The Era of Many-Modules SoC: Revisiting the NoC Mapping Problem

Isask'har Walter[1] Israel Cidon[1] Avinoam Kolodny[1] Daniel Sigalov[2]

[1]Department of Electrical Engineering, [2]Department of Applied Mathematics
Technion – Israel Institute of Technology, Haifa 32000, Israel
{zigi@tx, cidon@ee, kolodny@ee, dansigal@tx}.technion.ac.il

ABSTRACT

Due to technology scaling, it is expected that future chips would integrate tens to hundreds of functional units. The growing power and design costs limit the benefit of continuously increasing the universality and complexity of these units and motivate the usage of specialized hardware modules. These modules are likely to be replicated in order to exploit the inherent parallelism of many tasks. This trend already exists in CMPs (moving from multi-core to many-cores), ASSPs and FPGAs.

In this paper, we revisit the network on-chip (NoC) mapping problem in light of this expected trend. Specifically, we leverage the use of on-chip replicated specialized modules to minimize traffic and hence to reduce the power consumed by the NoC. We further improve the interconnect efficiency by making the mapping algorithm aware of the set of modules traversed by application data. To this end, we present an enhanced modeling of the resources and timing requirements within a system on-chip (SoC). We evaluate the benefit of the proposed approach and show a significant reduction in the cost of communication.

Categories and Subject Descriptors

C.5.4 VLSI Systems

General Terms

Algorithms, Design, Performance

Keywords

System on-Chip, Network on-Chip, Mapping, Optimization

1. INTRODUCTION

The scaling of VLSI technology increases the number of transistors available to the system architect, making room for the implementation of hundreds of modules on a single chip in the near future. While it is hard to predict the specific applications that would benefit from such massive architectures, it is clear that the number of unique modules in such SoCs will be considerably smaller than the total number of modules. In order to allow a high degree of parallel processing and design productivity, units would be massively replicated on the chip, creating classes of identical modules with specialized functionality and characteristics. This trend complies with the advance of CMP (chip multi processor) systems, where processors and cache banks are already replicated many times all over the chip, while replication of DRAM controllers in a CMP is envisioned [**1**].

Due to the growing power and design costs of implementing increasingly complex, general purpose modules, the majority of the on-chip modules will be massively replicated specialized cores optimized for sets of tasks. Examples of such modules include DSPs, memory banks, hardware accelerators, graphic processors, vocoders, etc. Consequently, the data path in systems is likely to become more deeply pipelined in the future. In parallel to this trend, increasing system complexity has given rise to a packet-switching interconnection network, termed network on-chip (NoC). Due to its parallelism, scalability and modularity, a NoC reduces communication costs and improves design productivity.

As the distance traversed by packets in the network affects the energy consumption and the performance of the system, the mapping of the SoCs modules into the NoC structure is an important step in the design process. Intuitively, communicating modules should be placed in close proximity if they exchange large amounts of data (to save energy) or if they require a short network latency. In this paper, we discuss the mapping problem in future SoCs. Specifically, we account for classes of replicated modules and for multi-stage processing pipelines.

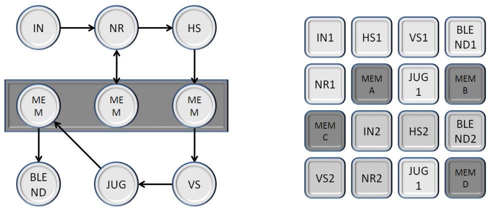
Typically, the inputs to a NoC mapping procedure are described by a demand matrix and a latency requirements matrix (or equivalent graph representations), which specify the flow bandwidth and the allowed delay between each pair of modules, respectively. In the classic demand matrix representation, each flow is bounded to a specific sender and receiver and no flexibility is allowed. Since the NoC infrastructure enables a sender to communicate with any other module, binding a flow to a particular pair of module instances is an unnecessary and expensive constraint. Alternatively, we may allow the mapping algorithm to benefit from the existence of identical modules, by assigning flows to the best replica within a module class. As flows are no longer associated with particular modules within a class, the mapping algorithm has a higher degree of freedom resulting in a more efficient design.

Moreover, we also argue that the commonly used latency requirements matrix limits the potential cost reduction in the mapping phase. By specifying only pair-wise module-to-module timing requirements, the application level data-flow is ignored, and the latency requirements are over-constrained. In this work, we introduce and evaluate an extended modeling of the SoC which captures the application's latency needs in a more appropriate way. In resemblance to retiming of a logic path in traditional circuit design, we allow trading off delays between flows that compose a stream of information processing: wherever applicable, we replace a "chain" of end-to-end delay constraints with a single, unified constraint, describing the latency requirement of an application stream, measured from the time the first module in the chain generates the data until the last module receives the data. By using this cross-layer knowledge of the application needs, the mapping algorithm exploits new degrees of freedom in the design space and the cost is reduced.

Figure 1a depicts a signal-flow graph of a video display of multimedia information chip [**2**]. Hardware processing of a single video stream requires six coprocessor units. In order to allow parallel processing of two video streams for a dual screen display ("application example II" in [**2**]), 12 coprocessors have to be mapped onto the chip (Figure 1b). Classically, the mapping algorithm would get as part of its input an explicit binding of a coprocessor with the memory it uses (e.g., NR1 to MEM_A, NR2 to MEM_D, and etc.). Moreover, a traditional formulation of this mapping problem would specify timing requirements for each of the 18 unidirectional flows within the application (nine for each of the parallel processed streams).

Our approach includes two fundamental modifications: the mapping phase groups the four available memory buffers into a single MEMx class, and finds the most efficient coprocessor-memory partition as part of the optimization process. In addition, as the overall performance of the application is dictated by the time a block of data traverses the whole chain of processing modules rather than by individual, single hop delays, the seven timing requirements for each stage are replaced by one end-to-end delay constraint. This reduces the total timing requirements for the system from 18 to 6 (as the single hop constraint between the NR and memory module cannot be replaced).

The rest of this paper is organized as follows: in section 2 we survey related work. In section 3 we describe the classic NoC traffic modeling, cost function and optimization algorithm. In section 4 we introduce our extended modeling and mapping schemes. In section 5 we present our cross-entropy optimization engine and in section 6 we evaluate the proposed schemes. In section 7 we conclude the paper.



(a) (b)

Figure 1: A dual-screen video display of multimedia SoC
(a) Signal-flow graph of a single video stream, describing nine individual point-to-point flows. Seven types of cores are involved: input processing (IN); noise reduction (NR); horizontal scaling (HS); vertical scaling (VS); address generating (JUG); and video blending (BLEND), and memory buffers (MEM). (b) A mapping of a dual screen SoC as a 4x4 tiled chip.

2. RELATED WORK

Due to its direct implications on overall system power and performance, the module mapping problem has received a considerable attention in the NoC literature. Since even simple formulations of the mapping problem are NP-hard, optimal mapping of large systems cannot be obtained within a reasonable amount of computation and heuristic optimization methods are often used. In the first paper to tackle mapping for NoC [**3**], a deterministic branch-and-bound algorithm is used to minimize the communication energy in the system. In [**4**], a heuristic algorithm and splitting of traffic facilitate the minimization of the average delay of packets in the network. In [**5**], the bandwidth requirements of the application and message dependencies are analyzed in order to find a mapping that reduces both power consumption and the application execution time. In [**6**], a multi-objective genetic algorithm is used to find a good tradeoff between power consumption and application execution time. Similarly, the authors of [**7**] suggest a random search method to minimize a cost function capturing the distance and the number of shortest paths between cores. While a linear-programming based approach is used in the core mapping phase described in [**8**] to solve a similar problem, the algorithms presented in [**9**] and [**10**] also address the resulting link load. Although providing powerful means to tackle power minimization in the mapping problem, these proposals either ignored the performance of the NoC or used the resulting performance as a quality measure rather than as a constraint for the mapping algorithm. In addition, the metrics used only account for the overall average performance, ignoring the actual, per-flow communication delay which is essential for SoC applications. Moreover, a strict association between the functionality of a module and its identity is assumed, so that the existence of replicated modules cannot be leveraged.

The earliest published work to consider energy efficient mapping of a bandwidth and latency constrained NoC is [**11**], where the authors describe a tabu-search based automated design process of a NoC providing quality-of-service guarantees. Another mapping scheme that uses delay constraints is described in [**12**]. There, a low complexity heuristic algorithm is used to map the cores onto the chip and then routing is selected so that all constraints are met. Similarly, the mapping schemes used in [**13**][**14**][**15**] all use the per-flow, source-destination latency requirements of the application as input to the design process while the end-to-end requirements imposed by the application are not considered.

An example of leveraging application-level constraints for NoC design is shown in [**16**], where end-to-end temporal requirements are used for buffer sizing. A specific problem of placing replicated memory controllers in the CMP domain was recently addressed in [**1**]. However, the placement targets bandwidth problems while the interconnect power consumption is not considered.

A related research field addresses the task mapping and scheduling problem. In a typical formulation of the problem, the application is described using a fine-grained task graph and the optimization algorithm aims at finding an efficient task-to-core (rather than core-to-tile) partitioning. In this work, we assume that task allocation has been performed prior to the tile-mapping phase and is provided as an input (e.g., [**7**][**8**][**9**][**10**][**11**]). However, ideas similar to the ones discussed here can be incorporated into the task mapping phase or into a joint task-core mapping algorithm. While the NoC literature is rich in task mapping algorithms, two papers are particularly relevant in this context. In [**17**], six NoC design problems (topology selection, task assignment, tile mapping, routing path allocation, task scheduling, and link speed assignment) are solved using a nested genetic algorithm optimization method. In [**18**], the authors propose a hardware-software co-synthesis algorithm that chooses a set of cores from an IP library, and then performs the tile mapping and task allocation so that total energy consumption is minimized. However, while focusing on the multi-phased optimization schemes, these papers do not directly address the potential benefit that lies in the replication of modules and in the extended formulation of timing requirements.

3. CLASSIC NOC MAPPING

Throughout this paper, we assume a mesh-grid NoC in which any shortest-path routing scheme is used (e.g. dimension ordered, YX-XY, etc.). Though this is a common design choice, the proposed techniques can be modified to accommodate other architectures and topologies, including non-regular ones. For simplicity, we also assume that all N modules have a similar shape and we focus on the mapping problem rather than the floorplanning phase. However, very similar concepts can be used during the floorplanning of the SoC to optimize the cost of the interconnect.

In this paper, a mapping $\pi \in P$ is the assignment of N tiled modules onto the chip, while P is the set of all $N!$ possible mappings.

3.1 Classic Traffic Characterization

The traffic within a SoC is traditionally described by a matrix of N^2 values, each of which specifies the amount of traffic transmitted between an ordered pair of modules:

$$B = \left\{ b_{i \to j} \mid 1 \le i, j \le N \right\} \tag{1}$$

where $b_{i \to j}$ is the bandwidth sent by module m_i to module m_j (composing flow $f_{i \to j}$), or 0 is such a flow does not exist.

As explained in Section 1, when timing requirements are considered in the mapping phase, they are classically listed as a set of pair-wise constraints. Since run-time, dynamic effects for congestion are hard to predict, hop-count is often used as a metric of latency (e.g., [**12**]). This approximation is accurate enough to be used by the mapping algorithm [**7**] as NoCs are typically designed to operate in light loads such that congestion effects are not dominant. However, other, more elaborate analytic delay models can be equally used. Formally written, the set of timing constraints is defined as:

$$T^{REQ} = \left\{ t_{i \to j}^{REQ} \mid 1 \le i, j \le N \right\} \tag{2}$$

where $t_{i \to j}^{REQ}$ is the timing requirement for the flow running from module m_i to module m_j, or ∞ if such a requirement does not exist. Given a module mapping π, classic mapping schemes consider it feasible if the following holds:

$$t_{i \to j} \le t_{i \to j}^{REQ}, \ \forall 1 \le i, j \le N \tag{3}$$

where $t_{i \to j}$ is the delay imposed on a flow $f_{i \to j}$ in mapping π.

Adhering to a widely used modeling of the cost of a given mapping (e.g., [**7**][**10**][**17**]), we use the following cost function:

$$Cost(\pi \in P) = \sum_{l \in L} BW_l = \sum_{1 \le i, j \le N} \left[b_{i \to j} \cdot Dist(i, j) \right] \tag{4}$$

where P is the set of all possible mappings of the modules to the chip; L is the set of all NoC links; BW_l is the bandwidth delivered over link l in mapping π; and $Dist(i,j)$ is the distance between module i and module j (measured in hop count) in mapping π. This expression directly captures the dynamic power consumed by the interconnect routers and links for the repeated charging of gate capacities and wires. Nevertheless, other cost functions (accounting for maximum channel load [**1**], static power consumption, etc.) may be used according to the designer needs.

3.2 Classic NoC Mapping

Given a set of bandwidth requirements (1) and timing constraints (2), the goal of traditional mapping algorithms is to find a mapping $\pi \in P$ that satisfies (3) and minimizes the cost (4). Intuitively, such mapping algorithms will try to map modules that exchange large amounts of traffic as close as possible to each other without violating any timing constraint.

As the number of all possible mappings $|P|$ prohibits a brute force search for the optimal mapping, an efficient solution must be found by examining only a small subset of the design space. This can be achieved by cleverly ruling out large sets of expensive solutions (e.g., [**3**]). Alternatively, heuristic algorithms are frequently used to solve the minimization problem. Popular choices include simulated annealing and genetic algorithms.

4. EXTENDED MAPPING ALGORITHM

4.1 Extended Traffic Specification

In the proposed extended traffic specification, the modules are divided into classes, so that modules in each class have identical functionality (e.g., a class of DSPs, hardware accelerators, memory banks etc.). Each unique (non-replicated) module forms a class of its own. Instead of using pair-wise traffic specifications (1), traffic is described by the amount of bandwidth each module m_i has to exchange with any of the modules in class C_j, as follows:

$$BW_i^k = \left\{ b_i \to C_k \mid 1 \le i \le N, 1 \le k \le |C| \right\} \tag{5}$$

where $b_i \to C_k$ is the amount of traffic generated by module m_i and destined for a module (or modules) of class k and C is set of all classes in the system.

Unlike classic mapping algorithms, the extended mapping algorithm also has to generate a match between sources and class destinations, specifying the amount of bandwidth exchanged between a source module and each module m_j in a class C_k:

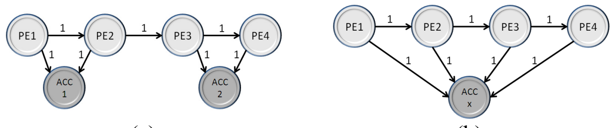
$$\left\{ b_{i \to j} \mid BW_i^k \ne 0, m_j \in C_k \right\} \tag{6}$$

When using the extended, class-based traffic specification, flows are no longer bound to any particular module within a class. If no further limitation is introduced, this might result in one or more modules being assigned multiple flows, offering them traffic beyond their saturation point. Therefore, the following set of constraints is introduced for any selected flow assignment:

$$\sum_{1 \le j \le N} b_{j \to i} \le \sigma_i, \ \forall 1 \le i \le N \tag{7}$$

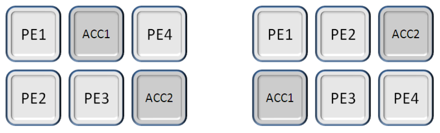
where σ_i is the maximal bandwidth module m_i can receive.

This relaxed specification of traffic requirement allows the mapping algorithm to match a source with any of the destinations within a class. Potentially, this reduces the cost of the NoC with respect to a mapping done with rigid, point-to-point traffic specifications. A simple example is illustrated in Figure 2: in this example, six modules should be mapped onto a 2x3 tiled chip. Four of the modules (PE1-4) have unique functionality while the other two (ACC1-2) are identical functional units (e.g., hardware accelerators). The communication in the application to be mapped is of a pipeline composed of the four processing elements PE1, PE2, PE3 and PE4. In addition, each of those modules needs to use one of the two accelerators available in the chip. For the sake of simplicity, we assume all flows have the same bandwidth requirement of 1 unit. A classic representation of the problem is depicted in Figure 2a, where PE1 and PE2 were arbitrarily chosen to use ACC1 while PE3 and PE4 would use ACC2. One optimal mapping is shown in Figure 3a, reflecting a minimal cost of 9, as defined in (4). Conversely, Figure 2b describes the same problem using the extended formulation: ACC1 and ACC2 are grouped together to a class called ACCx and the mapping algorithm is allowed to choose which processing element would use which accelerator. The resulting optimal mapping is shown in Figure 3b. Enabled by the flexibility introduced by the extended formulation, the mapping algorithm allocated ACC1 to PE1 and PE3, and ACC2 to PE2 and PE4. The cost of this mapping is only 7, reflecting a reduction of 20% with respect to the classic mapping.



(a) (b)

Figure 2: Classic (a) and extended (b) traffic specification



(a) (b)

Figure 3: Classic mapping (a) vs. extended mapping (b)

4.2 Extended Specification of Timing Requirements

In order to describe the timing requirements of the application, point-to-point constraints should be used only when absolutely necessary, as they impose rigid module proximity. By examining the flow of data in an application, we identify streams of information: a stream is composed of a chain of flows which can have a single, unified timing requirement. Formally written, a stream Ω is defined as follows[1]:

$$\Omega = \left\{ f_{m_{i_1}\to m_{i_2}}, f_{m_{i_2}\to m_{i_3}}, \ldots, f_{m_{i_{k-1}}\to m_{i_k}} \right\}$$
$$\mathrm{t}_{\Omega}^{REQ} = t_{m_{i_1}\to m_{i_2}}^{REQ} + t_{m_{i_2}\to m_{i_3}}^{REQ} + \ldots + t_{m_{i_{k-1}}\to m_{i_k}}^{REQ} \qquad (8)$$
$$\left\{ m_{i_1}, m_{i_2}, \ldots, m_{i_k} \right\} \in M$$

where m_{i0}, m_{i1},…,m_{ik} are the modules composing the chain; $f_{m_i \to m_j}$ is a flow of packets sent by module m_i to module m_j; and k is the largest number for which Ω is still a stream. Note that in case a flow is not a part of a longer chain of flows, it simply composes a stream of its own of length $k=1$.

In the proposed formulation, the set of timing requirements (2) is replaced with the set of requirements of all streams:

$$T_{\Omega}^{REQ} = \left\{ t_{\Omega_i}^{REQ} \mid \Omega_i \neq \phi \right\} \qquad (9)$$

The mapping π must meet all stream timing requirements:

$$t_{\Omega_i}^{\pi} \le t_{\Omega_i}^{REQ} \ , \ \forall t_{\Omega_i}^{REQ} \in T_{\Omega}^{REQ} \qquad (10)$$

where $t_{\Omega_i}^{\pi}$ is the delay imposed on a stream i in mapping π and $t_{\Omega_i}^{REQ}$ is its timing requirement.

Figure 4 and Figure 5 illustrate the benefit of using the extended traffic requirements: in this example, four modules (PE1-4) should be mapped onto a 2x2 chip. The application to be mapped is assumed to be implemented by a pipeline of all four modules and needs to have a total latency no larger than 4 time units (measured in hops). The pipeline also includes a bypass from PE2 to PE4 with a delay requirement of 1 time unit. Table 1 describes one of the possible formulations of this input data using the classic, point-to-point requirements, allowing a longer delay to the high bandwidth flow from PE1 to PE2. Table 2 describes the same problem in the extended formulation, only restricting the mapping algorithm by the total delay of the two data streams. Interestingly, this problem has no feasible solution using the classic mapping while it does have one when the extended formulation is used, depicted in Figure 5. As can be seen, placing PE2 two hops away from PE3 allows the timing requirement to be met; a step which was prohibited in the classic formulation. Analyzing the result, the mapping algorithm simply traded-off the long delay allowed between PE1 and PE2 with the delay between modules PE2 and PE3, leaving the total pipeline delay to be 4 time units.

Table 1

Flow	REQ
PE1→PE2	2
PE2→PE3	1
PE3→PE4	1
PE2→PE4	1

Table 2

Stream	REQ
PE1→PE2→PE3→PE4	4
PE2→PE4	1

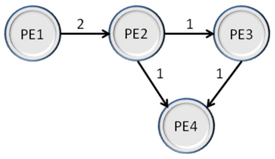


Figure 4: Bandwidth requirements

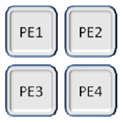


Figure 5: Resulting mapping

While this is a very simple problem which only involves a small number of modules and can be studied without using CAD tools, it sheds light on the challenges the architect faces when designing large scale systems. In such systems, the designer might only learn that there is no valid solution after a very long run of the mapping algorithm. As shown in Section 6, even if a solution can be found by the classic point-to-point tools, they are often more expensive than those generated by the proposed scheme.

4.3 Extended NoC Mapping

Similarly to the classic problem formulation, the proposed mapping algorithm aims at minimizing the power consumption by shortening the paths traversed by intensive data flows and placing pairs with low communication further apart, while meeting all timing requirements. However, due to the higher degree of freedom expressed in the extended formulation, it is able to find lower cost solutions.

Given a set of class bandwidth requirements (5), module capacities σ_i, and timing constraints (9), the goal of the extended mapping algorithm is to find a mapping $\pi \in P$ that satisfies (7) and (10) while minimizing the cost (4).

5. CROSS-ENTROPY OPTIMIZATION

Throughout this paper we use Cross-Entropy (CE) optimization [**19**] to find low cost mappings. CE is a modern optimization heuristic that was proven useful in many combinatorial optimization problems. The algorithm is akin to evolutionary algorithms such that a population of solutions is evolved. However, generation of new solutions is based on statistical methods of sampling and estimation of parameters. CE is inherently a global search method and therefore the risk of getting trapped in a local minimum is reduced. Extensive comparisons with solutions produced by simulated annealing (not shown here

[1] In this work, we consider the delay caused by the network distance and ignore other contributing factors (packetization/reassembly delays, module processing times, etc.) to allow a direct comparison with the classic scheme. However, if those latencies are known, they can be accounted for in both mapping schemes in a straight-forward manner.

due to space limitation) reveal that CE is superior in terms of speed and quality. This complies with the observation reported in [7] regarding the relatively poor results achieved by simulated annealing and other optimization methods (genetic algorithms and adaptive search procedures) when solving similar problems.

The goal is to find the global extremum of some cost function S(x) defined on an element space X. We assume that a global minimum is required, and that X is a discrete space of a finite (yet very large) number of elements such that exhaustive search is infeasible. It is assumed that one possesses a family of probability functions f(x;v) defined over X parameterized by a parameter vector v. Using the following steps, the algorithm sequentially updates the parameter such that the corresponding probability density becomes concentrated around the global minimum:

```
1. Given an initial parameter vector $v=v_0$, sample
   a random population of K solutions $x_1,x_2,…,x_k$
   from the distribution given by f(x;v).
2. Evaluate the costs S(xi),i=1,…,K.
3. Using the ρK (0<ρ<1) elite (lowest cost)
   samples, obtain a new density function f(x;v)
   by calculating a new vector v via Maximum
   Likelihood (ML) estimation.
4. Repeat steps 1-3 with the new vector v unless
   maximum number of iterations is reached or no
   improvement is obtained for a predefined number
   of iterations.
```

To adapt the generic algorithm to the mapping problem, we maintain a vector of probabilities of length *N* (number of modules to be mapped) for each tile in the placement matrix. Each entry in the vector represents the probability that the corresponding module is placed in that tile. Namely, each tile is associated with a multinomial random variable with *N* possible outcomes – an outcome for each module. The probability distribution for sampling a solution (a mapping) is thus determined by *N* multinomial random variables parameterized by the corresponding vectors of probabilities.

For each tile, we generate a random sample of the corresponding multinomial random variable to specify the module to be placed in that tile (step 1). Once the module is placed, it cannot be placed in any other tile. Thus, the corresponding probabilities of other tiles are nullified and the remaining probabilities are re-normalized.

After evaluating the costs of all sampled mappings (step 2), we update the probabilities associated with each tile. To this end, we consider the elite set of sampled mappings. The probability of module *i* to be placed in tile *j* is relative to the number of times this module is placed in that tile in the elite sample set. For the above probability distribution, this update scheme is the ML estimation of step 3.

6. EVALUATION

In this section, we evaluate the potential benefit that lies in the extended formulation of the mapping problem. We use a 6x6 tiled network on which 36 cores are mapped. For each random system requirement scenario, two mappings are compared: one is the best mapping generated by a classic algorithm and the other is the best mapping found using the extended formulation. In order to get reliable results, each experiment is repeated 12 times. All flows have an identical bandwidth of 1 unit. For the parameters of the CE algorithm, we use ρ=0.1 and K=2000 to comply with the recommended values in the CE literature for problems of this size [19]. Typical running time until convergence is a few minutes per mapping on a PC with a 3GHz processor.

6.1 Class Modeling

In order to evaluate the gain enabled by the extended formulation capturing the classes of replicated modules 4.1), we create random application data-flow graphs which are mapped to the system cores. To allow a fair comparison, the maximal load of each class module in the extended formulation (σ_i in Eq. 7) is limited to the maximal load presented to it when the classic formulation is used. Figure 6 shows the cost (Eq. 4) reduction for a varying number of class modules and size of each class. If only a single class of modules is available, the total cost is reduced by 6% when the number of modules in that class is 4 (11.1% of the 36 modules in the system). The saving grows to 33% as the class size is increased to 20 (55.6% of all modules). For the same total portion of replicated modules, increasing the number of classes (thus reducing the size of each class) results in a lower reduction in cost. For example, four classes of size five each (20 replicated modules in total, 55.6% of all modules) lowers the cost by 20% compared to the mapping generated by a classic algorithm.

6.2 Extended Timing Specification

To compare the mappings generated using the classical and extended timing formulations, we use random application data-flow graphs. We consider a many-modules SoC composed of a multi-stage single stream application (similar to the one illustrated in Figure 1a), and some additional modules. The application data-flow graphs account for two types of traffic. Type I traffic is composed of a set of chained flows, going through all stream processing modules one by one. Type II traffic is composed of all other flows, running from sources to destinations selected uniformly at random from all system modules. To complete the specification of the data-flow graph, random timing requirements are set for flows in the system. In the extended specification, the timing requirements of the stream flows (type I traffic) are replaced with a single requirement that is equal to the sum of the replaced constraints and restricts the total delay of the path from the first module of the chain to the last one (total pipeline delay).

Figure 7 shows the savings in case the pipeline is 16 modules long and type II traffic has timing requirements randomly selected in the range of 1 to 4 hops. The savings are presented versus the total allowed pipeline delay, for varying amounts of timing requirements for the type II flows. The X-axis reflects the rigidity of the pipeline delay requirements: “tight” means that the mapping of the modules must keep the number of hops composing the pipeline no more than 50% longer than the pipeline’s own length (24 in the case of a 16 module pipeline); “medium” is for 75%, “loose” is for 100%; and “very loose” for 125%, representing a case in which the pipeline has a large slack time. The Y-Axis represents the cost reduction compared to the solution achieved when a classic algorithm is used. Each group of bars represents a different amount of extra timing requirements, i.e., constraints imposed on type II flows. For example, “20% extra constraints” means that there are 20% more timing requirements in addition to the constraints imposed on the flows composing the pipeline stream. These cases reflect different latency needs outside the main processing data path.

The extended timing specification results in up to 18% reduction in the cost of communication. Interestingly, for different pipeline delay requirements, the peak saving is achieved for a different mixture of constraints. For example, for a pipeline with a tight timing requirement, the largest reduction is achieved when type II traffic has no timing constraints, while for a very loosely bounded

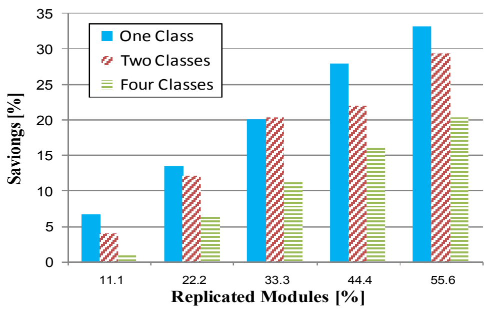


Figure 6: Average cost savings

Reduction of the total communication cost due to specification of classes of replicated modules.

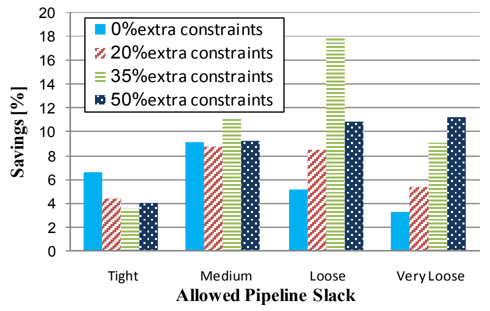


Figure 7: Average cost savings

Reduction of the total communication cost due to extended specification of timing requirements for various amounts of point-to-point constraints.

pipeline, the peak saving is received when the non-pipeline traffic has a large number of timing requirements. This is caused by the complex effects of increasing the allowed pipeline delay and adding point-to-point timing requirements. Intuitively, allowing a looser pipeline constraint increases the potential benefit that lies in the extended timing specifications as larger delays can be traded-off between different stages of the pipeline. However, when the total pipeline requirement is very loose, the point-to-point constraints of the pipeline are large enough to allow a classic algorithm to generate an efficient mapping.

7. SUMMARY

In this paper, we examine the NoC module assignment problem in light of two trends already witnessed in current SoC designs, caused by the steep increase in the number of transistors available on a chip and by the growing importance of keeping power consumption low. The first is the availability of replicated modules that have identical functionality. The second is the implementation of task-specific optimized modules (rather than general purpose processing elements), prolonging the path traversed by data on the chip. We argue that the design process of the NoC should account for this development, as classic modeling prevents exploiting the true benefits of a NoC based design. We show the importance of modifying the traditional formulation of the network assignment problem to account for classes of replicated modules and application-level latency needs. The new formulation leads to a more efficient tile mapping and yields significant savings in the cost of communication.

8. ACKNOWLEDGMENTS

We would like to thank Rudy Beraha for his helpful suggestions and insights. This work was partially supported by the Semiconductor Research Corporation (SRC) and Intel Corp.

REFERENCES

[1] D. Abts, N.E. Jerger, J. Kim, D. Gibson, and M. Lipasti, "Achieving Predictable Performance Through Better Memory Controller Placement in Many-Core CMPs", in *International Symposium on Computer Architecture*, 2009.

[2] P.H.N de With and E.G.T Jaspers, "Chip-Set for Video Display of Multimedia Information", *IEEE Trans. on Consumer Electronics*, vol. 45, pp. 707-716, 1999.

[3] J. Hu and R. Marculescu, "Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures", in *Proc. Design Automation & Test in Europe (DATE)*, 2003.

[4] S. Murali and G. De Micheli, "Bandwidth-Constrained Mapping of Cores onto NoC Architectures", in *Design, Automation and Test in Europe Conference (DATE)*, 2004, pp. 896-901.

[5] C. Marcon et al., "Exploring NoC Mapping Strategies: an Energy and Timing Aware Technique", in *Design, Automation and Test in Europe Conference (DATE)*, 2005, pp. 502-507.

[6] G. Ascia, V. Catania, and M. Palesi, "Multi-Objective Mapping for Mesh-Based NoC Architectures", in *Int. conf. on Hardware/Software Co-Design and System Synthesis (CODES ISSS)*, 2004, pp. 182-187.

[7] R. Tornero, J.M Orduna, M. Palesi, and J. Duato, "A Communication-Aware Topological Mapping Technique for NoCs", in *the 14th int. Euro-Par conference on Parallel Processing*, 2008.

[8] K. Srinivasan, K.S. Chatha, and G. Konjevod, "An Automated Technique for Topology and Route Generation of Application Specific On-Chip Interconnection Networks", in *International Conference on Computer Aided Design*, 2005.

[9] Z. Wenbiao, Y. Zhang, and Z. Mao, "Link-load balance aware mapping and routing for NoC", *WSEAS Transactions on Circuits and Systems*, vol. 6, no. 11, pp. 583-591, November 2007.

[10] C.E. Rhee, H.Y. Jeong, and H. Soonhoi, "Many-to-Many Core-Switch Mapping in 2-D Mesh NoC Architectures", in *IEEE International Conference on Computer Design*, 2004, pp. 438-443.

[11] S. Murali, L. Benini, and G. De Micheli, "Mapping and Physical Planning of Networks-on-Chip Architectures with Quality-of-Service Guarantees", in *Asia South Pacific design automation*, 2005.

[12] K. Srinivasan and K.S. Chatha, "A Technique for Low Energy Mapping and Routing in Network-on-Chip Architectures", in *Low Power Electronics and Design*, 2005, pp. 387-392.

[13] A. Hansson, K. Goossens, and A. Radulescu, "A Unified Approach to Constrained Mapping and Routing on Network-on-Chip Architectures", in *International conference on Hardware/software co-design and system synthesis (CODES ISSS)*, 2005, pp. 75-80.

[14] K. Goossens et al., "A Design Flow for Application-Specific Networks on Chip with Guaranteed Performance to Accelerate SOC Design and Verification", in *Design, Automation and Test in Europe Conference (DATE)*, 2005, pp. 1182-1187.

[15] S. Murali, M. Coenen, A. Radulescu, K. Goossens, and G. De Micheli, "Mapping and Configuration Methods for Multi-Use-Case Networks on Chips", in *ASP-DAC*, 2006, pp. 146-151.

[16] A. Hansson, M. Wiggers, A. Moonen, K. Goossens, and M. Bekooij, "Enabling Application-Level Performance Guarantees in Network-Based Systems on Chip by Applying Dataflow Analysis", in *IET Computers & Digital Techniques*, 2009.

[17] D. Shin and J. Kim, "Communication Power Optimization for Network-on-Chip Architectures", *Journal of Low Power Electronics*, vol. 2, pp. 165-176, August 2006.

[18] W.H Hung and C.L Yang, Y.S Chang, A. Su Y.J Chen, "An Architectural co-Synthesis Algorithm for Energy-Aware Network-on-Chip Design", in *ACM symposium on Applied computing* , 2007, pp. 680-684.

[19] R.Y. Rubinstein and D.P. Kroese, *The Cross-Entropy Method: a Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning.*: Springer-Verlag NY Inc, 2004.