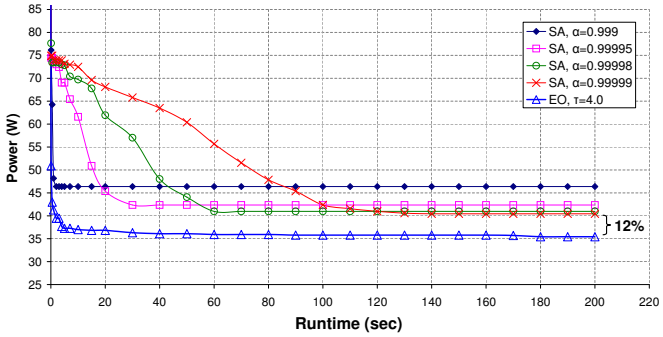


Fig. 4: Evolution of SA and EO solutions in time.

Figure 5 reports the power (equation (1)) of the EO solution with respect to the best value obtained by SA with various α. 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% (configuration *256T/16*, P_{comp}/P_{comm} = 5.0). Only for 3 of 36 explored configurations (*64T/4*, *64T/16* and *400T/4* with P_{comp}/P_{comm} = 0.2) EO was slightly worse than SA. The difference in this case did not exceed 2.0%.

EO tends to perform better 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 and random swaps do not concentrate on the components with worst fitness.

As an example, Fig. 6 shows the final voltage distributions for the *256T* example with $S_{vi} = 16$. The system has 16 voltage islands and each island contains 16 processors with a mixture of C1, C2 and C3 classes, as shown in Fig. 6(a). The final voltage assignment for each island is represented by the three colors in the figure. The solution obtained by SA has 8 islands at 1.2V, 5 at 1.0V and 3 at 0.8V. The one obtained by EO 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% smaller than the SA solution.

Another intuitive result is that the total power grows with the size of voltage islands (Fig 6(c)). The island size sets the number of tiles

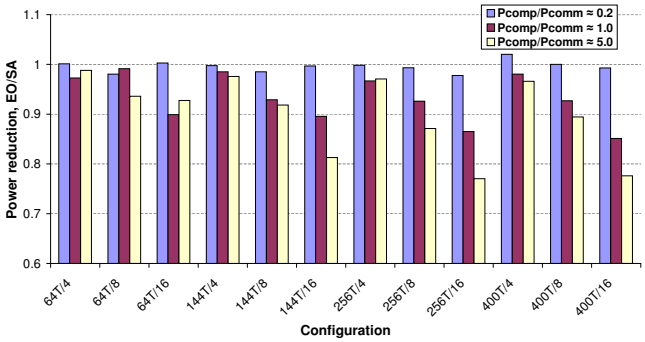
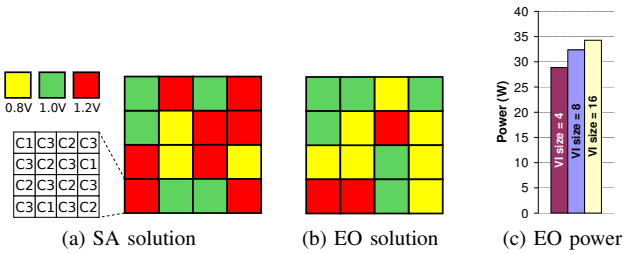


Fig. 5: Power reduction by EO with respect to SA.



(a) SA solution (b) EO solution (c) EO power

Fig. 6: Voltage distribution and power for *256T* example.

that must run at the same voltage. Hence, larger islands imply less mapping flexibility for individual tiles to reduce voltage.

E. Impact of link capacity and memory bandwidth

The link capacity and memory controller bandwidth constraints have a relevant influence on power consumption. In this example we fix the memory bandwidth and analyze how the solution changes as the link capacity constraint becomes more stringent. The dependency of power on the memory bandwidth has a similar trend. We use again the test case $256T$ with $S_{vi} = 16$ and set the bandwidth of each memory controller to 3 *Gbps*.

The results for SA and EO are plotted in Fig. 7. A sequence of solutions for different values of capacity constraints was obtained. The minimum capacity required to obtain feasible solutions was $Cap_{\min} = 0.91$ *Gbps*, and was reached by both methods. The trendlines included in the plot help to analyze the evolution of the solutions as the link capacity changes.

The tendency for power is to increase as capacity constraint tightens up. This happens principally due to the growth in communication cost as the tasks need to be spread to avoid congestion in the links.

Although the gap between the EO and SA costs decreases as the capacity approaches to Cap_{min}, EO wins in power for all considered values. This example represents the typical behavior, observed for both metaheuristics, when optimizing a configuration subject to capacity constraints.

VI. Conclusions

Design reuse will become a major paradigm for engineering many-core systems. This paper has addressed the problem of static task mapping for large-scale tiled CMPs with multiple voltage islands, as one of the approaches 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 metaheuristic has shown to be an efficient and scalable approach to solve this complex combinatorial problem.

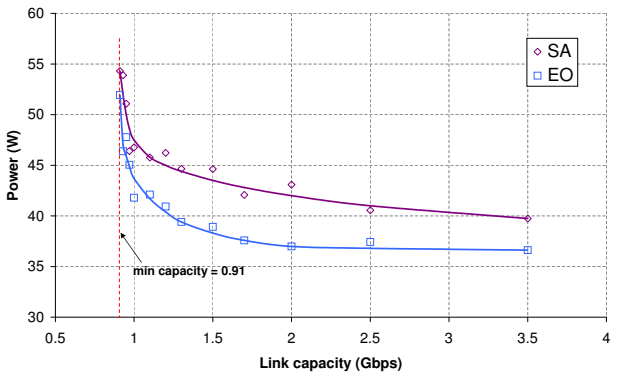


Fig. 7: Impact of link capacity on system power.

VII. 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] D. Pham et al., "Overview of the architecture, circuit design, and physical implementation of a first-generation cell processor," *Solid-State Circuits*, vol. 41, no. 1, pp. 179–196, 2006.

[2] S. Rusu et al., "A 45nm 8-core enterprise Xeon processor," in *Solid-State Circuits*, nov. 2009, pp. 9–12.

[3] T. Fisher et al., "Design solutions for the Bulldozer 32nm SOI 2-core processor module in an 8-core CPU," in *Solid-State Circuits*, feb. 2011, pp. 78–80.

[4] S. Bell et al., "TILE64 - processor: A 64-core SoC with mesh interconnect," in *Solid-State Circuits*, feb. 2008, pp. 88 –598.

[5] S. Vangal et al., "An 80-tile 1.28TFLOPS network-on-chip in 65nm CMOS," in *Solid-State Circuits*, feb. 2007, pp. 98 –589.

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

[7] M. Azimi et al., "Integration Challenges and Tradeoffs for Tera-scale Architectures," *Intel Technology Journal*, August 2007.

[8] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai, "The impact of performance asymmetry in emerging multicore architectures," in *International Symposium on Computer Architecture*, 2005, pp. 506–517.

[9] D. Lackey et al., "Managing power and performance for System-on-Chip designs using voltage islands," in *Proc. International Conf. Computer-Aided Design (ICCAD)*, 2002, pp. 195–202.

[10] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *High Performance Computer Architecture*, Feb. 2008, pp. 123–134.

[11] W.-K. Mak and J.-W. Chen, "Voltage island generation under performance requirement for soc designs," in *Proc. of Asia and South Pacific Design Automation Conference*, 2007, pp. 798–803.

[12] P. Ghosh and A. Sen, "Energy efficient mapping and voltage islanding for regular NoC under design constraints," *J. High Perform. Syst. Archit.*, vol. 2, pp. 132–144, August 2010.

[13] W.-L. Hung et al., "Temperature-aware voltage islands architecting in System-on-Chip design," in *Proc. International Conf. Computer Design (ICCD)*, 2005, pp. 689–696.

[14] G. Varatkar and R. Marculescu, "Communication-aware task scheduling and voltage selection for total systems energy minimization," in *Proc. International Conf. Computer-Aided Design (ICCAD)*, 2003.

[15] G. Chen, F. Li, S. Son, and M. Kandemir, "Application mapping for chip multiprocessors," in *Proc. ACM/IEEE Design Automation Conference*, 2008, pp. 620 –625.

[16] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers, Inc., 2003.

[17] S. Boettcher and A. G. Percus, "Extremal optimization: Methods derived from co-evolution," in *Genetic and Evolutionary Computation*, 1999, pp. 825–832.

[18] I. De Falco, A. Della Cioppa, D. Maisto, U. Scafuri, and E. Tarantino, "A multiobjective extremal optimization algorithm for efficient mapping in grids," in *Applications of Soft Computing*, vol. 58, 2009, pp. 367–377.

[19] G. D. Micheli and L. Benini, *Networks on Chips: Technology and Tools (Systems on Silicon)*. Morgan Kaufmann Publishers, Inc., 2006.

[20] D. Abts, N. D. E. Jerger, J. Kim, D. Gibson, and M. H. Lipasti, "Achieving predictable performance through better memory controller placement in many-core CMPs," in *Int. Symp. Computer Architecture*, 2009, pp. 451–461.

[21] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[22] T. Zhou, W.-J. Bai, L.-J. Cheng, and B.-H. Wang, "Continuous extremal optimization for Lennard-Jones clusters," *Phys. Rev. E*, vol. 72, no. 1, Jul 2005.

[23] A. Pinto, L. P. Carloni, and A. L. Sangiovanni-Vincentelli, "A methodology for constraint-driven synthesis of on-chip communications," *IEEE Transactions on Computer-Aided Design*, vol. 28, no. 3, pp. 364–377, Mar. 2009.

[24] "CPLEX," http://www.ilog.com/products/cplex.