2012 IEEE 18th International Conference on Parallel and Distributed Systems

Topology Virtualization for Throughput Maximization on Many-Core Platforms

Tianyi Wang, Gang Quan
Department of Electrical and Computer Engineering
Florida International University
Miami, Florida 33174
Email: {tiawang, gaquan}@fiu.edu

Shangping Ren
Department of Computer Science
Illinois Institute of Technology
Chicago, Illinois 60616
Email: ren@iit.edu

Meikang Qiu
Department of Electrical and Computer Engineering
University of Kentucky
Lexington, Kentucky 40506
Email: mqiu@engr.uky.edu

***Abstract*—As transistor's feature size continues to scale down into the deep sub-micron domain, IC chip performance variation caused by manufacturing process becomes un-negligible and can cause significant discrepancies between an application's nominal design and its actual realization on individual many-core platforms. In this paper, we study the problem on how to reduce the total schedule length of a task graph when realizing its nominal design on individual Network-on-Chip(NoC) based many-core platform with faulty cores. Different from traditional approaches to re-define the mapping/scheduling decisions in the nominal design, our methods judiciously mirror the physical architecture of each individual platform to the logical platform, based on which the nominal design is conducted. To facilitate the phyical/logic architecture virtualization, we develop a performance metric based on the *opportunity cost*, a concept borrowed from the economics field. Three virtualization heuristics are presented in this paper. Our experimental results show that the proposed approach can achieve up to 30% with an average 15% performance improvement by taking advantage of the heterogeneity of each individual platform.**

***Keywords*-process variations; multi-core/many-core; virtualization; nominal design; performance yield**

I. Introduction

With the continuous scaling down of the transistor feature size, billions of transistors are integrated on a single chip [1]. Multi-core/many-core architecture is becoming mainstream. Most of desktop computers and server computers nowadays use high performance processors with multiple processing cores. Intel has announced more advanced many-core platforms consisting of 48 and 80 general purpose processing cores [2], [3], [4].

In the meantime, however, as transistor feature size continues to shrink to the degree below the wavelength of light used to print them, it becomes difficult to precisely control the manufacturing process. This can lead to significant variations in key transistor parameters, such as transistor channel length, channel width, oxide thickness, and threshold voltage, which can further result in the maximal working frequency and power consumption of processing core varying from core to core and chip to chip [5], [6], even if all of them use the same, identical design. The 2008 International Technology Roadmap for Semiconductor (ITRS) [1] predicts that circuit variability will increase from 48% to 66% in the next ten years.

One major problem caused by manufacturing variations is the fabrication yield. Reduced feature size and increased chip area have increased the number and density of transistors on a single die, leading to a significantly decreased fabrication yield. According to [7], without considering defect tolerance during the architecture design phase, even under the best case, the yield of cell processors can be as low as only 10% to 20%. Therefore, micro-architecture level and core-level redundancies are employed to improve the fabrication yield. According to [8], incorporating core-level redundancy, at or below 100 nm technology, will achieve better yield performance than micro-architecture level redundancy.

Another serious problem caused by manufacturing variations is performance variations, such as maximum clock frequency, power dissipation, etc. It has been shown that the frequency variation can be as much as 30% and up to 20x variations in chip leakage power for a processor designed in 180nm technology [9]. Based on a test structure fabricated in IBM's 65 nm Silicon-On-Insulator (SOI) technology, Aarestad et al. [10] showed that worst case delay variations caused by chip-to-chip process variations can be as large as 21%. As design parameters of processing cores deviate from their nominal values, the system design objectives can be severely compromised, or even worse, a computing system can malfunction or even fail.

Significant achievements have been made in recent research [6], [11] by employing new materials. However, layout techniques and other device technologies on mitigating performance variations, which are induced by manufacturing variations will become increasingly challenging as transistor size continues to scale towards dimensions close to or below 10 nm [12]. Besides extensive research on device and layout level techniques (e.g [6], [11]) to address manufacturing-variation problems, there are increasing interests to address this problem from architecture and system levels. For example, performance binning techniques are proposed (e.g. [13]) to cluster chips with similar performance. As a result, even in the presence of large manufacturing variations, processors of the

This work is supported in part by NSF under projects CNS-0969013, CNS-0917021, CNS-1018108, CNS-1018731, and CNS-0746643.

1521-9097/12 $26.00 © 2012 IEEE
DOI 10.1109/ICPADS.2012.63

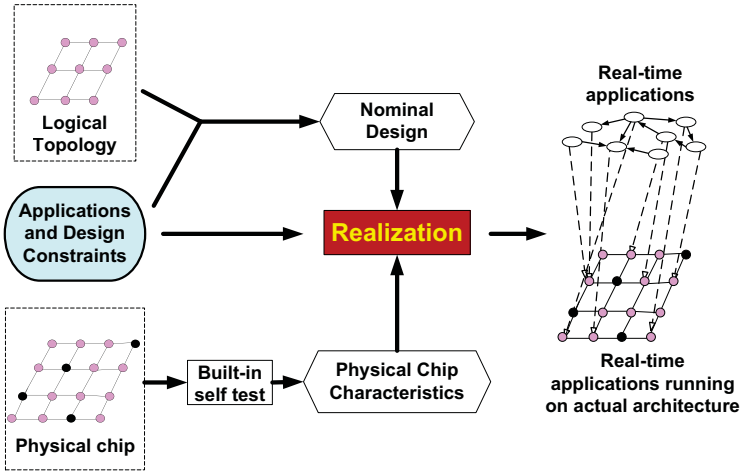


Fig. 1. A framework to take advantage of performance heterogeneity to optimize system performance. An advanced built-in-self-test module is associated with each chip to collect the performance characteristics of the chip. The collected information is then used to map the physical hardware architecture to the logical architecture based on which the nominal design is conducted, with the goal to maximize the performance of the nominal design on this particular chip.

same grade have less performance variations. One drawback of this approach, though, is that it cannot deal with performance variations among different cores within the same chip.

In the presence of performance variations, it becomes too pessimistic to design a system based on worst-case scenarios. Therefore, instead of adopting the traditional design methodology which is based on deterministic parameters, several researchers incorporate statistical analysis into system-level design. Wang et al. [14] presented a task allocation and scheduling algorithm to map a task graph to a multi-core platform with the goal to maximize the performance yield rate, i.e. the probability that a processor can meet the desired performance of a given application. They further extended their work to considered not only performance differences of multiple cores, but also physical link differences in NoC as well. Momtazpour et al. [15] considered a similar task graph mapping problem on a multi-core NoC architecture, with the goal to maximize the percentage of manufactured chips that can meet power constraints for a given application. These statistical approaches try to optimize results in a probabilistic manner. However, from the perspective of an individual processor, the designs can be too optimistic or too pessimistic due to different performance variations.

We believe that the core heterogeneity due to performance variations, if handled properly, can in fact improve the performance of a nominal design. As a result, in this paper, we are interested in developing appropriate *virtualization* techniques that can judiciously mirror physical architecture to logical architecture and at the same time improve the throughput of the nominal design on each individual hardware platform. Figure 1 illustrates the virtualization framework of our approach. We assume that each chip is equipped with an advanced *built-in-self-test*(BIST) module, that can detect faulty cores and capture performance variances when a device starts. Note that simple modules such as those introduced in [16], [6] can be easily incorporated into a multi-core platform for detecting purpose. The performance characteristics captured by the BIST module will be used to mirror the logical architecture to the underlying physical architecture with the goal of maximizing the application performance.

A few researches [17], [18], [19], [20], [21] have been conducted which are closely related to our work. Zhang et al. [19], [20] proposed several heuristics to replace faulty cores with redundant cores to improve the fabrication yield. They further extended their work to deal with performance variations by constructing sub-meshes using cores with similar performance [21]. These approaches do not take application characteristics into consideration. A more recent work proposed by Yue at el. [17], [18] improved upon Zhang's work [19], [20] by taking application characteristics into consideration, and intended to maintain the similar real-time performance after replacing faulty cores with redundant cores. These approaches only deal with faulty cores and do not take performance variations into consideration.

In this paper, our goal is to minimize the execution latency of an application by properly mirroring a physical multi-core architecture, which may have faulty cores and significant core-level performance variations, to the logical architecture. Three virtualization heuristics are presented in this paper. Specifically, we introduce a novel performance metric developed based on the *opportunity cost* [22], i.e. a concept originated from the economics domain, to guide our virtualization process. We have conducted extensive experimental studies to investigate the benefits of the proposed framework and heuristics. Our experimental results show that the heuristics can achieve up to 30% (with 15% in average) performance improvement (i.e. schedule length) over the existing methods.

The rest of the paper is organized as follows. In Section II,

we first use an example to motivate the research in this paper and then formulate the problem. We discuss the virtualization heuristics we developed in Section III. Experimental results are discussed in Section IV. We draw the conclusions in Section V.

II. Preliminary

In this section, we first use a simple example to motivate our research. We then introduce the system models used in this paper and define the research problem formally.

A. Motivating example

Consider an application where task graph is as shown in Figure 2(a). The application is designed for a multi-core architecture with standard 3×3 mesh. Assume that the nominal design, i.e. the design based on the nominal performance of the chip, has been given and shown on the logical mesh in Figure 2(a): tasks with same colors and shades are assigned to the same core. For example, tasks 0,3,7, and 12 are mapped to core (0,0); tasks 2,6, and 10 to core (0,1); tasks 4,8, and 9 to core (1,0); and tasks 1,5, and 11 core (1,1).

Now consider when realizing this design on a practical platform. To improve the yield rate, manufacturers usually add redundant cores on the same chip. In our case, we assume the physical mesh size of the chip is 3×4 with 3 redundant cores shown in Figure 2(b). Assume that core (1,1) happens to be a faulty core. One approach is to replace the faulty core with a redundant core using the Row Rippling Column Stealing (RRCS) algorithm presented in [19] or similar approaches detailed in [17], [18]. However, these approaches do not take core-to-core performance variations into consideration. As shown in Figure 2(c), instead of replacing the faulty core only, we can re-map the physical architecture to the logical architecture to optimize the performance of the nominal design.

Since programmers make the nominal design solely based on the logical topology without being aware of what the physical topology really looks like, opportunities exist to mirror the logical topology based on the actual performance and other characteristics of the physical topology to optimize the system performance. For instance, in Figure 2(b), the logical mesh is 3×3. When running application programs according to the nominal design, the operating system (OS) only cares about a logical mesh of 3×3, without knowing how this logical mesh is mapped to the underlying physical mesh. As a result, we can judiciously choose the physical topology to the logical topology mapping (such as Figure 2(c)) such that nominal design performance is maximized for each individual hardware platform.

Since we assume that we know the specifications of real-time applications, and with BIST module, we are able to know the exact performance for each core. Theoretically, we can then re-map and re-schedule task nodes accordingly. However, this essentially implies that we have to re-design the entire application for each individual platform, which can be expensive if not infeasible at all. Note that, by virtualizing each individual physical hardware architecture properly to the logical architecture, we have the potential to take advantage of the uniqueness of each chip to optimize the system performance without the need to change its software.

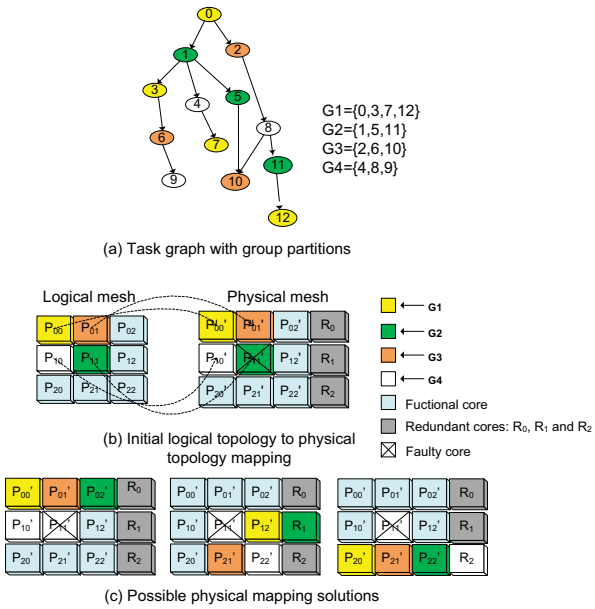


Fig. 2. A motivation example.

B. System models

In this section, we formally define our system models, including both application model and architecture model.

The *application* in the paper is modeled as a directed acyclic task graph $G = \{V, E\}$. $V = \{v_1, v_2, ..., v_k\}$ and each task node v_i represents a task in the application. We use $|v_i|$ to represent the execution time of task node v_i under the nominal frequency. $E = \{e(i,j) = (v_i, v_j) |$ if task node v_i communicates with task node v_j $\}$. Each arc, i.e. $e(i,j) \in E$ also indicates the dependency between two task nodes v_i and v_j with direction from task node v_i to task node v_j. A weight $w(e(i,j))$ is associated with each arch $e(i,j)$ to represent the data volume to be transferred from task v_i to v_j.

For the *logical architecture* (denoted as $A^l_{r \times c}$), we assume it consists of $r \times c$ homogeneous cores that form a standard $r \times c$ mesh architecture, i.e.

$$A^l_{r \times c} = \{C^l_{i,j}, i = 0, ..., r-1; j = 0, ..., c-1\}. \quad (1)$$

where $C^l_{i,j}$ represents the core located at position (i,j) in the logical mesh architecture. We assume that each core has the same nominal frequency of 1. We also assume that the communication at any link has a nominal speed of 1. In our system model we focus on the process variations on each individual core, and we assume there is no process variations on the links.

The *nominal design* of application G based on the logical architecture $A^l_{r \times c}$ (denoted as $\mathcal{N}(G, A^l)$) is defined by the

mapping between the task nodes in G and processor cores in $A^l_{r\times c}$. That is

$$\begin{aligned}\mathcal{N}(G,A^l) &= \{(v_i,C^l_{x,y})|v_i \text{ is assigned to core } C^l_{x,y}\}, \quad (2)\\ i &= 1,...,k; \quad (3)\\ 0 &\le x \le r-1;\\ 0 &\le y \le c-1.\end{aligned}$$

We assume the traditional NoC mesh network architecture and the deterministic X-Y routing algorithm [23] is used.

We define the *physical architecture* (denoted as $A^p_{m\times n}$) as a $m\times n$ mesh architecture, i.e.

$$A^p_{m\times n} = \{C^p_{i,j}, i = 0,...,m-1; j = 0,...,n-1\}. \quad (4)$$

where $C^p_{i,j}$ represents the core located at position (i,j) in the physical mesh architecture. We use f_{ij} to represent the maximum clock frequency of core $C^p_{i,j}$, which is normalized to the nominal frequency of the logical core. $f_{ij}=0$ indicates that core $C^p_{i,j}$ is a faulty core.

C. Problem formulation

With the system model defined above, we are now ready to define our research problem.

Problem 1: Given

- an application G;
- a logical architecture $A^l_{r\times c}$;
- the nominal design of G on $A^l_{r\times c}$, i.e. $\mathcal{N}(G,A^l)$;
- the physical architecture $A^p_{m\times n}$ and its performance variations, i.e. $f_{ij}, i = 0,...m-1; j = 0,...,n-1$,

Find the mapping of $\mathcal{M} = \{C^l_{i,j} \to C^p_{x,y} | i = 0,...,r-1; j = 0,...,c-1; 0 \le x \le m-1; 0 \le y \le n-1\}$ such that the maximum latency to execute G based on $\mathcal{N}(G,A^l)$ is minimized.

III. Our approach

In this section, we present three approaches to solve Problem 1 as defined above. The first approach is a simple heuristic that tries to match the logic node with the largest workload to the highest performance core in the physical architecture. The second and the third approaches are developed based on the *opportunity cost*, a concept originated from the economics discipline. The second approach considers only the core performance. The third approach considers not only the core performance but also the communication cost.

A. A simple workload/performance matching heuristic

Our goal is to map the logical topology to the physical topology such that an existing nominal design can achieve the best performance in the presence of faulty cores and performance variations on the physical topology. Since different cores may have different performances or processing speeds, an intuitive approach is therefore to match the logical core with the largest workload assignment to the physical core with the highest processing speed. The rationale behind this approach is that, the larger the workload is assigned to highest performance core, the more workload can be benefited from the highest processing speed. The algorithm, which we called *simple workload/performance matching* (SWPM) heuristic, is presented in Algorithm 1.

Algorithm 1 A simple heuristic to match high workload logical core to high performance physical core.

1: $\mathcal{M} = \emptyset$;
2: LC = The sorted list of $C^l_{i,j} \in A^l_{r\times c}, i = 0,...,r-1; j = 0,...,c-1$ by their workload based on nominal design $\mathcal{N}(G,A^l)$ in decreasing order;
3: LP = The sorted list of $C^p_{i,j} \in A^p_{m\times n}$ by f_{ij}, $i = 0,...,m-1; j = 0,...,n-1$ in decreasing order;
4: **for** $i = 0$ to $sizeof(LC)-1$ // for each logical core in the sorted list **do**
5: **if** The total workload assigned to $LC(i) > 0$ **then**
6: $\mathcal{M} = \mathcal{M} + \{LC(i) \to LP(i)\}$;
7: **end if**
8: **end for**

Algorithm 1 sorts the logic cores based on the assigned workloads and the physical cores based on their performances. Then, a logic core is mapped one by one to a physical core accordingly from the two lists. The complexity mainly comes from sorting of the two lists. we assume the physical mesh is larger than the logical mesh. Therefore, the complexity of Algorithm 1 is $O((m\times n)log(m\times n))$.

While Algorithm 1 is fast and intuitive, it has several issues. First, even though those high performance cores are used to speedup executions of larger workloads, these workloads do not necessarily locate on the *critical path*, i.e. the longest execution path of a task graph. In that case, the latency improvement when executing the task graph is limited. Second, Algorithm 1 considers only performance differences of different cores and do not take their locations into consideration. When two neighboring logical cores are separated far away in the physical mesh, the increased communication overhead can offset the performance improvement or even degrade the overall performance, or the latency when executing tasks. In what follows, we develop two new approaches to address these problems.

B. Opportunity cost based workload/performance mapping

It is desirable to optimize the latency of the critical path to improve the performance when realizing the logical topology to the physical topology. In the meantime, however, optimizing the critical path too aggressively may cause other execution paths to become critical and thus degrade the optimization performance. The problem is then how to develop effective algorithms for logical/physical topology mapping that can optimize the maximum latency when executing a task graph. In what follows, we discuss a heuristic developed for this goal. For the sake of simplicity, we first assume the communication cost is negligible.

When designing a highly effective logical to physical topology mapping, one critical problem is how to evaluate the impact or performance of a decision when mapping a logical

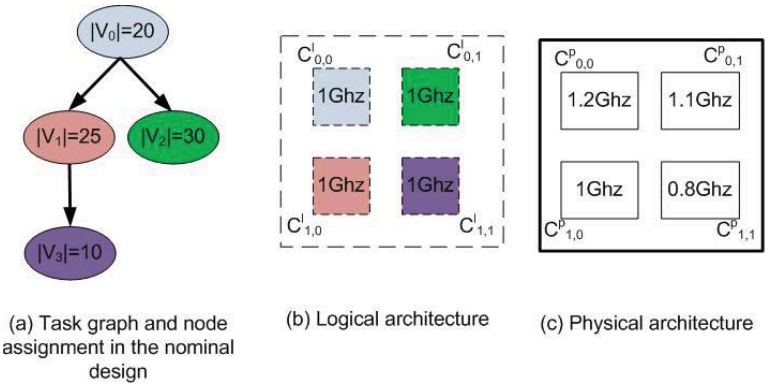


Fig. 3. An illustrative example for opportunity cost and performance metric. The colors and shades of task nodes in (a) imply the corresponding assignment to the logical topology: $v_0 \rightarrow C^l_{0,0}$, $v_1 \rightarrow C^l_{1,0}$, $v_2 \rightarrow C^l_{0,1}$, $v_3 \rightarrow C^l_{1,1}$.

node to a physical node. Note that it is not difficult to prove that Problem 1 is in fact NP-hard. In our approach, we resort to adopting the *opportunity cost* as the metric to make our decisions. The opportunity cost is the cost of any activity measured in terms of the value of the next best alternative forgone (that is not chosen). It is the sacrifice related to the second best choice available to someone, or group, who has picked among several mutually exclusive choices. For more details about the opportunity cost, readers can refer to [22] or other related references.

We use a simple illustrative example to explain the concept of the opportunity cost and its use in designing our performance metric. Figure 3(a) shows a task graph with four nodes. The colors and shades represent their assignments to the logical topology as shown in Figure 3(b). To calculate the latency when executing a task graph, we assume that, if a logical core has not been mapped, all the task nodes assigned to this logical core take their nominal execution time; if a logical core has already been mapped, then new execution times on the practical cores will be used.

Algorithm 2 Workload/performance mapping based on opportunity cost.

1: $\mathcal{M} = \emptyset$;
2: LC = The list of $C^l_{i,j} \in A^l_{r\times c}, i = 0,...,r-1; j = 0,...,c-1$ with workload assignments larger than 0;
3: LP = The list of $C^p_{i,j} \in A^p_{m\times n}$, excluding faulty cores;
4: **while** $LC \neq \emptyset$ **do**
5: Find $C^l_{i,j} \in LC$ and $C^p_{x,y} \in LC$ such that $\mathcal{P}(C^l_{i,j} \rightarrow C^p_{x,y})$ is maximized;
6: $\mathcal{M} = \mathcal{M} + \{C^l_{i,j} \rightarrow C^p_{x,y}\}$;
7: Remove $C^l_{i,j}$ and $C^p_{x,y}$;
8: **end while**

Now consider the decision of mapping logical core $C^l_{0,0}$ to physical core $C^p_{0,0}$. The task graph latency of this mapping is 51.67. Since the latency in the nominal design is 55, we define that the *profit* of this decision is 55 - 51.67 = 3.33. For the rest of the alternatives to map logical core $C^l_{0,0}$, the best choice is to map it to $C^p_{0,1}$ with latency of 53.18. The corresponding profit is 55 - 53.18 = 1.82, which is the opportunity cost to map $C^l_{0,0}$ to $C^p_{0,0}$. We thus define the performance of the decision as the difference of its profit and opportunity cost, or 3.33 - 1.82 = 1.51. In what follows, we formally define the performance metric used in our approach.

Definition 1: Given a decision to map logical core $C^l_{i,j}$ to physical core $C^p_{x,y}$, i.e. $C^l_{i,j} \rightarrow C^p_{x,y}$, let its profit be denoted as $Prof(C^l_{i,j} \rightarrow C^p_{x,y})$, and let its opportunity cost (i.e. the performance associated with the best choice to map $C^l_{i,j}$ other than $C^p_{x,y}$) be denoted as $OC(C^l_{i,j} \rightarrow C^p_{x,y})$. Then the *performance* of the decision, denoted as $\mathcal{P}(C^l_{i,j} \rightarrow C^p_{x,y})$, is defined as

$$\mathcal{P}(C^l_{i,j} \rightarrow C^p_{x,y}) = Prof(C^l_{i,j} \rightarrow C^p_{x,y}) - OC(C^l_{i,j} \rightarrow C^p_{x,y}). \quad (5)$$

Specifically, for the example in Figure 3, we have $\mathcal{P}(C^l_{0,0} \rightarrow C^p_{0,0}) = 1.51$, $\mathcal{P}(C^l_{0,1} \rightarrow C^p_{0,0}) = 0$, $\mathcal{P}(C^l_{1,0} \rightarrow C^p_{0,0}) = 1.9$, and $\mathcal{P}(C^l_{1,1} \rightarrow C^p_{0,0}) = 0.76$. It is interesting to note that, according to Definition 1, mapping the logical core with the largest workload assignment (i.e. $C^l_{0,1}$) to the fastest core (i.e.$C^p_{0,0}$) does not reduce the critical path latency and thus has the lowest performance.

After establishing the metric to evaluate a mapping decision, we are ready to introduce our heuristic algorithm, which is given in Algorithm 2. The most critical section of Algorithm 2 is the while loop, which selects the mapping with the largest performance according to equation (5). In the worst case, the complexity of the while loop is $O(kmn)$ since $m \times n$ different mappings need to be checked, and the complexity to obtain the latency for a task graph is k, where k is the number of task nodes. In the worst case, the while loop will be executed for $r \times c$ times. Therefore, the overall complexity of Algorithm 2 is $O(krcmn)$.

C. Logical/physical topology mapping with communication awareness

Neither Algorithm 1 nor Algorithm 2 takes the communication cost into consideration. They work fine when the

communication cost is really small and negligible. When the communication cost becomes significant, especially for many-core platforms, the qualities of the mapping results by Algorithm 1 and Algorithm 2 can be severely compromised. In what follows, we propose an iterative algorithm (shown in Algorithm 3) to improve the performance of existing mapping results with taking the communication into consideration.

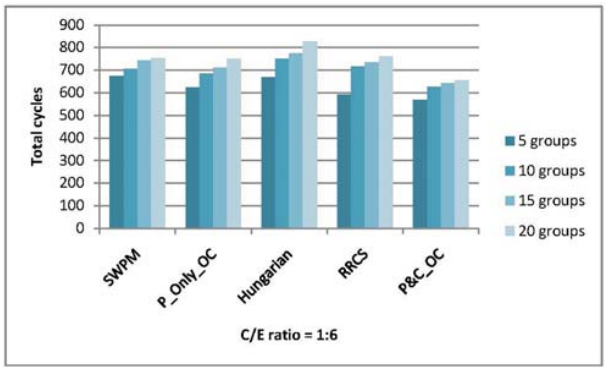
Algorithm 3 Logical/Physical mapping with communication cost awareness.

1: Initialize $\mathcal{M}_0$; // by Algorithm 1 or 2
2: L_{orig} = latency of executing G based on $\mathcal{M}_0$;
3: Improvement = 0;
4: **while** Improvement $< \varepsilon$ // user defined threshold **do**
5: LC = The list of $C^l_{i,j} \in A^l_{r\times c}, i = 0,...,r-1; j = 0,...,c-1$ with work assignment larger than 0;
6: LP = The list of $C^p_{i,j} \in A^p_{m\times n}$, excluding faulty cores;
7: $\mathcal{M} = \emptyset$;
8: **while** $LC \neq \emptyset$ **do**
9: Find $C^l_{i,j} \in LC$ and $C^p_{x,y} \in LC$ such that $\mathcal{P}(C^l_{i,j} \rightarrow C^p_{x,y})$ is maximized;
10: $\mathcal{M} = \mathcal{M} + \{C^l_{i,j} \rightarrow C^p_{x,y}\}$;
11: Remove $C^l_{i,j}$ and $C^p_{x,y}$;
12: **end while**
13: L_{new} = latency of executing G based on $\mathcal{M}_0$;
14: $Improvement = \frac{L_{orig}-L_{new}}{L_{orig}}$;
15: $L_{orig} = L_{new}$;
16: **end while**

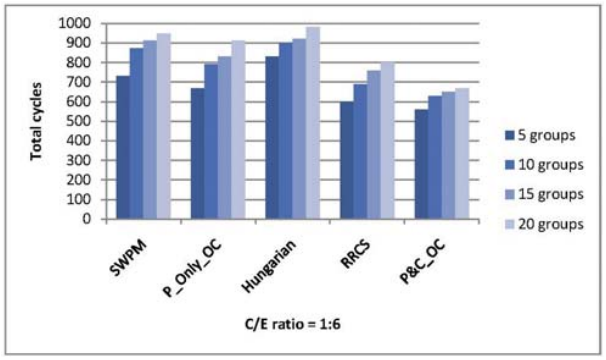
In principle, Algorithm 3 uses similar performance metric based on opportunity cost to evaluate a mapping decision. When calculating the latency for the task graph, the communication cost based on XY-routing can be incorporated into the calculation of performance of a mapping decision. Another major difference between Algorithm 3 and Algorithm 2 is that Algorithm 3 can iteratively improve the mapping solution, until the improvement threshold defined by user can be satisfied. The complexity of the while loop from line 8 to 12 is similar to that in Algorithm 2. The overall complexity of Algorithm 3 depends on the exact value of ε.

IV. Experimental results

In this section, we perform three experiments to study the performance of three approaches we presented in the previous section. For ease of presentation, we use *SWPM* to denote Algorithm 1, *P_Only_OC* for Algorithm 2, and *P&C_OC* for Algorithm 3. We also compare our algorithms with two previous work, i.e. the *RRCS* algorithm [20] and the *Hungarian* algorithm [18]. The *RRCS* algorithm intends to replace the faulty cores and reshape the mesh to mirror the logical mesh while the *Hungarian* algorithm tries to re-map the physical mesh to logical mesh to minimize the communication changes. We investigated the performance of these five different approaches under different mesh sizes, numbers of task nodes, communication/execution ratios, as well as their computational costs.



(a) 5×6 *mesh*



(b) 10×11 *mesh*

Fig. 4. Performance vs. different mesh sizes and different group numbers

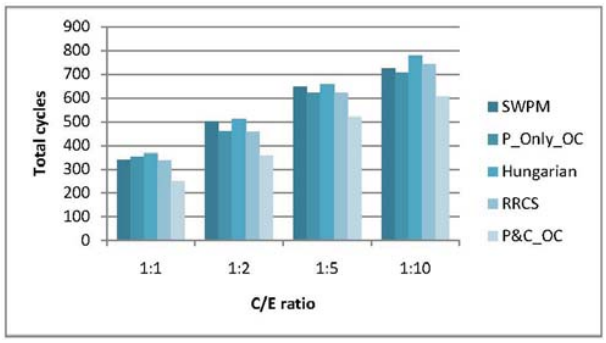
A. Experimental setup

In our simulation study, we used TGFF [24] to randomly generate task graphs (60 nodes) and also randomly cluster task nodes into groups and map to different logical cores, from a 5×6 and a 10×11 mesh. The reason we used $n \times (n+1)$ mesh is because the *RRCS* and the *Hugarian* algorithms assume such topology. The communication of each edge and execution time of each task are randomly generated. The frequency of each processor is also randomly generated using normal distribution with mean value $\mu = 1$, i.e. the nominal frequency, and variance value $\sigma = 0.1$, based on [9]. Unless specified otherwise, we assume the *P&C_OC* algorithm stops after 200 iterations. All experiments were running on a Window XP/SP3 platform powered by Intel(R) Core(TM)2 Duo CPU @ 2.93GHz with 3.21 GB of RAM.

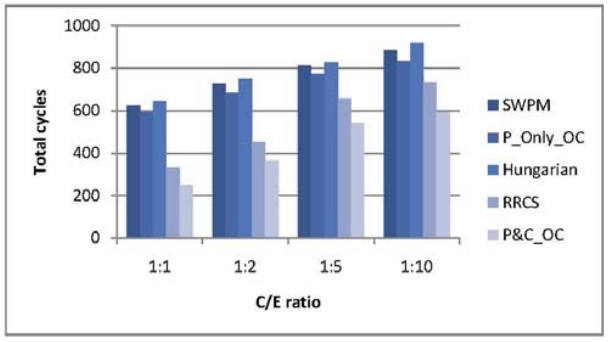
B. Performance vs. mesh sizes and group numbers

In this experiment, we compared the performance of different algorithms under different size meshes: 5×6 and 10×11. The execution time of a task node was randomly generated from interval [10:50]. The communication cost of an edge was randomly chosen from interval [1:10]. The average result among all test cases were collected and plotted in Figure 4.

From Figure 4, we can see that *P_Only_OC* consistently outperformed *SWPM* under different mesh sizes and different group numbers. For example, for mesh size of 5×6 and



(a) 5×6 *mesh*



(b) 10×11 *mesh*

Fig. 5. Performance vs. different communication/execution ratios

task group number of 5, we can see that *P_Only_OC* outperformed *SWPM* as much as 10% in latency reduction. This is because *SWPM* optimizes aggressively on the logical cores with large workload assignments. Unfortunately, as indicated in our previous illustrative example (Figure 3), the overall performance can be severely limited if the workloads are not located on the critical path. *P_Only_OC*, on the other hand, judiciously chooses logical/physical mapping based on application characteristics and thus can outperform *SWPM*.

When comparing *P_Only_OC* and *RRCS*, it is interesting to see that *P_Only_OC* performs better than *RRCS* for small meshes but becomes worse than *RRCS* when the mesh size is large or the task group number is small. For example, for mesh size of 5×6 and group size of 10, *P_Only_OC* outperformed *RRCS* by approximately 4%. For large mesh size of 10×11, *RRCS* can outperform *P_Only_OC* by as much as 12.5%. This is because *P_Only_OC* can take the application information into consideration and outperform *RRCS*. However, our experimental results also indicate that *P_Only_OC* works only in small mesh size. For large mesh sizes, the *P_Only_OC* algorithm can potentially distribute tasks far away from each other and therefore degrade the overall performance. And the *Hungarian* algorithm is the worst one as we have discussed previously, it is good for small mesh size, i.e. 5×5 and small number of faulty cores, i.e. no more than 4 faulty cores. However, in our setup, we assume 5 and 10 faulty cores for 5×6 mesh and 10×11 mesh, respectively. The *Hungarian*

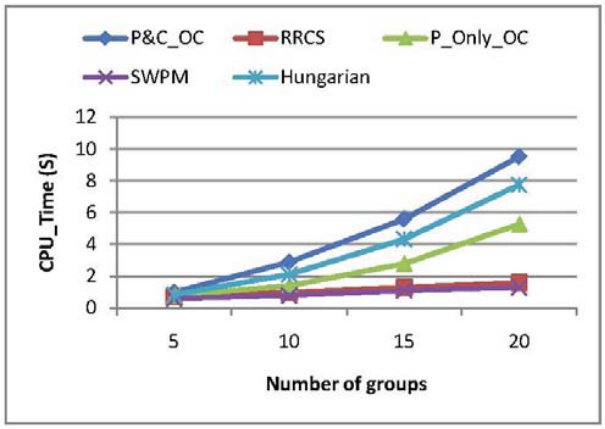


Fig. 6. Computational Time comparisons for different algorithms on 10×11 mesh

algorithm always tries to re-map the faulty cores using the redundant cores which are aligned to the right most column of the mesh, therefore, results in poor performance.

Finally, we can see that *P&C_OC* consistently outperforms other algorithms for different mesh sizes and groups, and the results improved with the growth of mesh size and number of task groups. From Figure 4, on average *P&C_OC* can outperform *RRCS* by 13% and 16% for mesh size of 5×6 and 10×11, respectively. The experimental results greatly highlight the excellent performance of *P&C_OC*.

C. Performance vs. different communication/execution ratios

Next, we study how communication cost can affect the performance of different approaches. Let communication cost be generated within interval [a,b] and execution time of task node be generated within interval [c,d], the C/E ratio can be defined in Equation 6.

$$ratio = \frac{b+a}{d+c}, \tag{6}$$

We randomly generated different test cases with different C/E ratio and tested the four algorithms mentioned above. The C/E ratios were set to be 1:1, 1:2, 1:5, 1:10. The results for different test cases were collected and plotted in Figure 5.

From Figure 5(b), the improvement of *P&C_OC* over *P_Only_OC* increases as communication cost increases. When communication cost is much less than the execution cost (C/E ratio = 1:10), *P&C_OC* improves upon *P_Only_OC* about 30%. When the communication cost is almost comparable to the execution cost (C/E ratio = 1:1), the average latency by *P_Only_OC* is more than double that by *P&C_OC* and *SWPM*, i.e. approaches that do not take the communication into consideration.

D. Computational cost

We next studied the computational cost for each algorithm on mesh size of 10×11. Figure 6 shows the computation times with different numbers of task groups for the five algorithms.

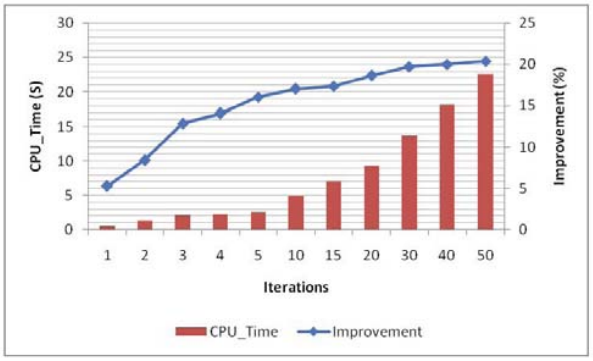


Fig. 7. Computational Time and Improvement comparisons for different iterations on 10×11 mesh

For the *P&C_OC* algorithm, we set the threshold to 16%. It is not surprising to see that *SWPM* and *RRCS* have computational cost nearly linear to the task group, while *P_Only_OC* , *Hungarian* and *P&C_OC* are increasing very fast, as discussed before.

To further understand the computational complexity of *P&C_OC*, we conducted another set of experiments with 20 task groups and kept track of the solution quality for each iteration. As shown in Figure 7, we can see that from the first iteration all the way to 50th iteration, the CPU time increases rapidly. The improvement also grows rapidly during the first several iterations until it reaches around 22% in improvement and becomes saturated. How to speedup the *P&C_OC* without compromising its solution quality is an interesting problem worthy of future study.

V. Conclusions

Performance variations can reduce the fabrication yield and degrade the quality of the nominal design. Different from previous research at the device level, during the post-fabrication, or the statistical approach at the system level, we propose to deal with the process variations when deploying the nominal design to a dedicated device. We introduce a framework to judiciously reconfigure the underlying physical architecture to mirror the logical architecture and maximize the performance of the nominal design. Heuristics based on the concept of opportunity cost are introduced in the paper. From our experimental studies, the proposed approach can achieve up to 30% and with an average 15% of performance improvement (i.e. schedule length) by taking advantage of the heterogeneity of each individual platform.

References

[1] ITRS, *International Technology Roadmap for Semiconductors (2008 Edition)*. Austin, TX.: International SEMATECH, http://public.itrs.net/.

[2] J. Howard, S. Dighe, and Y. Hoskote, "A 48-core ia-32 message-passing processor with dvfs in 45nm cmos," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, Feb. 2010, pp. 108–109.

[3] L. Seiler and D. Carmean, "Larrabee: A many-core x86 architecture for visual computing," *Micro, IEEE*, vol. 29, no. 1, pp. 10–21, Jan. 2009.

[4] S. Vangal, J. Howard, and G. Ruhl, "An 80-tile sub-100-w teraflops processor in 65-nm cmos," *Solid-State Circuits, IEEE Journal of*, vol. 43, no. 1, pp. 29–41, Jan. 2008.

[5] S. Nassif, K. Bernstein, D. Frank, A. Gattiker, W. Haensch, B. Ji, E. Nowak, D. Pearson, and N. Rohrer, "High performance cmos variability in the 65nm regime and beyond," in *Electron Devices Meeting, 2007. IEDM 2007. IEEE International*, 2007, pp. 569–571.

[6] K. Kuhn, "Managing process variation in intel's 45nm cmos technology," *Intel Technical Journal*, vol. 12, no. 2, pp. 93–110, Jun. 2008.

[7] E. Sperling, "Turn down the heat ... please," *Electronic Design, Strategy, News*, p. 1, Jul. 2006.

[8] P. Shivakumar, S. W. Keckler, C. R. Moore, and D. Burger, "Exploiting microarchitectural redundancy for defect tolerance," in *Proceedings of the 21st International Conference on Computer Design*, ser. ICCD '03. IEEE Computer Society, 2003, pp. 481–. [Online]. Available: http://dl.acm.org/citation.cfm?id=946246.946573

[9] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Proceedings of the 40th annual Design Automation Conference*, 2003, pp. 338–342.

[10] J. Aarestad, C. Lamech, J. Plusquellic, D. Acharyya, and K. Agarwal, "Characterizing within-die and die-to-die delay variations introduced by process variations and soi history effect," in *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, Jun. 2011, pp. 534–539.

[11] K. Kuhn, "Cmos transistor scaling past 32nm and implications on variation," in *Advanced Semiconductor Manufacturing Conference (ASMC), 2010 IEEE/SEMI*, Jul. 2010, pp. 241 –246.

[12] Y. Taur, "Cmos design near the limit of scaling," *IBM Journal of Research and Development*, vol. 46, no. 2.3, pp. 213–222, Mar. 2002.

[13] J. Sartori, A. Pant, R. Kumar, and P. Gupta, "Variation-aware speed binning of multi-core processors," in *Quality Electronic Design (ISQED), 2010 11th International Symposium on*, Mar. 2010, pp. 307–314.

[14] F. Wang, C. Nicopoulos, X. Wu, Y. Xie, and N. Vijaykrishnan, "Variation-aware task allocation and scheduling for mpsoc," in *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, Nov. 2007, pp. 598–603.

[15] M. Momtazpour, E. Sanaei, and M. Goudarzi, "Power-yield optimization in mpsoc task scheduling under process variation," in *Quality Electronic Design (ISQED), 2010 11th International Symposium on*, Mar. 2010, pp. 747 –754.

[16] J. Oliver, R. Amirtharajah, V. Akella, and F. T. Chong, "Credit-based dynamic reliability management using online wearout detection," in *Proceedings of the 5th conference on Computing frontiers*, 2008, pp. 139–148.

[17] K. Yue, S. Ghalim, Z. Li, F. Lockom, S. Ren, L. Zhang, and X. Li, "A greedy approach to tolerate defect cores for multimedia applications," in *Embedded Systems for Real-Time Multimedia (ESTIMedia), 2011 9th IEEE Symposium on*, Oct. 2011, pp. 112–119.

[18] K. Yue, F. Lockom, Z. Li, S. Ghalim, S. Ren, L. Zhang, and X. Li, "Hungarian algorithm based virtualization to maintain application timing similarity for defect-tolerant noc," in *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, Feb. 2012, pp. 493–498.

[19] L. Zhang, Y. Han, Q. Xu, and X. Li, "Defect tolerance in homogeneous manycore processors using core-level redundancy with unified topology," in *Design, Automation and Test in Europe, 2008. DATE '08*, Mar. 2008, pp. 891–896.

[20] L. Zhang, Y. Han, Q. Xu, X. wei Li, and H. Li, "On topology reconfiguration for defect-tolerant noc-based homogeneous manycore systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, no. 9, pp. 1173–1186, Sep. 2009.

[21] L. Zhang, Y. Yu, J. Dong, Y. Han, S. Ren, and X. Li, "Performance-asymmetry-aware topology virtualization for defect-tolerant noc-based many-core processors," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, Mar. 2010, pp. 1566–1571.

[22] J. M. Buchanan, "Opportunity cost," in *The New Palgrave Dictionary of Economics Online (Second ed.)*, 2010.

[23] C. Glass and L. Ni, "The turn model for adaptive routing," in *Computer Architecture, 1992. Proceedings., The 19th Annual International Symposium on*, 1992, pp. 278 –287.

[24] R. Dick, D. Rhodes, and W. Wolf, "Tgff: task graphs for free," in *Hardware/Software Codesign, 1998. (CODES/CASHE '98) Proceedings of the Sixth International Workshop on*, Mar. 1998, pp. 97–101.