Architectural Exploration of Large-Scale Hierarchical Chip Multiprocessors

Nikita Nikitin, Javier de San Pedro, and Jordi Cortadella, *Member, IEEE*

***Abstract*—The continuous scaling of nanoelectronics is increasing the complexity of chip multiprocessors (CMPs) and exacerbating the memory wall problem. As CMPs become more complex, the memory subsystem is organized into more hierarchical structures to better exploit locality. To efficiently discover promising architectures within the rapidly growing search space, exhaustive exploration is replaced with tools that implement intelligent search strategies. Moreover, faster analytical models are preferred to costly simulations for estimating the performance and power of CMP architectures. The memory traffic generated by CMP cores has a cyclic dependency with the latency of the memory subsystem, which critically affects the overall system performance. Based on this observation, a novel scalable analytical method is proposed to estimate the performance of highly parallel CMPs (hundreds or thousands of cores) with hierarchical interconnect networks. The method can use customizable probabilistic models and solves the cyclic dependencies between traffic and latency by using a fixed-point strategy. By using the analytical model as a performance and power estimator, an efficient metaheuristic-based search is proposed for the exploration of large design spaces. The proposed techniques are shown to be very accurate and a promising strategy when compared to the results obtained by simulation.**

***Index Terms*—Analytical modeling, chip multiprocessing, design space exploration, metaheuristics, numerical methods.**

I. Introduction

THE CONTINUOUS shrinking of CMOS technology has enabled the integration of multiple cores and distributed memory in one chip. Parallelism has also been one of the paradigms to make computations more power efficient. In the last few years, multicore systems have evolved from having few cores to complexity of chip multiprocessors (CMPs) with hundreds of computing units [1], [2].

Tiled CMPs are an effective approach to architect general-purpose processors under the intense time-to-market pressure [3], [4]. The replication of tiles provides a rapid way of floorplanning many computing units in one chip and communicating them with scalable interconnect fabrics. Fig. 1(a) shows an example of a CMP with 16 tiles, each including a computing core (C) with private *L1* cache, a larger on-chip cache (L2), and a router (R) that communicates with the on-chip interconnection network (a mesh). Four memory controllers (MC) provide access to the off-chip memory.

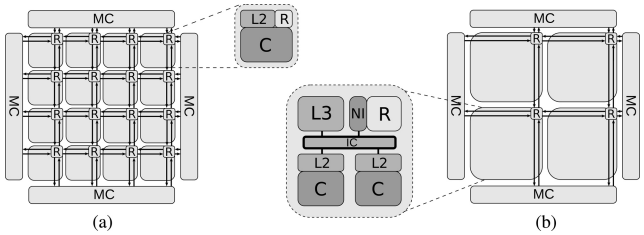


Fig. 1. CMP layouts. (a) Flat. (b) Hierarchical.

To exploit the locality of memory references, hierarchical interconnects have been proposed [4], [5]. Several cores can be grouped into one cluster to share the on-chip cache, accessible through a local interconnect (e.g., bus, crossbar, ring, etc.). Hierarchy increases the intracluster hit-ratio and reduces the traffic in the top-level interconnect. Fig. 1(b) shows an implementation of a CMP with four clusters. Each cluster has two cores with private caches, a shared cache (L3), a local interconnect (IC), a router, and a network interface (NI).

Given the vast space of design parameters, CMP designers are faced with the complex problem of selecting the best architecture subject to a set of constraints. Many design options must be explored, such as the variety of core implementations, interconnect types, topologies, cache hierarchies, and memory management policies. Moreover, the amount of configurations increases drastically as the technology advances, allowing more cores and memory to fit into the chip area.

The complexity of the search space makes simulation-driven exhaustive exploration of all design points prohibitively expensive. One of the ways to handle this problem is to decrease the number of points to be considered. Exhaustive exploration is therefore replaced with an intelligent search strategy, e.g., leveraging the methods of machine learning [6] or design of experiments [7].

Another option to shorten the exploration time is to reduce the cost of evaluating every design point, by using the analytical models. Along this line, several models for CMP exploration have been recently proposed in [8] and [9].

The goal of this paper is to develop a scalable and parameterizable methodology for efficient architectural exploration of large-scale hierarchical CMPs within vast design spaces that are far from being explorable exhaustively. The proposed approach is built on top of an analytical power-performance

Manuscript received October 7, 2012; revised May 29, 2013; accepted June 6, 2013. Date of current version September 16, 2013. This work was supported in part by a gift from Intel Corporation, the Project FORMALISM under Grant CICYT TIN2007-66523, the Generalitat de Catalunya under Grant ALBCOM-SGR 2009-2013, and FPI Grant BES-2008-004612. This paper was recommended by Associate Editor Y. Xie.

N. Nikitin was with the Department of Software, Universitat Politècnica de Catalunya, Barcelona 08034, Spain. He is now with the Department of Computer and Information Science, Norwegian University of Science and Technology, Trondheim 7491, Norway (e-mail: nikita.nikitin@idi.ntnu.no).

J. de San Pedro and J. Cortadella are with the Department of Software, Universitat Politècnica de Catalunya, Barcelona 08034, Spain (e-mail: jspedro@lsi.upc.edu; jordi.cortadella@upc.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2013.2272539

model for CMPs, and performs an accurate metaheuristic-based search that does not require simulation.

Analytical models for CMPs are based on the models for their components, mainly the cores and the memory subsystem. The fundamental problem in evaluating the CMP performance is the calculation of the latency for memory requests, given the parameters of the interconnect fabrics and memory hierarchy. This latency critically depends on the interconnect contention—a phenomenon that cannot be underestimated when exploring the architectural design space. In fact, ignoring contention leads to optimistic values for latency and throughput, and may overestimate architectures with saturated interconnects [10]. Accounting for contention is particularly important for hierarchical CMPs, as it is essential to verify that the required bandwidth is delivered at all levels.

The efficiency of the analytical models for CMP exploration is typically traded off with their quality, ranging from rapid high-level performance estimators for a complete CMP [9], to nonstationary multifractal traffic models for workload characterization [11]. In this direction, the model introduced in this paper takes an intermediate position. On one hand, it is built under the assumption of average-case modeling, in both spatial and timing domains, contributing to the model efficiency. On the other hand, guided by the importance and complexity of contention modeling, we aim at applying analytical methods, which can be parameterized with arbitrary models for CMP components.

The developed model is shown to be a light-weight power-performance estimator, which can be used in exploration frameworks for rapidly pruning the design spaces with substantial accuracy. At the same time, it retains modularity and allows replacement of individual models for cores, memories or the interconnect, without the need to change the overall approach. Section II explains some of the possible directions, in which the proposed model can be improved.

More precisely, the contributions of the paper are as follows.

1) An analytical method for estimating the performance and power of hierarchical CMPs is proposed. The model not only captures the behavior of various core architectures and cache hierarchies, but also estimates the traffic and contention of different interconnect topologies.
2) To model the contention in the interconnect, the cyclic dependency between latency and memory traffic is formulated as a system of nonlinear equations. Two numerical methods are used to solve it efficiently: fixed-point iteration and bisection. These methods can be parametrized with any black-box model for latency, making the approach flexible to incorporate new interconnect models.
3) Two metaheuristics, simulated annealing (SA) [12] and extremal optimization (EO) [13], are customized for architectural exploration. They use the proposed analytical model as the cost function of every design point. The experiments demonstrate the effectiveness of the method for power-performance exploration.

II. Related Work

The field of CMP design space exploration has been widely studied in the last few years. Many simulation-based frameworks extensively investigate the parameters of multicore architectures and memory hierarchies. Exhaustive simulations were previously used for performance-oriented exploration of core and cache organizations and the off-chip memory bandwidth [14]. Other works emphasize the importance of the joint optimization for power, performance and thermal characteristics [15], and the impact of chip floorplan [16].

Efficient simulation-driven exploration was investigated by the means of intelligent search techniques. In [17], predictive modeling is used to reduce the search space by simulating sample points and teach the models to describe relationships among design variables. The design of experiments paradigm for architectural exploration is proposed in [7]. The work in [6] compares the application of several metaheuristics with machine-learning methods, reporting orders of runtime savings compared to exhaustive simulation.

Analytical models appeared to replace costly simulations and provide quick estimations of the performance of the architectures. The model in [8] studies the trade off between the number of cores and the on-chip memory size for throughput optimization. However, it ignores contention in nonuniform memory architectures, typical for many-core systems. In [18], the authors introduce an energy-performance analytical model for CMP architectures, however, they only consider bus interconnects with a simplified contention model. The work in [19] analyzes finite cache penalties in memory hierarchies, but the interconnects are also restricted to buses. McPAT [20] is another powerful tool with low-level analytical models for area, timing, and power of multicore architectures. However, the lack of traffic and throughput models makes it unsuitable for characterization of hierarchical interconnects.

Combining simulation with analytical modeling was also considered for trading off the accuracy and efficiency in CMP performance analysis. In this direction, FIST proposed approximating the behavior of network-on-chip (NoC) routers with load-delay curves, replacing the need for modeling packet propagation [21]. Reservation-based timing models developed in [22] represent another approach to modeling contention in virtual cut-through networks, by tracking the periods of link acquisitions.

In this paper, we propose a generic method for analytical modeling of hierarchical CMPs. Our method can be parametrized by arbitrary models for CMP components. As such, we apply the mechanistic model in [23] to reflect architectural properties of the cores. We consider the application of the latency model in [24] due to its flexibility. However, other models can also be considered, such as [25]. It introduces an accurate model for heterogeneous NoCs that can be useful for modeling a variable number of virtual channels and link capacities at different levels of the hierarchical interconnect. In [26], an approach similar to [24] is proposed, offering an accurate backpressure analysis at the cost of model efficiency.

Queueing approaches were shown to have limitations, such as the assumption of modeling system in equilibrium and the inability to capture the multifractal nature of the workload traffic [27], [11]. These recent works propose alternative nonstationary traffic models, which are based on considering the behavior of interconnect queues using a statistical physics approach. However, the analysis in [27] and [11] considers the traffic pattern in dependence of an inseparable combination of the architecture and the workload. For the problem of architectural exploration, the separation of workload and

architectural parameters is required in order to find the best candidate architecture for the given workload application. This separation remains an open question, which is however out of scope of this paper. Finally, we refer the interested reader to [28] for a comprehensive analysis of the research problems and techniques in interconnect modeling.

This paper is also the first one to integrate an analytical model for CMP with metaheuristic optimization, thus bringing additional degree of speed-up for exploration. One of the benefits of metaheuristics with respect to analytical optimization methods proposed by [18] is the ability to generate a moderately-sized set of nearly optimal configurations. These can be further simulated to reduce the chance of choosing suboptimal architectures due to inaccuracy of the model.

Heterogeneous architectures are considered as a promising strategy for engineering energy-efficient many-core systems [29], [30]. In addition to incorporating different instruction-set architectures on a chip, in [31], it was shown that technological heterogeneity is another dimension for exploring energy-efficient computation beyond the CMOS. While homogeneous tiled CMPs are at the focus of the analysis presented in this paper, the proposed technique can also be applied for heterogeneous systems. However, in order to capture the benefits of heterogeneity, elaborated workload models are essential, which is the primary objective of future work.

III. Analytical Performance and Power Models

This section introduces the models for the evaluation of CMP performance and power and formulates the cyclic dependency between traffic and latency.

A. Assumptions and Input Parameters of the Models

This paper focuses on systems with two-level hierarchical interconnect fabrics. However, the approach can be applied for an arbitrary number of hierarchical levels. Several components are grouped into a cluster: cores, components of the memory subsystem and the local interconnect. The top-level interconnect provides communication between the clusters and access to the off-chip memory [Fig. 1(b)].

The system has in total N cores with parametrizeable architectures: in-order, out-of-order, single-, or multithreaded. The workload model assumes that every core is executing an application, characterized by two parameters. $\texttt{IPC}_0$ is the ideal throughput of the core for the application, i.e., the amount of instructions per cycle executed by the core, assuming zero-latency memory. MPI is the average number of memory references generated per instruction.

There are several ways how exploration can be performed for multiple applications. In the first scenario, all cores can execute only one type of application simultaneously, which can change in time. In this case, it is possible to evaluate the performance of every configuration with a weighted objective function. For example, if two types of applications are executed, $\mathcal{A}_1$ and $\mathcal{A}_2$, some configuration delivers performance $\texttt{IPC}(\mathcal{A}_1)$ and $\texttt{IPC}(\mathcal{A}_2)$, respectively, then the aggregate performance can be defined as

$$\texttt{IPC} = a_1 \cdot \texttt{IPC}(\mathcal{A}_1) + a_2 \cdot \texttt{IPC}(\mathcal{A}_1) \tag{1}$$

where a_1 and a_2 are the user-defined weights. In the other scenario, when several types of applications can be executed in parallel, they have to be assigned to the cores assuming some predefined mapping algorithm. The exploration procedure, in this case, can be applied by executing the mapping algorithm for each configuration generated during the exploration.

Without loss of generality, we assume that the memory subsystem has four hierarchy levels. Every core has a private *L1* cache and possibly, a private *L2* cache of larger size but higher latency. The clusters incorporate modules of a distributed *L3* cache, shared by all cores. The off-chip memory is accessible via a set of memory controllers. The latencies of the caches and off-chip memory are parameters of the model.

The term memory flow is used to denote a feasible communication between a core and a component of the memory subsystem. For example, each core may access its own *L1* or *L2* caches, or any of the *L3* modules or *MC*s. For simplicity of the analysis, in this section, we only consider the request and reply flows between the cores and the memories. However, a specific coherence protocol can be modeled by adding the traffic of synchronization flows. The experiments for this paper are performed using an abstraction of a cache coherency protocol, as described in Section VI-B.

The set of all possible memory flows for core c is denoted as $F(c)$. Every flow $f \in F(c)$ is realizable with probability p_f, that defines the probability for c to request data from a certain memory component. In our paper, we calculate these probabilities using a model of cache miss behavior for the workload in consideration, which represents the dependency between miss ratio and cache size. A power-law model was proven to be a good approximation [32]

$$Miss(S) = \kappa S^{-\alpha} \tag{2}$$

where S is the cache size, and κ, α are the model parameters (Fig. 2). Alternatively, the miss model can be precharacterized using simulation and specified as a set of points in the cache-size/miss-ratio plane.

Since *L3* is a distributed cache, its access latency depends on the cluster where the requested data is stored. The algorithms for data allocation are typically implemented in the operating system and are out of scope of this paper. Hence, we assume a smart allocation algorithm, which favors the locality of communication. The probability to find the data in a particular cluster is assumed to be inversely proportional to the distance between the requesting core and the cluster. This assumption increases the total throughput of the system, since it decreases the average memory access latency. In general, the method can be parameterized with any other model for distributed cache.

B. Static Latency

In this section, we describe how to calculate the average static latency of memory accesses for a core c in the presence of memory hierarchy. Given the probability p_f for each particular flow $f \in F(c)$ and its static latency L_f, the average static latency L_c^{st} is

$$L_c^{st} = \sum_{f \in F(c)} p_f L_f. \tag{3}$$

Since requests to *L3* and *MC* are sent via the communication network, its delay must also be considered. This delay is defined using the routing function $\mathcal{R} : f \to \pi(f)$, that for any flow f returns its routing path $\pi(f)$. The total latency to access

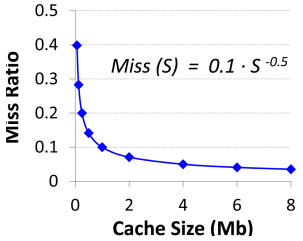


Fig. 2. Power-law cache miss model.

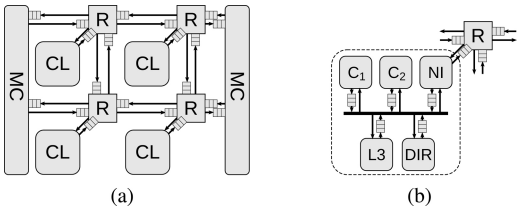


Fig. 3. Queueing model for (a) mesh NoC and (b) cluster.

an *L3* instance is the sum of the network traversal latency along the path $\pi(f)$ and the *L3* latency. The total latency of the off-chip memory accesses is calculated likewise.

Note that the routing function and memory latency depend on the interconnect topology. The proposed model can be applied to arbitrary topologies, including hierarchical multilevel interconnects. In this paper, we consider two-level interconnects with mesh at the top level and buses or rings in clusters. For rings, shortest-path routing is assumed, while for meshes XY-routing is used [33]. Dateline with two virtual channels guarantees deadlock freedom in rings. To support a different routing function, one should modify the routing procedure, which calculates the traffic distribution according to the selected routes. In general, any deterministic or even adaptive routing can be used, specifying the probabilities for selecting every path.

The flow probabilities p_f are obtained using the dependency of miss ratio on the cache size, *Miss(S)*, given by (2). Assuming the sizes S_{L1}, S_{L2} of the two low-level caches, the probabilities to access them are

$$\begin{aligned} p_{L1} &= 1 - Miss(S_{L1}), \\ p_{L2} &= (1 - p_{L1})(1 - Miss(S_{L2})). \end{aligned}$$

As *L3* is shared, the miss ratio is defined by the effective *L3* size, S_{L3}^{eff}, seen by each core [34]. To estimate S_{L3}^{eff} we use the concept of the average number of cores sharing each line, as proposed by [34]. The probability to access *L3* is then

$$p_{L3} = (1 - p_{L1})(1 - p_{L2})(1 - Miss(S_{L3}^{eff})).$$

Finally, p_{L3} should be multiplied by the probability to find the data in a particular *L3* instance (cluster). A similar strategy is used to calculate the probabilities of flows to every memory controller.

C. Queueing Model for the On-Chip Interconnect

Equation (3) describes the static latency of memory accesses. Another important part of the communication delay is the dynamic or contention latency [33]. Contention happens in the interconnect fabrics when several packets compete for the same shared resource, such as a bus or an NoC link. This results in additional delays experienced by packets in the buffers distributed over the on-chip interconnect. One of the approaches to estimate the contention delays is to model the CMP as a system of queues and apply queuing theory to calculate the buffer delays.

In this paper, two-level hierarchical interconnects are assumed, with a 2-D mesh at the top level. At the lowest (cluster) level, buses, uni- and bi-directional rings are used for the interconnection. Fig. 3(a) shows the queueing representation of the top-level mesh. The mesh routers (R) have up to five input-buffered ports to store the incoming flits. The primary ports of the routers are connected to the clusters (CL), which in case of a flat CMP organization may consist of one device [e.g., a core with private caches in Fig. 1(a)]. Fig. 3(b) presents an example of queueing model for a bus-based cluster, corresponding to one tile of the hierarchical CMP. This cluster consists of five devices, communicating via a shared bus: two cores with private caches, an instance of an *L3* shared cache, a directory and a network interface. Every device has a buffer to store the requests to the bus. To distribute the off-chip memory traffic uniformly over the mesh and avoid high contention of certain routers, we assume that memory controllers have multiple connections to the mesh, as shown in Fig. 3(a).

D. Total Memory Latency

The average total latency for core c, L_c, is calculated by adding the static latency (L_c^{st}) and the queue delays along the communication paths (w_q). Hence, given the paths $\pi(f)$ for every flow f, we extend equation (3) accordingly

$$L_c = L_c^{st} + \sum_{f \in F(c)} \left(p_f \sum_{q \in \pi(f)} w_q \right). \tag{4}$$

To find the values for w_q, an analytical model for the on-chip interconnects can be used. In this paper, we apply the model from [24], which offers a convenient definition of queue delays via the injection rates in closed form. The method calculates the probabilities of the packets coming from different bus (or router) inputs to move toward the same output. It represents an efficient generalization of the M/G/1 queueing model [35]. The efficiency of this model is essential for our iterative procedure, described in Section IV. Another advantage of this model is the capability to deal with a variety of interconnect types, such as buses and router-based topologies (including meshes, uni- and bi-directional rings, and other topologies).

Given the vector of injection rates into the interconnect, $\overline{\lambda} \in \mathbb{R}^N$, the model in [24] proposes to express queue delays in the form of a system of equations with a matrix W

$$\overline{w}_q = W \times \overline{\lambda} \tag{5}$$

where $\overline{w}_q$ is the vector of delays for all queues of the interconnect. The exact form of the matrix W is given by the expressions (5) and (18) in [24]. What only remains is to compute the rates $\overline{\lambda}$, which is covered in the next section.

E. Throughput Model

The throughput of a CMP and the traffic in the interconnect are closely related. To derive the exact dependencies, we start with the performance model for a single core, given in [36].

For a core with the average rate of accesses to remote memory (*RemRate*), and the cost of an access (*RemCost*), the average number of cycles for executing an instruction, CPI, is

$$\mathtt{CPI} = \mathtt{CPI}_0 + \mathit{RemRate} \cdot \mathit{RemCost} \tag{6}$$

where $\mathtt{CPI}_0 = 1/\mathtt{IPC}_0$ is the ideal CPI, derived under the assumption of zero-latency memory. Note that $\mathtt{IPC}_0$ is a function of both the core and the workload. For a single-threaded strictly in-order core, the cost of a remote access is the average latency, given by (4), and the remote rate is given by the MPI value. As throughput is typically measured in IPC, the reciprocal of CPI, from (6) we obtain

$$\theta_c = \frac{1}{\mathtt{CPI}} = \frac{1}{\frac{1}{\mathtt{IPC}_0} + \mathtt{MPI} \cdot L_c}. \tag{7}$$

The throughput of the entire CMP, Θ, is then calculated as the total performance of individual cores: $\Theta = \sum_c \theta_c$.

The rate of memory accesses, λ_c, is the probability for a core to issue a remote memory request per cycle. λ_c is proportional to the core throughput and the MPI

$$\lambda_c = \theta_c \cdot \mathtt{MPI} = \frac{\mathtt{MPI}}{\frac{1}{\mathtt{IPC}_0} + \mathtt{MPI} \cdot L_c}. \tag{8}$$

F. Multithreaded and Out-of-Order Cores

Equation (8) can be extended for the case of multithreaded and out-of-order cores. A multithreaded core can be modeled as a group of single-threaded cores. The latency for each thread remains L_c, but the total memory rate becomes $\lambda_c^{mt} = M\lambda_c$, where M is the number of threads.

The difference in modeling an out-of-order core is that the remote memory access does not force the core to stall, hence, the effective remote latency L_c decreases [36]. Following the techniques in [23], we make two adjustments to capture this behavior in our model. First, the authors in [23] consider the short latencies of access to *L1* and *L2* hidden by instruction reordering. In our model, it can be modeled by excluding the flows between the core and the first two levels of local cache from the memory-flow set. $F_{OoO}(c) = F(c) \setminus \{f(c, L1), f(c, L2)\}$. Equation (4) is used with the new $F_{OoO}(c)$ to calculate the average total latency.

The second adjustment to the model is done by considering memory-level parallelism of the workload (MLP), which is the average number of memory requests issued in parallel [37]. When a core issues several requests in parallel, the latency penalty is amortized due to overlapping of the requests. We capture this fact by adjusting the MPI value to reduce the fraction of memory requests per instruction (and hence the penalty)

$$\mathtt{MPI}_{OoO} = \frac{\mathtt{MPI}}{\mathtt{MLP}}.$$

The average MLP can be obtained by workload profiling. This new MPI value is used with (7) and (8) to obtain the throughput and traffic of the out-of-order cores.

Although the applied core model is simplistic, it can be extended straightforwardly with microarchitectural aspects, e.g., as discussed in [23]. The separation of the models for core and workload is desirable, since we would be able to explicitly extract the instruction- and thread-level parallelism of the applications. Note that modifying the core model independently of the interconnect model is possible due to the modularity of the proposed approach.

G. Cyclic Dependency Between Memory Latency and Traffic

In order to calculate the buffer delays, (5) requires the injection rates at every input (source) of the interconnect, while (8) gives the rates of request generation per core. Note that the injection rates in a flat interconnect are directly defined by the core rates: for a CMP with N cores, $\overline{\lambda} = \{\lambda_1, .., \lambda_N\}$. In case of a hierarchical interconnect fabric, the core rates will correspond to the injection rates at the sources of the cluster-level interconnects, such as the bus in Fig. 3(b). The injection rates to the top-level mesh can be calculated, given the fraction of intercluster traffic. The latter is defined by the probabilities of access to the *L3* and the off-chip memory, discussed in Section III-B. Below we directly consider the dependency of memory latency on the core rates.

From (4), (5), and (8) we observe the following system of dependencies:

$$\forall c = 1, .., N: \quad \begin{cases} L_c = L_c(\overline{\lambda}, \overline{w}_q) \\ \lambda_c = \lambda_c(L_c) \end{cases} \tag{9}$$

where L_c is the total memory latency for core c, defined by (4) as the function of the injection rates of all cores, $\overline{\lambda}$, and queue delays $\overline{w}_q$ in (5). λ_c is the injection rate for core c, the function of its proper latency, L_c (8). Hence, the dependencies (4), (5), and (8) create a system of equations with respect to the vectors of variables $\overline{L}$, $\overline{\lambda}$, and $\overline{w}_q$.

This result is quite intuitive: the latency of the memory requests traversing the interconnect depends on the injection rate of requests, due to the network contention. On the other hand, the request rate is determined by the latency, as no new memory requests are issued if the execution of cores stalls due to the absence of data. System (9) emphasizes the cyclic dependency between the latency and rate of memory requests. In Section IV, we describe the methods to resolve this dependency.

H. Power Model

In this paper, we model power analytically using a first-order approximation of leakage and dynamic power for individual components, such as cores, caches and interconnects. Leakage power of component c is proportional to the unit leakage $p_{leak,c}$ and the area A_c

$$P_{leak,c} = p_{leak,c} \cdot A_c. \tag{10}$$

Dynamic power is primarily defined by the utilization of components. For cores, it is proportional to the throughput of the core θ_c, the energy of executing single instruction $E_{inst,c}$, and the core frequency $Freq_c$

$$P_{dyn,c} = E_{inst,c} \cdot \theta_c \cdot Freq_c. \tag{11}$$

Similarly, the cache power depends on the number of accesses to the cache per cycle (e.g., traffic), Λ_c, and the energy per access $E_{acc,c}$

$$P_{dyn,c} = E_{acc,c} \cdot \Lambda_c \cdot Freq_c. \tag{12}$$

Dynamic power for the components of on-chip interconnect, such as routers and links, is proportional to the traffic through the component, Λ_c, and the energy per flit transmission $E_{flit,c}$.

$$P_{dyn,c} = E_{flit,c} \cdot \Lambda_c \cdot Freq_c.$$

Here the areas A_c and the performance metrics θ_c and Λ_c of the components are calculated by the analytical model. Frequency is a parameter of exploration. To find the power and energy coefficients, we employ the data obtained from [38] for the cores, CACTI 5.3 model [39] for caches and Orion 2.0 model [40] for on-chip routers and links.

The energy per instruction is approximated using (10)–(12) and the values for the core thermal design power, nominal throughput, and frequency are obtained from [41].

IV. Analytical Methods for Latency Estimation

The solution of a nonlinear system (9) can be found using one of the Newton-based methods [42], typically available in a generic solver, such as MATLAB [43]. The downside of this approach is the need for the calculation of function derivatives, which is a computationally expensive process. Apart from that, this approach only works for analytical models that can be represented with closed-form equations.

In this section, we propose two numerical methods to efficiently resolve the cyclic dependency (9). The first method, fixed-point iteration, delivers the exact solution in case of convergence and can be applied to arbitrary configurations. The second approach, based on bisection, always converges for our problem but finds an approximate solution. However, it resulted to be a good approximation for tiled homogeneous CMPs (Section IV-B).

The subgradient method [44] was also considered as an alternative to bisection. Although it is more accurate, it is also significantly slower. In this paper, we focus on the first two methods, since their performance and quality represent the best compromise for the evaluation of homogeneous architectures.

A. Fixed-Point Iteration

The algorithm proposed, in this section, is a popular numerical method for solving systems of nonlinear equations [42]. While the theoretical speed of convergence of this method is relatively slow, it performs well in practice due to its low cost for a single iteration. Given a system of equations in the form

$$\overline{x} = F(\overline{x}) \tag{13}$$

where $\overline{x}$ is the vector of variables and F is the system matrix, and an initial guess $\overline{x}_0$, the following iterative procedure can be used to find a solution $\overline{x}^*$ (fixed point) of the system:

$$\overline{x}_{n+1} = F(\overline{x}_n), \ n \geq 0.$$

In our setting, $\overline{x}$ is composed of the variables $\{\overline{L}, \overline{\lambda}, \overline{w}_q\}$ and matrix F is defined by the right-hand terms of (4), (5), and (8). For the initial guess, $\overline{x}_0$, we use static latencies (3) and compute other values using the same equations: $\overline{L}_0 = \overline{L}^{st}$, $\overline{\lambda}_0 = \lambda(\overline{L}_0)$, $\overline{w}_{q,0} = W(\overline{\lambda}_0)$.

The benefit of the proposed method is that it does not require closed-form analytical expressions for latencies. Furthermore, any black-box model for the dependency of interconnect latency on injection rate can be used. The method hence maintains the modular structure of hierarchical interconnects and permits plugging independent models for different topologies.

As a numerical method, fixed-point iteration is subject to convergence issues. For a system in the form (13), the sufficient condition for convergence is [42]

$$\sum_i \left| \frac{\partial F}{\partial x_i} \right| < 1.$$

In our case, this requires the latency to grow slowly with the injection rate, and vice versa. This condition holds for the communication fabrics that perform far from their saturation throughput (for instance, see chapter 23 in [33]). Although this is a sufficient condition, it is not necessary for convergence. In practice, we observe that for the majority of configurations the iterative procedure converges.

A second issue of the fixed-point iteration is due to the analytical models based on queueing theory: the queueing models work under the assumption of the system being in the steady state [35]. This means that for any router with service time T and the sum of arrival rates to its inputs λ, the following condition must hold: $\lambda T < 1$. In other words, there should be no unbounded packet accumulation in the input queues of the router. Unfortunately, this requirement may be not satisfied by the initial solution, due to the underestimated latency, and hence overestimated λ_c. To handle this situation as well as the configurations for which the fixed-point iteration diverges, we propose a method based on the bisection search of λ_c, to find a reasonable and fast approximation to the solution.

B. Bisection Search for Traffic Rate

The advantage of the bisection method is that it always converges for our model (due to the intermediate value theorem [42]). Since every core generates traffic at certain rate, λ_c, multidimensional bisection [45] can be applied to find the exact rates. However, a good approximation to the exact rates can be obtained by using the less complex unidimensional bisection. By simulation, we observed that the traffic rates of the cores of tiled CMPs with homogeneous clusters change proportionally to their estimates, obtained by the static latency. Hence, we initialize the vector of injection rates $\overline{\lambda}$ with the values estimated by static latency, and on every bisection step adjust all rates in the same proportion.

To introduce the bisection more formally, let us rewrite (8) by isolating L_c, and using the star symbol to distinguish from the latency in (4)

$$L_c^*(\lambda_c) = \frac{1}{\lambda_c} - \frac{1}{\mathtt{MPI} \cdot \mathtt{IPC}_0}. \tag{14}$$

Notice that as opposed to the latency L_c in (4), which increases with the injection rate, the latency L_c^* in (14) decreases with λ_c. The reason is that (4) models the interconnect contention, disregarding its impact on the performance of cores. In turn, (14) models the throughput of cores, neglecting the impact of contention on the memory latency. This emphasizes the cyclic dependency between the memory latency and traffic, and the need to solve the system (9).

From (4) and (14), we define the average latencies $L(\overline{\lambda})$ and $L^*(\overline{\lambda})$ as the functions of the vector $\overline{\lambda}$

$$L(\overline{\lambda}) = \frac{1}{N} \sum_{c=1}^{N} L_c(\overline{\lambda}), \quad L^*(\overline{\lambda}) = \frac{1}{N} \sum_{c=1}^{N} L_c^*(\lambda_c).$$

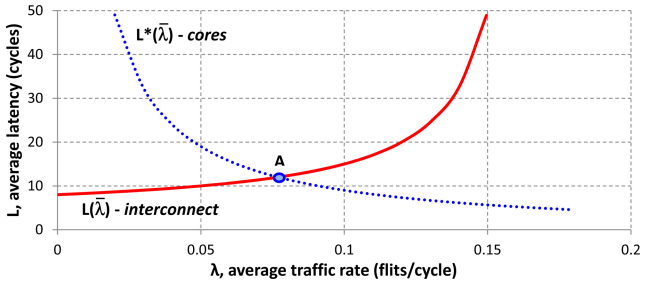


Fig. 4. Behavior of the latency functions $L(\bar{\lambda})$ and $L^*(\bar{\lambda})$.

Finally, we introduce the latency difference function, $F(\bar{\lambda})$

$$F(\bar{\lambda}) = L(\bar{\lambda}) - L^*(\bar{\lambda}).$$

Fig. 4 shows the typical behavior of these functions. To depict a 2-D view of this behavior, we plot $L(\bar{\lambda})$ and $L^*(\bar{\lambda})$ as a function of the average rate $\Lambda = \frac{1}{N}\sum_{c=1}^{N}\lambda_c$. The curve $L^*(\bar{\lambda})$ shows that the average rate of memory requests increases as the latency decreases. On the contrary, $L(\bar{\lambda})$ shows that the average latency increases with the injection rate. The real values for latency and traffic are defined by the intersection point A of these curves, that can be found as a root of $F(\bar{\lambda})$. Hence, we use the bisection as a root-finding method, that does not require the exact knowledge of the function $F(\bar{\lambda})$ and can be used with any black-box analytical model for latency.

Bisection searches for $\bar{\lambda}$ that satisfies the condition $|F(\bar{\lambda})| < \epsilon$, where ϵ is the solution tolerance. The initial range for $\bar{\lambda}$ is limited by the traffic, obtained with static latency. $\bar{\lambda}_{min} = \bar{0}$, $\bar{\lambda}_{max} = \bar{\lambda}(L_c^{st})$. Assuming the proportionality in variation of the individual components of $\bar{\lambda}$, all components are updated simultaneously. For any pair of consecutive iterations i and $i+1$, either $\bar{\lambda}_{min}^{i+1} = \bar{\lambda}^i$ when $F(\bar{\lambda}^i) < 0$, or $\bar{\lambda}_{max}^{i+1} = \bar{\lambda}^i$ when $F(\bar{\lambda}^i) > 0$. The iteration is continued until the required tolerance for $F(\bar{\lambda})$ or $\bar{\lambda}$ is met [42].

V. Metaheuristics for Architectural Exploration

The architectural exploration for CMPs must be performed on a highly discrete design space. For this reason, hill-climbing strategies for combinatorial optimization are a promising approach, as it was shown in [6].

In this paper, we consider two metaheuristics based on probabilistic extensions of hill climbing: SA [12] and EO [13]. Both methods are inspired by statistical physics and are commonly used to solve complex combinatorial problems. The benefits of SA are the ability to escape from local optima during the exploration of solutions and the easiness of customizing the algorithm for every particular combinatorial problem by defining local transformations. These are the main reasons why it has been successfully applied in various complex design automation problems [46]. In the context of CMP exploration, SA can be efficiently used by defining local transformations that can progressively modify the variables of the system.

EO has recently emerged as a very competitive alternative to SA. For our problem, both heuristics exhibit similar performance and results, although for some design spaces one behaves more efficiently than the other.

TABLE I
Configuration Parameters for Two Types of Cores

Name	Description
X, Y	Dimensions of the top-level mesh
$\mathcal{I}$	Type of interconnect in a cluster
N_1, N_2	Number of the first/second-type cores in a cluster
$L1_1, L1_2$	*L1* cache size for the first/second type of cores
$L2_1, L2_2$	*L2* cache size for the first/second type of cores

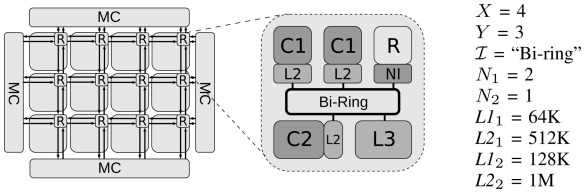


Fig. 5. Configuration example and its variables.

SA and EO are much easier to customize than other strategies like evolutionary algorithms [47], in which the crossover operators of configurations with heterogeneous variables often generate inconsistent solutions.

A. Configuration and Transformations

We have a design space $\mathbb{S}$ that contains all possible architectural configurations. Each configuration is uniquely determined by the values of the variables shown in Table I. These variables include dimensions of the top-level mesh (X, Y), type of interconnect within a cluster ($\mathcal{I}$), and for every type of core i, the number of cores per cluster (N_i) and the sizes of the *L1* and *L2* caches ($L1_i$, $L2_i$). Table I shows the complete list of variables when the design space contains two types of cores.

Fig. 5 shows an example of configuration. Here, we assume that the *L3* cache is used to fill all the area that is left after placing the cores, private caches, and interconnect. For this reason, the size of the *L3* cache is computed using the area constraint and is not a variable.

The concept of configuration neighborhood has a key importance for metaheuristics. The neighborhood of some configuration $\mathcal{C}$ is defined using a set of rules to obtain neighbors from $\mathcal{C}$. We refer to these rules as transformations. More precisely, every transformation identifies one or several variables of $\mathcal{C}$ that are modified to create a neighbor.

As an example, consider the `IncX` transformation, which increments the X-dimension of the top-level mesh by one. If $\mathcal{C}$ is a configuration with a 4×3 mesh [as in Fig. 5(a)], then `IncX` defines a neighbor of $\mathcal{C}$, which is a 5×3 mesh with the same values for the other variables.

Selecting the neighborhood size is another important aspect of metaheuristics. Small neighborhoods may cause metaheuristics to get stuck in the local optima and lead to suboptimal solutions. Large neighborhoods may become the reason for slow advance through the design space and significantly increase the algorithm execution time. The neighborhood size for $\mathcal{C}$ is primarily determined by the cardinality of the transformation set. Hence, an important question to solve is to find a moderately sized set of transformations to balance performance and quality.

Three types of transformations are proposed for efficient exploration, as described below. To provide a visible example of

TABLE II
LIST OF TRANSFORMATIONS FOR TWO TYPES OF CORES

First-order		Second-order	
IncX	DecX	IncX-DecY	IncY-DecX
IncY	DecY	IncX-DecN$_1$	IncN$_1$-DecX
Inc$\mathcal{I}$	Dec$\mathcal{I}$	IncX-DecN$_2$	IncN$_2$-DecX
IncN$_1$	DecN$_1$	IncY-DecN$_1$	IncN$_1$-DecY
IncN$_2$	DecN$_2$	IncY-DecN$_2$	IncN$_2$-DecY
Inc$L1_1$	Dec$L1_1$	IncN$_1$-DecN$_2$	IncN$_2$-DecN$_1$
Inc$L1_2$	Dec$L1_2$	**Reclustering**	
Inc$L2_1$	Dec$L2_1$	RecIncX	RecDecX
Inc$L2_2$	Dec$L2_2$	RecIncY	RecDecY

the transformation set, Table II enumerates all transformations for two types of cores.

1) *First-Order Transformations:* First, we define a group of basic transformations, each affecting only one variable of the configuration. They are referred to as first-order transformations. With every variable V (from Table I) we associate a pair of transformations, IncV and DecV, that increase and decrease the value of V to the next available value in the domain, respectively. For example, if X is allowed to take any integer value between 2 and 10, then the IncX transformation applied to a 4×4 mesh [Fig. 6(a)] will produce a 5×4 mesh [Fig. 6(b)]. DecY applied to a 4×4 mesh will produce a 4×3 mesh [Fig. 6(c)]. The other variables remain unchanged.

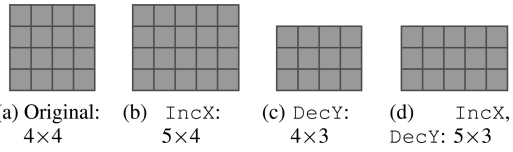
For the other variables, the next available values may be different. For example, the next available value for a memory of 128 KB can be 256 KB. The first-order transformations are summarized in the left column of Table II. Every type of core implies independent transformations. Thus, for two different types of cores, IncN$_1$ and IncN$_2$ are different transformations. If some transformation produces an illegal configuration, the latter is excluded from the search space.

2) *Second-Order Transformations:* First-order transformations still define small neighborhoods that may impede the search to escape from local optima. This fact is particularly important for highly constrained design spaces, when the penalty for violating a constraint is too high.[1]

Consider again the configuration with 4×4 mesh shown in Fig. 6(a). IncX produces a 5×4 mesh with 25% more area and a low probability to be accepted if the area penalty is high [Fig. 6(b)]. On the other hand, DecY yields a 4×3 mesh that has 25% less area [and hence less computing cores and cache, Fig. 6(c)]. It is not likely to be accepted either, since the performance of this configuration is significantly lower, compared to the original solution. However, if we apply both transformations simultaneously, we obtain a 5×3 mesh, as shown by Fig. 6(d). This solution has comparable area and performance, and hence higher probability to be accepted.

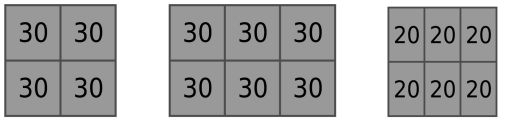
We refer to the transformations that perform a simultaneous update of two variables in the current configuration as second-order transformations. Rather than proposing transformations for all pairs of variables, we select a small group of variables, which we observe to enhance the neighborhood effectively. In particular, we create second-order transformations for the mesh dimensions (X and Y) and the number of cores of each type (N_i). For any pair of these variables, one is increased, while the other is decreased. An example of the IncX-DecY transformation was shown in Fig. 6(d). The right column of Table II enumerates the second-order group.

[1]The mechanism of penalty functions used for constraint modeling is explained in Section V-B.



(a) Original: 4×4 (b) IncX: 5×4 (c) DecY: 4×3 (d) IncX, DecY: 5×3

Fig. 6. Transformations applied to a 4×4 mesh.



(a) 2×2 mesh with 120 cores (b) 3×2 mesh with 180 cores (c) 3×2 mesh with 120 cores

Fig. 7. Reclustering of the 2×2 mesh.

3) *Reclustering:* Finally, we introduce the third type of transformation, reclustering, which decreases the search time notably, as it extends the neighborhood with promising configurations that were previously achievable only after a sequence of steps.

The idea of reclustering is illustrated in Fig. 7. Assume the original configuration is a 2×2 mesh with 30 cores per cluster, i.e., 120 cores in total [Fig. 7(a)]. The IncX transformation will result into a 3×2 mesh, as in Fig. 7(b). However, as the number of cores per cluster has not changed, the new configuration will have 180 cores with a 50% area increase. This is likely to cause unacceptable area (and power) penalties and prevent the exploration from ever escaping the 2×2 size neighborhood. To compensate, we propose adjusting the number of cores accordingly, so that the total number of cores remains constant. In the example, the new number of cores per cluster becomes $\lfloor \frac{30 \cdot 2}{3} \rfloor = 20$. Fig. 7(c) shows the new configuration. We refer to this procedure as reclustering, as it permits changing the number of clusters without causing strong penalties for any of the metrics.

There are four reclustering transformations, which are defined by either increasing, or decreasing any of the mesh dimensions. They are listed in the right column of Table II.

B. Exploration Problem

We consider architectural exploration as a constrained optimization problem. In addition to optimizing the value of objective function we have to satisfy a number of constraints for other metrics. For simplicity, in this paper, we focus on the following formulation: maximize the overall chip performance (IPC), subject to the resource budget, i.e., area and power constraints. The methodology described here applies to other formulations of the exploration problem and also allows the incorporation of additional metrics.

To enforce the satisfaction of constraints we use penalty functions. For every constrained metric, the objective is penalized once the metric value exceeds the constraint value.

Let us consider an area constraint indicating that the total area cannot exceed the value MaxArea. We define the relative excess of *Area* as

$$Ex(Area) = \frac{\max(0, Area - MaxArea)}{MaxArea}. \quad (15)$$

The excess is zero when the area constraint is satisfied. Otherwise, it gives a relative degree of area violation. Next, we define the area penalty in the objective function

$$Pen(Area) = \frac{1}{1 + \mu \cdot Ex(Area)} \tag{16}$$

where μ is a penalty factor that grows as exploration advances. This decreases the probability of accepting infeasible configurations as the exploration evolves toward the final solution. Now, consider the objective function in the form

$$Obj = \texttt{IPC} \cdot Pen(Area).$$

If the area constraint holds ($Area \leq MaxArea$), then $Pen(Area) = 1$ and the objective is equal to the IPC of configuration. If $Area > MaxArea$, then $Pen(Area) < 1$ and the IPC value in the objective function is degraded.

Penalty functions in the form of (16) allow adjusting the objective of infeasible configurations with respect to the violation degree. This mechanism makes the design space smoother and allows leaving the feasibility region temporarily, rather than stopping at its boundaries. When the exploration is out of the feasibility region, the penalty grows with the distance to the region, thus forcing the exploration to progressively return to the feasibility region.

The complete objective function used in this paper contains the penalty terms for area, power, and aspect ratio of the top-level mesh, AR. These terms are defined similar to (16), so the final equation is

$$Obj = \texttt{IPC} \cdot Pen(Area) \cdot Pen(Power) \cdot Pen(AR). \tag{17}$$

C. Exploration With Simulated Annealing

The SA algorithm is outlined in procedure 1. It starts with some initial configuration, that can be chosen randomly. We typically assign the lowest feasible value for every variable of configuration to prevent the constraints from being violated.

The algorithm implements a conventional annealing schedule. Given an initial temperature T_{init} and cooling factor $\alpha < 1$, a new configuration (*NewC*) is generated (lines 1–1). It may be accepted probabilistically, depending on the current temperature T_{cur} (line 1). The value of T_{cur} decreases as the system evolves in time (line 1). Penalization weight μ grows in time to decrease the probability of accepting infeasible solutions (line 1). For every temperature, a number of moves that depends on the size of the problem (P, the number of transformations) is generated.

NewC is obtained by selecting some transformation t_i and applying it to *CurC*. The probability distribution for the selection of t_i can be specified by the user. In this paper, we used uniform probabilities for all t_i and selected the transformation to be applied at every iteration randomly.

The new configuration is accepted probabilistically according to Procedure 2. Configurations with better objective value [calculated using (17)] are always accepted (line 2), other configurations are accepted with probability P_a

$$P_a = e^{-\gamma}, \quad \gamma = \frac{\text{Obj}(CurC)}{\text{Obj}(NewC)} \cdot \frac{1}{T_{cur}}. \tag{18}$$

This probability depends on two exponent factors. The former avoids the acceptance of solutions with high degradation of the objective function. The latter reduces the probability of hill climbing as the temperature cools down. To find the value for T_{init}, we apply a common strategy to calculate the average cost variance and use (18) to obtain a high initial acceptance probability, such as 0.95 [46].

Procedure 1 SIMULATEDANNEALING

1: *CurC* ← *BestC* ← "Some initial solution"
2: $T_{cur} = T_{init}$
3: **while** improvement in last k iterations **do**
4: **for** P iterations **do**
5: select t_i randomly with uniform probability
6: generate *NewC* by applying t_i to *CurC*
7: **if** ACCEPT(*CurC*, *NewC*) **then** *CurC* ← *NewC*
8: **if** Obj(*CurC*) > Obj(*BestC*) **and** FEASIBLE(*CurC*) **then** *BestC* ← *CurC*
9: **end for**
10: $T_{cur} = \alpha \cdot T_{cur}$
11: $\mu = \mu/\alpha$
12: **end while**
13: **return** *BestC*

Procedure 2 ACCEPT(*CurC*, *NewC*)

1: **if** Obj(*NewC*) > Obj(*CurC*) **then return** true
2: **else return** true with probability P_a, defined by (18)

Note that the best solution is updated only when the *CurC* is feasible as given by the FEASIBLE(*CurC*) procedure in line 1. This procedure returns true if and only if *CurC* satisfies all constraints (for area, power, and aspect ratio in our case).

D. Exploration With Extremal Optimization

Extremal optimization is inspired by the principle of evolution in ecosystems, which were observed to evolve by selecting against its worst components (or features). We draw here an analogy with the exploration problem, by considering every configuration to be determined by the conjunction of its variables, $\mathcal{P}$, similarly to the components (features) of the ecosystem.

Given a configuration, EO evaluates the fitness of its variables by comparing the current objective value with that of the neighbors, defined by the transformation set. A variable is well fit if the configuration objective cannot improve significantly by applying any of the transformations changing this variable. EO focuses on improving the status of variables with low fitness.

Since there are transformations that update several variables simultaneously, it is more convenient to work with the transformation fitness. As opposed to variables, well-fit transformations are those that cause higher improvement of the objective, when applied. More precisely, given the current configuration *CurC*, the fitness of transformation t_i is

$$\Phi = Obj(NewC) \tag{19}$$

where *NewC* is the configuration obtained by applying t_i to *CurC*. To maximize the objective function, at every iteration EO algorithm selects a transformation with high fitness and applies it to the current configuration.

Local optima are avoided by randomizing the selection process. The transformations are ranked according to their fitness in descending order (the best transformations have

Procedure 3 EXTREMALOPTIMIZATION

1: $CurC \leftarrow BestC \leftarrow$ "Some initial solution"
2: **while** some improvement in the last k iterations **do**
3: *local search:* evaluate P randomly selected transformations and accept only those that improve the Obj($CurC$);
4: sort all transformations in descending order of Φ;
5: select t_i according to equation (20);
6: apply t_i to $CurC$;
7: **if** Obj($CurC$) > Obj($BestC$) **and** FEASIBLE($CurC$) **then** $BestC \leftarrow CurC$;
8: $\mu = \mu \cdot \beta^{\tau}$;
9: **end while**
10: **return** $BestC$

lower indices in the rank). The transformations are randomly selected by some probability distribution biased toward the ones with highest fitness values. The power-law distribution is a typical one for EO. For example, if the system has N transformations ranked from 1 to N in descending order of their fitness, the index i of the selected transformation can be calculated as follows:

$$i = \lceil N \cdot p^{\tau} \rceil \tag{20}$$

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

The EO algorithm is described in Procedure 3. In this paper, we use a variation of EO called continuous extremal optimization [48]. This variant combines EO with a local search at the beginning of each iteration, contributing to improve the objective value of the final solution and the speed of the algorithm.

To select a transformation t_i, all transformations are sorted according to their fitness value (19). The power-law described by (20) is used to randomize the selection. Finally, t_i is accepted unconditionally and *BestC* is updated if the objective is better than any other configuration visited so far. Penalization weight μ increases as the system evolves, as in the SA algorithm. Since EO does not have the notion of temperature, we select the following law to update μ (line 3), where β is a constant, slightly greater than one (such as 1.01). The selection of β has to assure that μ grows slow enough for an efficient exploration, which implies a periodical acceptance of infeasible configurations.

VI. EXPERIMENTAL RESULTS

This section presents the experiments performed to validate the contributions of this paper. We first start by describing the simulation environment that has been used to compare with the results obtained by the models.

A. Simulation Environment

A cycle-accurate simulator for hierarchical interconnect networks has been designed [49]. It uses BookSim 2.0 [33] as underlying infrastructure. The simulator can model the contention of the interconnect network at flit level.

Three enhancements were made to BookSim. First, the probabilistic traffic injection patterns were replaced by state-machine models for cores, caches, and memory controllers. The cores inject memory requests according to parameters characterizing the average workload of the system. Cores are stalled when they are waiting for responses from memory. Both in-order and out-of-order cores can be modeled. Memories accept requests from cores and send replies after a predefined latency.

Support for hierarchical topologies was added, thus enabling the simulation of multilevel interconnect networks with an arbitrary number of levels. Finally, models for bus and multibus topologies were also created.

Each simulation was run long enough to obtain a 2% relative error (the same value used for the analytical model) with a 95% confidence degree. The 95% confidence interval is guaranteed using the batch means method [50].

B. Validation of the Analytical Model

This section describes the experiments used to validate the analytical performance model for CMPs. The experiments demonstrating the efficiency of fixed-point iteration and bisection methods for resolving the cyclic dependency have been extensively discussed in [10].

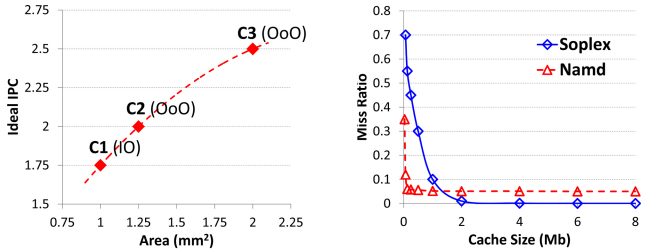
To validate the quality of the model in estimating performance, we carry out a CMP design space exploration experiment. For every configuration of the design space, we obtain the throughput using both analytical modeling and simulation. The search space in this experiment is made intentionally small in order to allow an exhaustive simulation of all configurations. With this experiment, we demonstrate that the analytical model selects a set of best-throughput architectures very similar to the simulator, but in much shorter time.

The parameters of the exploration space $\mathbb{S}_1$ for this experiment are listed in Table III. The value chosen for chip area is typical for CMP exploration studies [8], [18]. The estimates for the area of each component are derived from the parameters of the Intel Core 2 Duo E6400 processor [41]. We scale the core area and memory density down to 16 nm to allow hundreds of cores within the chip area. Table IV shows the area-performance model for cache memory.

In the studies of this section we consider three types of cores. Fig. 8(a) plots the ideal performance of every type of core (*C1*, *C2*, and *C3*) as a function of the core area. The parameters for *C2* are obtained from [38], whereas the parameters for *C1* and *C3* are generated by applying Pollack's rule [51] to the parameters of *C2*. We also assume that the smallest core, *C1*, is in-order (IO), while the other two are out-of-order (OoO). For this particular experiment, only the *C2* cores are considered (the other cores will be used in subsequent experiments).

Without loss of generality, we select two applications for the workload model, namely, *soplex* and *namd* from SPEC CPU2006. Fig. 8(b) depicts the miss ratio produced by these applications as a function of cache size, and emphasizes the difference in their cache requirements. We assume that only one type of application is run by all cores simultaneously and apply the strategy described in Section III-A to select configurations optimizing the throughput for both applications. The weights in the aggregate cost function (1) are $a_i = 0.5$, giving equal priority to both workloads.

A simple abstraction of a cache coherency protocol is considered, under the following assumptions. There are three classes of messages: requests, replies, and acks, which are sent through dedicated physical subnetworks to favor traffic



(a) (b)

Fig. 8. Exploration setup. (a) Core types. (b) Cache miss model of *soplex* and *namd* applications.

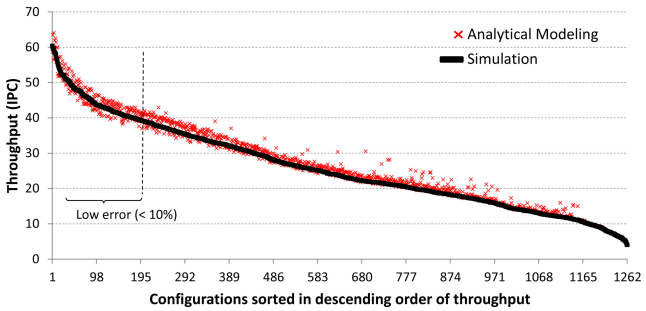


Fig. 9. Throughput comparison for analytical model and simulation.

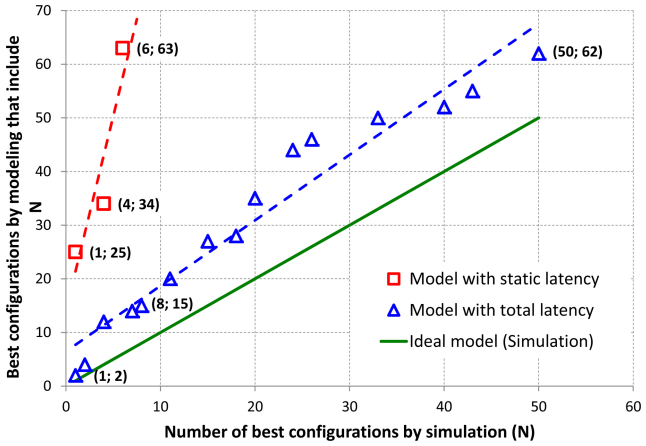


Fig. 10. Order comparison for analytical model and simulation.

separation [1]. The *L3* directory is distributed and cores have to access the *L3* cache to retrieve data, in case of a hit, or to redirect their requests to *MC*, otherwise. The invalidation flows are not modeled, since their traffic is assumed to be minor. Finally, the cache model is inclusive, so any miss generates write-back traffic to the next cache level or *MC*.

The number of cores and cache sizes are varied to explore the trade off between computing units and on-chip memory. Three types of local interconnects are considered inside the clusters: buses, uni-, and bi-directional rings. The exploration of the mesh dimensions compromises the number of clusters and processors per cluster. The maximum power of all configurations is limited to 200 W.

Given these parameters, our framework exhaustively generates all feasible configurations in $\mathbb{S}_1$, producing 1262 configurations in total. The simulation of all configurations takes about 22 h, while the analytical model takes 62 s, delivering more than a 1200× speedup. The best configuration obtained by simulation has a throughput of 60.41 IPC. This architecture has a 4×3 mesh (12 clusters with uni-ring interconnect, eight cores per cluster), a total of 96 cores with 128 Kb *L1*, 1 Mb *L2* private caches, and 108 Mb shared *L3* cache.

In Fig. 9, we sort the configurations by throughput along the horizontal axis, as estimated by simulation. One can see that the analytical model tracks the simulation curve with reasonable accuracy. The analytical model underestimates contention and, for this reason, the discrepancy with simulation increases with higher values of contention. Configurations with similar throughput may have different contention values, hence the noisy behavior of the modeling curve. The average absolute error in throughput is 4.3%, which corresponds to the error reported by the latency model [24].

The worst-case error for all nearly-optimal configurations does not exceed 10% (low-error zone), although it grows as we move away from the optimum. This is explained by the fact that architectures with balanced interconnects tend to have higher throughput and less contention. The precision of the analytical model drops when the contention increases, hence the error grows for configurations which are far from the optimum. Since the design exploration problem is aimed at selecting configurations with the highest throughput, this loss of precision is not critical.

What really matters for exploration are the relative, rather than the absolute values of throughput. When exploring a huge design space we would like to discard suboptimal architectures and keep a moderate subset of promising solutions. These configurations can be further simulated to select the best one. Hence, we are interested in comparing the order of configurations by the highest throughput, as delivered by the analytical model and the simulation. Here is where our technique demonstrates very accurate results: Fig. 10 shows the comparison for the best-throughput order. To make the picture illustrative, we only consider the 50 best configurations, although this tendency is maintained for the whole set of configurations. The horizontal axis specifies the number N of best configurations chosen by simulation. The vertical axis indicates the minimum number of best configurations chosen by the analytical model that include the N best ones by simulation. For example, the point with coordinates (1;2) means that the best configuration by simulation is the second best by modeling. The rightmost point on the plot (50;62) means that the 62 best configurations by modeling include the 50 best by simulation. This is actually a very accurate result for the analytical model, considering that more than a thousand of configurations are compared.

We also demonstrate that estimations based on static latency deliver a deficient order. It biases the exploration toward configurations with large bus-based clusters, given the fact that the long contention latency in the buses is not considered. The point (1;25) in Fig. 10 means that the best configuration is on the 25th position when not considering contention.

C. Exploration With Metaheuristics

In this section, we present the experiments used to validate the quality and efficiency of metaheuristics. For this purpose, we consider a substantially larger search space $\mathbb{S}_2$, containing about $1.5 \cdot 10^9$ configurations. The parameters of $\mathbb{S}_2$ as well as its comparison with $\mathbb{S}_1$ are given in Table III.

TABLE III
PARAMETERS OF THE EXPLORATION SPACES $\mathbb{S}_1$ AND $\mathbb{S}_2$

Parameter	$\mathbb{S}_1$ values	$\mathbb{S}_2$ values
Core types	*C2*	*C1*, *C2*, *C3*
L1 size	64, 128 Kb	64, 96, 128 Kb
L2 size	128 Kb to 1 Mb	64 Kb 1 Mb
Mesh dimensions	2×2 to 10×10	2×2 to 16×16
Chip area	350 mm^2	
Memory density	1 mm^2 / Mb	
Off-chip memory latency	100 cycles	
Type of cluster interconnect	Bus, uni-ring, bi-ring	
Interconnect link width	256 bits	
Workload `MPI`	0.25	
Workload `MLP`	1.25	

TABLE IV
CACHE AREA-PERFORMANCE MODEL

Cache size	64K	128K	256K	512K	1M	2M	4M	8M
Area (*mm*^{2})	0.063	0.125	0.25	0.5	1.0	2.0	4.0	8.0
Latency (*cycles*)	2	3	4	5	6	7	8	9

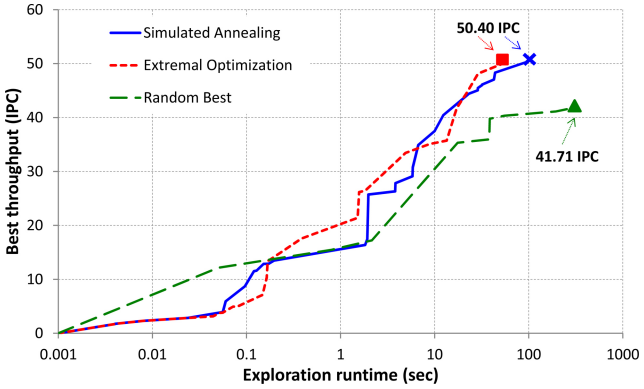


Fig. 11. Evolution of best discovered throughput in time.

1) *Evolution of Search in Time:* Fig. 11 shows the evolution of the best throughput discovered by the heuristics as the exploration evolves. To emphasize the need for intelligent search, we compare SA and EO with a naive random best (RB) strategy. RB simply generates random solutions, without tracking any history, keeping only the best known result.

The exploration is performed with a maximum power $P_{\max} = 180W$, which was found to be a reasonable value to explore the trade off between throughput and power. The metaheuristic parameters are $\alpha = 0.995$ and $\tau = 1.6$.

One can observe that SA discovers the optimum,[2] 50.40 `IPC`, in about 100 s (total time ≈ 160 s). EO reaches the optimum in just 70 s (total time ≈ 130 s). However, for RB it takes almost 300 s to find a configuration with 41.71 `IPC`, 21% worse than the optimum. The exhaustive exploration of all configurations in $\mathbb{S}_2$ would have taken more than 100 days using a single-core machine. These facts justify the importance of the metaheuristics in the exploration.

2) *Power-Performance Exploration:* This study shows how our model can be used to explore power-performance trade offs. Fig. 12 depicts configurations with best throughput (x-axis) discovered by metaheuristics with different power budgets (y-axis). We start with the power limit that delivers highest performance (80.73 `IPC` at 300 W) and reduce the budget in amounts of 20 W until we reach 140 W. This plot illustrates the expected decrease in performance for configurations with lower power budgets.

[2]We have computed the optimum by exhaustively running the model for all configurations in $\mathbb{S}_2$. A high-performance computing cluster was used, which effectively reduced the exploration time to five days.

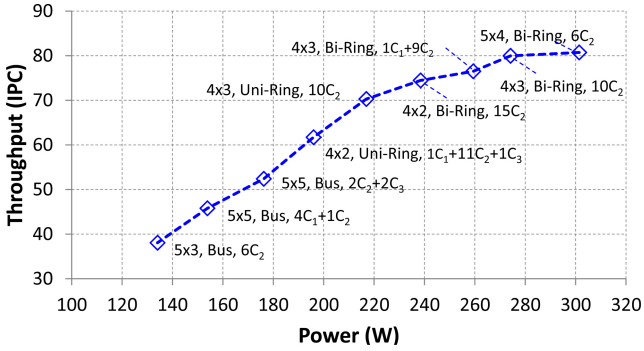


Fig. 12. Power-performance trade off in $\mathbb{S}_2$.

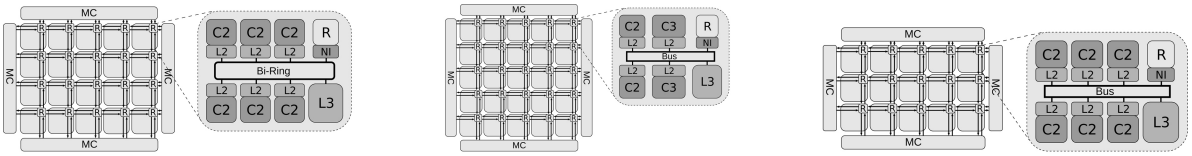
The configuration with highest performance (80.73 `IPC`) has a power of 300 W. The block diagram of this configuration is shown in Fig. 13(a) and represents a 5×4 mesh with six *C2* cores and bi-ring interconnect in clusters. As we reduce the power budget, the mesh dimensions are reduced, decreasing the number of high-performance but power-hungry routers. The clusters incorporate more cores, resulting into higher latencies of memory access. Therefore, performance of configurations decreases. Note that the ring-based interconnects are preferred for budgets > 180 W.

Bi-rings deliver the best performance, however they are less power-efficient when compared to buses. Hence, buses replace rings as soon as the power budget drops significantly (below 180 W). The layout of the first configuration with a bus, at 180 W limit, is shown in Fig. 13(b). It has a 5×5 mesh, two *C2* and two *C3* cores per cluster. With further decrease in the budget, power is saved by either reducing the number of cores, or by selecting more power-efficient cores (e.g., *C1*). Thus, the configuration for 140 W limit contains 15 bus-based clusters with six cores, totaling 90 cores per chip [Fig. 13(c)].

3) *Comparison With Simulation:* The main question this section tries to answer is: how can we check that the accuracy of the results given by metaheuristics is acceptable for a huge search space? To answer this question, one would need to simulate all configurations exhaustively. This task is intractable due to enormous computational cost. What we propose is to run the exploration and store the n best configurations discovered by the search (n is a parameter). Afterward, we simulate those n configurations and check whether the best configuration retains its rank after simulation.

In this experiment, we run the EO algorithm with $\tau = 1.6$. As the search evolves, a set of $n = 100$ best configurations is maintained. Upon the algorithm termination these configurations are simulated. The dashed line in Fig. 14 shows the throughput values in descending order, as calculated by the analytical model, with the maximum being 80.73 `IPC` (#1) and the minimum 71.35 `IPC` (#100). Then, all 100 configurations are simulated, and for each one the throughput obtained by simulation is plotted (solid line in Fig. 14).

The best configuration by simulation is #3, with a throughput of 81.87 `IPC`. Although this configuration is assigned the



(a) No power constraint (300*W*) (b) 180*W* (c) 140*W*

Fig. 13. Layouts of best configurations, discovered by exploration for some power budgets.

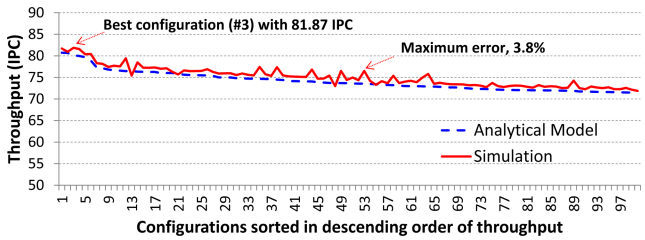


Fig. 14. Throughput of 100 best configurations.

third place in the order created by model, the difference in throughput of #1 and #3 is less than 2%. The maximum error between modeling and simulation is within 3.8%. Certain deviations in this experiment are inevitable due to the simplifying assumptions of the analytical model. However, for the majority of configurations the tendency for the throughput is to decrease as the rank id increases, indicating a good correlation between the simulator and the model.

VII. Conclusion

Analytical models become essential for the characterization of CMP interconnects within vast design spaces of architectural parameters. This paper shows that such models need to incorporate the contention factor in order to adequately estimate performance. We have presented an analytical method to model the contention of hierarchical interconnects, by resolving the cyclic dependency between memory latency and traffic. Furthermore, we have shown how to efficiently reduce the number of configurations considered during the exploration by using metaheuristics for the combinatorial optimization. Both contributions are indispensable to make the architectural design space exploration of future hundred- and thousand-core CMPs tractable. The validity and efficiency of the proposed methodology were proved through extensive simulation and with the examples of architectural exploration.

Acknowledgments

The authors would like to thank J. Carmona, F. Guim, M. Kishinevsky, and U. Ogras for insightful comments and helpful discussions.

References

[1] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, B. Liewei, J. Brown, M. Mattina, M. Chyi-Chang, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook, "TILE64—Processor: A 64-core SoC with mesh interconnect," in *Proc. Solid State Circuits*, Feb. 2008, pp. 88–98.

[2] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," *Proc. IEEE*, vol. 96, no. 5, pp. 879–899, May 2008.

[3] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, L. Jae-Wook, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, "The Raw microprocessor: A computational fabric for software circuits and general-purpose programs," *IEEE Micro*, vol. 22, no. 2, pp. 25–35, Mar. 2002.

[4] J. Balfour and W. J. Dally, "Design tradeoffs for tiled CMP on-chip networks," in *Proc. Intl. Conf. Supercomputing*, 2006, pp. 187–198.

[5] R. Das, S. Eachempati, A. Mishra, V. Narayanan, and C. Das, "Design and evaluation of a hierarchical on-chip interconnect for next-generation CMPs," in *Proc. High Performance Comput. Arch.*, Feb. 2009, pp. 175–186.

[6] S. Kang and R. Kumar, "Magellan: A search and machine learning-based framework for fast multi-core design space exploration and optimization," in *Proc. Design Autom. Test Eur.*, 2008, pp. 1432–1437.

[7] D. Sheldon, F. Vahid, and S. Lonardi, "Interactive presentation: Soft-core processor customization using the design of experiments paradigm," in *Proc. Design Autom. Test Eur.*, 2007, pp. 821–826.

[8] T. Oh, H. Lee, K. Lee, and S. Cho, "An analytical model to study optimal area breakdown between cores and caches in a chip multiprocessor," in *Proc. ISVLSI*, May 2009, pp. 181–186.

[9] A. Cassidy, K. Yu, H. Zhou, and A. Andreou, "A high-level analytical model for application specific CMP design exploration," in *Proc. Design Autom. Test Eur.*, Mar. 2011, pp. 1–6.

[10] N. Nikitin, J. de San Pedro, J. Carmona, and J. Cortadella, "Analytical performance modeling of hierarchical interconnect fabrics," in *Proc. Int. Symp. Netw. Chip*, May 2012, pp. 107–114.

[11] P. Bogdan and R. Marculescu, "Non-stationary traffic analysis and its implications on multicore platform design," *Comput. Aided Design Integr. Circuits Syst.*, vol. 30, pp. 508–519, Apr. 2011.

[12] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[13] S. Boettcher and A. G. Percus, "Extremal optimization: Methods derived from co-evolution," in *Proc. Genetic Evol. Comput. Conf.*, 1999, pp. 825–832.

[14] J. Huh, D. Burger, and S. W. Keckler, "Exploring the design space of future CMPs," in *Proc. Int. Conf. Parallel Arch. Compilation Tech.*, 2001, pp. 199–210.

[15] Y. Li, B. Lee, D. Brooks, Z. Hu, and K. Skadron, "CMP design space exploration subject to physical constraints," in *Proc. High Performance Comput. Arch.*, Feb. 2006, pp. 17–28.

[16] M. Monchiero, R. Canal, and A. Gonzalez, "Power/performance/thermal design-space exploration for multicore architectures," *Parallel Distrib. Syst.*, vol. 19, no. 5, pp. 666–681, May 2008.

[17] E. İpek, S. A. McKee, R. Caruana, B. R. de Supinski, and M. Schulz, "Efficiently exploring architectural design spaces via predictive modeling," *SIGARCH Comput. Arch.*, vol. 34, no. 5, pp. 195–206, Oct. 2006.

[18] A. S. Cassidy and A. G. Andreou, "Beyond Amdahl's law: An objective function that links multiprocessor performance gains to delay and energy," *IEEE Trans. Comp.*, vol. 61, no. 8, pp. 1110–1126, Aug. 2012.

[19] R. E. Matick, T. J. Heller, and M. Ignatowski, "Analytical analysis of finite cache penalty and cycles per instruction of a multiprocessor memory hierarchy using miss rates and queuing theory," *IBM J. Res. Dev.*, vol. 45, pp. 819–842, Nov. 2001.

[20] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. Int. Symp. Microarch.*, 2009, pp. 469–480.

[21] M. Papamichael, J. Hoe, and O. Mutlu, "FIST: A fast, lightweight, FPGA-friendly packet latency estimator for NoC modeling in full-system simulations," in *Proc. ACM/IEEE Int. Symp. Netw. Chip*, May 2011, pp. 137–144.

[22] J. Navaridas, B. Khan, S. Khan, P. Faraboschi, and M. Lujan, "Reservation-based network-on-chip timing models for large-scale architectural simulation," in *Proc. ACM/IEEE Int. Symp. Netw. Chip*, May 2012, pp. 91–98.

[23] S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith, "A mechanistic performance model for superscalar out-of-order processors," *ACM Trans. Comput. Syst.*, vol. 27, pp. 1–37, May 2009.
[24] U. Ogras, P. Bogdan, and R. Marculescu, "An analytical approach for network-on-chip performance analysis," *Comput. Aided Design Integr. Circuits Syst.*, vol. 29, pp. 2001–2013, Dec. 2010.
[25] Y. Ben-Itzhak, I. Cidon, and A. Kolodny, "Delay analysis of wormhole based heterogeneous NoC," in *Proc. NOCS*, May 2011, pp. 161–168.
[26] S. Foroutan, Y. Thonnart, R. Hersemeule, and A. Jerraya, "An analytical method for evaluating network-on-chip performance," in *Proc. Design Autom. Test Eur.*, Mar. 2010, pp. 1629–1632.
[27] P. Bogdan and R. Marculescu, "Statistical physics approaches for network-on-chip traffic characterization," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synthesis*, 2009, pp. 461–470.
[28] 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 Trans. Comput. Aided Design*, vol. 28, no. 1, pp. 3–21, Jan. 2009.
[29] R. Kumar, D. M. Tullsen, and N. P. Jouppi, "Core architecture optimization for heterogeneous chip multiprocessors," in *Proc. Int. Conf. Parallel Architectures Compilation Tech.*, 2006, pp. 23–32.
[30] E. S. Chung, P. A. Milder, J. C. Hoe, and K. Mai, "Single-chip heterogeneous computing: Does the future include custom logic, FPGAs, and GPGPUs?" in *Proc. Int. Symp. Microarch.*, 2010, pp. 225–236.
[31] V. Saripalli, G. Sun, A. Mishra, Y. Xie, S. Datta, and V. Narayanan, "Exploiting heterogeneity for energy efficiency in chip multiprocessors," *J. Emerging Select. Topics Circuits Syst.*, vol. 1, no. 2, pp. 109–119, Jun. 2011.
[32] A. Hartstein, V. Srinivasan, T. Puzak, and P. Emma, "On the nature of cache miss behavior: Is it square root of 2," *J. Instruction Level Parallelism*, vol. 10, pp. 1–22, Jun. 2008.
[33] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Mateo, CA, USA: Morgan Kaufmann, 2003.
[34] A. R. Alameldeen, "Using compression to improve chip multiprocessor performance," Ph.D. dissertation, Dept. Comp. Sci., Univ. Wisconsin, Madison, WI, USA, 2006.
[35] L. Kleinrock, *Queueing Systems*, vol. 1. New York, NY, USA: Wiley, 1975.
[36] J. L. Hennessy and D. A. Patterson, *Computer Architecture, 4th Edition: A Quantitative Approach*. San Mateo, CA, USA: Morgan Kaufmann, 2006.
[37] Y. Chou, B. Fahs, and S. Abraham, "Microarchitecture optimizations for exploiting memory-level parallelism," in *Proc. ISCA*, 2004, pp. 76–87.
[38] T. K. Prakash and L. Peng, "Performance characterization of SPEC CPU2006 on Intel Core 2 Duo processor," in *Proc. ISAST*, 2008, pp. 36–41.
[39] CACTI. *An integrated cache and memory access time, cycle time, area, leakage, and dynamic power model* [Online]. Available: http://www.hpl.hp.com/research/cacti/
[40] A. Kahng, B. Li, L.-S. Peh, and K. Samadi, "Orion 2.0: A fast and accurate NoC power and area model for early-stage design space exploration," in *Proc. Design Autom. Test Eur.*, Apr. 2009, pp. 423–428.
[41] A. Danowitz, K. Kelley, J. Mao, J. P. Stevenson, and M. Horowitz. (2012, Apr.). *Parameters of Intel Core 2 Duo E6400 Processor at CPU Database* [Online]. Available: http://cpudb.stanford.edu/processors/1088
[42] R. Burden and D. Faires, *Numerical Analysis*. Pacific Grove, CA, USA: Brooks Cole, 2010.
[43] MATLAB. *The Language of Technical Computing* [Online]. Available: http://www.mathworks.com
[44] D. P. Bertsekas, *Nonlinear Programming*, 2nd ed. Belmont, MA, USA: Athena Scientific, Sep. 1999.
[45] G. Wood, "The bisection method in higher dimensions," *Math. Program.*, vol. 55, Jun. 1992.
[46] D. F. Wong, H. W. Leong, and C. L. Liu, *Simulated Annealing for VLSI Design*. Norwell, MA, USA: Kluwer Academic Publishers, 1988.
[47] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Berlin, Germany: Springer, 2003.
[48] 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, p. 016702, Jul. 2005.
[49] J. de San Pedro, "A simulation framework for hierarchical network-on-chip systems," Master's thesis, Dept. Software, Univ. Politècnica de Catalunya, Barcelona, Spain, 2012.
[50] G. S. Fishman, "Grouping observations in digital simulation," *Manage. Sci.*, vol. 24, no. 5, pp. 510–521, Jan. 1978.
[51] F. J. Pollack, "New microarchitecture challenges in the coming generations of CMOS process technologies," in *Proc. IEEE Micro*, 1999, p. 2.

Nikita Nikitin received the B.S. and M.S. degrees in computer science from Moscow Institute of Physics and Technology, Moscow, Russia, in 2005 and 2007, respectively. He received the Ph.D. degree in computer engineering from the Universitat Politècnica de Catalunya, Barcelona, Spain, in 2013.

He is currently a Researcher with the Department of Computer and Information Science, Norwegian University of Science and Technology, Trondheim, Norway. His current research interests include system-level design exploration, synthesis, and physical-aware optimization of many-core on-chip systems.

Javier de San Pedro received the M.S. degree in computer science from the Universitat Politècnica de Catalunya, Barcelona, Spain, in 2012, where he is currently pursuing the Ph.D. degree.

He previously focused on the simulation of chip multiprocessors and co-authored a few research papers on architectural exploration. His current research interests include the physical aspects of of large-scale chip multiprocessor design.

Jordi Cortadella (M'88) received the M.S. and Ph.D. degrees in computer science from the Universitat Politècnica de Catalunya, Barcelona, Spain, in 1985 and 1987, respectively.

He is currently a Professor with the Department of Software, Universitat Politècnica de Catalunya. In 1988, he was a Visiting Scholar with the University of California, Berkeley, CA, USA. He has co-authored numerous research papers and has been invited to present tutorials at various conferences. His current research interests include formal methods and computer-aided design of Very Large Scale Integration systems, with a special emphasis on asynchronous circuits, concurrent systems, and logic synthesis.

Dr. Cortadella has served on technical committees of several international conferences in the field of design automation and concurrent systems. He was a recipient of the Best Paper Awards at the International Symposium on Advanced Research in Asynchronous Circuits and Systems in 2004, the Design Automation Conference in 2004, and the International Conference on Application of Concurrency to System Design in 2009. In 2003, he was the recipient of a Distinction for the Promotion of the University Research by the Generalitat de Catalunya.