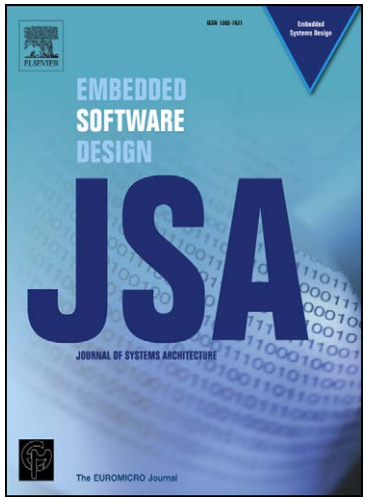


An Architectural Co-Synthesis Algorithm for Energy-Aware Network-on-Chip Design

Yi-Jung Chen, Chia-Lin Yang, Yen-Sheng Chang

PII: S1383-7621(09)00025-3
DOI: 10.1016/j.sysarc.2009.02.002
Reference: SYSARC 869

To appear in: *Journal of Systems Architecture*

Received Date: 21 October 2008
Revised Date: 25 February 2009
Accepted Date: 25 February 2009

Please cite this article as: Y-J. Chen, C-L. Yang, Y-S. Chang, An Architectural Co-Synthesis Algorithm for Energy-Aware Network-on-Chip Design, *Journal of Systems Architecture* (2009), doi: 10.1016/j.sysarc.2009.02.002

An Architectural Co-Synthesis Algorithm for Energy-Aware Network-on-Chip Design ⋆

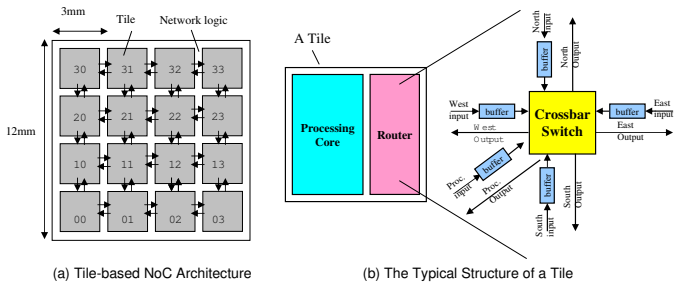
Yi-Jung Chen, Chia-Lin Yang *, Yen-Sheng Chang

Department of Computer Science and Information Engineering
National Taiwan University
No. 1, Sec. 4, Roosevelt Road, Taipei, 10617 Taiwan(R.O.C.)

Abstract

Network-on-Chip (NoC) has been proposed to overcome the complex on-chip communication problem of System-on-Chip (SoC) design in deep sub-micron. A complete NoC design contains exploration on both hardware and software architectures. The hardware architecture includes the selection of Processing Elements (PEs) with multiple types and their topology. The software architecture contains allocating tasks to PEs, scheduling of tasks and their communications. To find the best hardware design for the target tasks, both hardware and software architectures need to be considered simultaneously. Previous works on NoC design have concentrated on solving only one or two design parameters at a time. In this paper, we propose a hardware-software co-synthesis algorithm for a heterogeneous NoC architecture. The design goal is to minimize energy consumption while meeting the real-time requirements commonly seen in embedded applications. The proposed algorithm is based on Simulated-Annealing (SA). To compare the solution quality and efficiency of the proposed algorithm, we also implement the branch-and-bound and iterative algorithm to solve the hardware-software co-synthesis problem of a heterogeneous NoC. With the given synthetic task sets, the experimental results show that the proposed SA-based algorithm achieves near-optimal solution in a reasonable time, while the branch-and-bound algorithm takes a very long time to find the optimal solution, and the iterative algorithm fails to achieve good solution quality. When applying the co-synthesis algorithms to a real-world application with PE library that has little variation in PE performance and energy consumption, the iterative algorithm achieves solution quality comparable to that of the proposed SA-based algorithm.

Key words: Network-on-Chip, Hardware-Software Co-synthesis, Energy-aware Design



(a) Tile-based NoC Architecture (b) The Typical Structure of a Tile

Fig. 1. Architectural Overview of an NoC-based System [11]

1 Introduction

To cope with the complexity of System-on-Chip (SoC) design in billion transistors, Network-on-Chip (NoC) has been proposed to overcome the complex on-chip communication problem [4]. As shown in Figure 1(a), an NoC-based system is typically divided into a number of regular tiles interconnected by a 2D mesh network. A tile is composed of a *processing element* (PE) and a *router*. A PE could be a general-purpose CPU, Digital Signal Processors (DSP), Field-Programmable Gate Arrays (FPGAs), memory blocks, or Application-Specific Integrated Circuits(ASICs). PEs communicate with one another by sending packets via the mesh network instead of routing wires. The router embedded in each tile consists of input and output links, buffers and a crossbar switch. The abstract view of a tile is shown in Figure 1(b) [11].

The design flow of an NoC is shown in Figure 2. We assume an IP-centric design where PEs are selected from the IP library [25]. Given a set of target tasks (represented in task graph), a set of PEs are selected from the IP library (PE Selection), and tasks are allocated to PEs (Task Allocation). The selected PEs are mapped to the $n \times n$ tiles (Tile Mapping), and the schedule of tasks allocated to the same PE is decided (Task Scheduling). The communication paths

⋆ The preliminary version of this paper was published in SAC 2007[10]. In this paper, we extend our conference paper with:
(1) Implementing a branch-and-bound and an iterative algorithm to solve the NoC co-synthesis problem.
(2) Comparing the solution quality and algorithm efficiency achieved by the proposed SA-based algorithms, the branch-and-bound algorithm and the iterative algorithm.
(3) In addition to the original energy model that considers dynamic energy consumption only, we use an additional energy model that considers system leakage when performing system synthesis.

* Corresponding author.
Email addresses: d91015@csie.ntu.edu.tw (Yi-Jung Chen), yangc@csie.ntu.edu.tw (Chia-Lin Yang), r92043@csie.ntu.edu.tw (Yen-Sheng Chang).
URL: http://www.csie.ntu.edu.tw/∼ yangc (Chia-Lin Yang).

among tasks allocated to different PEs are determined through routing path allocation. From the design flow we can see that a complete NoC design contains exploration on both hardware and software architectures. The hardware architecture includes the selection of PEs and their topology. The software architecture contains the allocation of tasks to PEs and scheduling of tasks and their communication. The design of the hardware and software architecture actually interplays with each other. For example, a good task schedule might reduce the required PE computing capability to meet the real-time requirements of tasks. To find the best hardware design for the target tasks, both hardware and software architectures need to be considered simultaneously.

In this paper, we propose a hardware-software co-synthesis algorithm for a heterogeneous NoC platform. Previous works on NoC design have concentrated on solving only one or two design parameters at a time. Hu et al. in [12] solve *task allocation* and *scheduling* for a given NoC-based hardware architecture. They also proposed a *tile mapping* algorithm for a given PE communication graph in [11]. Shin et al. in [20] use genetic algorithm (GA) to solve *link speed assignment* problem for NoC. All the previous work does not address the issue of selecting IPs from the IP library. In an IP-centric design, the IP library could soon contain hundreds or even thousands of IPs in the future. It is challenging for an SoC designer to pick up the IPs manually to achieve the optimal design. Therefore, in this paper, we propose a hardware-software co-design algorithm that address the PE selection problem, and considers the interplay between the steps in the co-design flow. To the best of our knowledge, this paper is the first co-synthesis algorithm that allows fast NoC design space exploration considering both energy and performance factors.

It is obvious that the solution space of the hardware-software co-synthesis problem is huge, and it is impossible to search the design space completely. In this paper, we propose a *Simulated Annealing*(SA) [14] based co-synthesis algorithm. SA is a widely used non-deterministic algorithm for solving combinatorial optimization problem. The proposed co-synthesis algorithm targets at

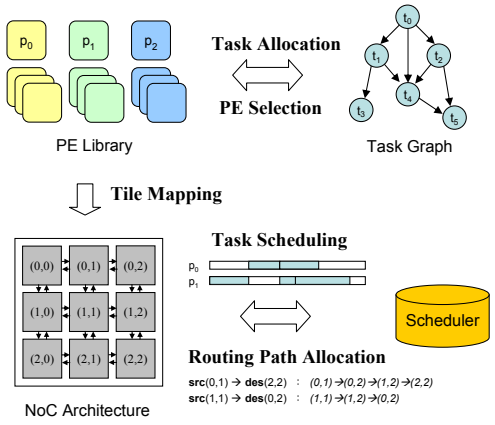


Fig. 2. Overall Network-on-Chip Design Flow

a heterogeneous NoC architecture running a task set with timing constraint. Since most embedded systems are battery-operated and have real-time requirements, the optimization goal of our co-synthesis algorithm is to minimize the energy consumption without violating real-time constraints. In this paper, we propose four SA-based co-synthesis algorithms; the baseline SA algorithm, Low-Temperature Moves on PE-Selection (LTM-PS), the Greedy PE-Selection method and Two-Stage SA. The details of each algorithm are described in Section 5.

To compare the solution quality and efficiency of the proposed algorithms, we also implement a branch-and-bound algorithm [5] and an iterative algorithm [25] to solve the hardware-software co-synthesis problem of NoC. Branch-and-bound is a general algorithm for finding optimal solutions of various optimization problems. In a branch-and-bound algorithm, a branch method is used to enumerate all the possible solutions, and a bounding method is used to prune branches that are not likely to achieve the optimal solution. An iterative algorithm, on the other hand, starts from the initial solution and iteratively converges to a better solution. Compared to the branch-and-bound algorithm, the iterative algorithm has short execution time but tends to fall into local optimal solutions. In this paper, we evaluate the proposed algorithms by comparing the the solution quality and algorithm efficiency of various co-synthesis algorithms. Here, we use the energy consumption of a synthesized system as the metric for solution quality, and the algorithm execution time as the metric for algorithm efficiency.

To evaluate the proposed SA-based co-synthesis algorithms, we apply all algorithms to a set of synthetic task sets generated by TGFF [6] and a real application task graph obtained from MPEG2 encoder. The experimental results show that the proposed Two-Stage SA performs the best among all SA-based algorithms. When considering dynamic energy consumption only, the Two-Stage SA algorithm achieves 32.9% and 7% less energy consumption than the baseline SA method with synthetic task sets and MPEG2 encoder, respectively. When comparing the performance of SA-based, branch-and-bound and iterative algorithms, the SA-based method achieves the best balance in both solution quality and execution time. Although the branch-and-bound algorithm always finds the optimal solution, the execution time of branch-and-bound is unacceptable when the solution space is large. For systems synthesized by the iterative algorithm, the energy consumption is up to 82% more than that of the system synthesized by baseline SA. With MPEG2 encoder, since the PE library has little variation in PE performance and energy consumption, the iterative algorithm achieves comparable solution quality to that of the SA-based method. Although the iterative algorithm can not achieve good solution quality for complex task sets, the iterative algorithm uses only 35% of baseline SA execution time to find a synthesis result.

The rest of this paper is organized as follows. We briefly review the related works on NoC design techniques in Section 2 and present a concise specification of our co-synthesis models in Section 3. A formal problem definition is described in Section 4. The proposed SA-based architectural co-synthesis algorithms are described in Section 5. The branch-and-bound and iterative algorithm for the co-synthesis problem of NoC are described in Section 6. The experimental results are discussed in Section 7. Section 8 concludes the paper.

2 Related Work

NoC has been proposed to mitigate on-chip interconnection problem [9, 2, 15, 4, 26]. In [4], Dally and Towles introduce the concept of on-chip networks, sketch a simple network, and discuss some challenges in the architecture and design of these networks. Ye et al. [26] give a detail analysis for the power consumption on network communication. Kumar et al. [15] propose a Network-on-Chip platform including both the architecture and the design methodology. Based on this architecture, Millberg et al. [16] present the Nostrum NoC supporting multiple communication services modelled by a protocol stack. All the above papers essentially advocate the advantages of using NoCs as effective means to design high performance SoCs.

For NoC system design, Hu et al. [11, 12] propose an energy-aware task allocation and scheduling algorithm which schedules both communication and computation for NoC architecture. They also propose an energy-aware tile mapping algorithm which exploits routing flexibility of regular NoC architectures. The algorithms in both papers are proposed to optimize only one or two aspects of the NoC-based design framework at one time. Shin et al. in [20] propose a communication optimization technique for NoC with voltage scalable links. In this work, they addressed the importance of inter-related steps of the NoC design flow and presented a GA-based algorithm to solve link speed assignment problem. However, it omits the PE Selection step and only focuses on link speed assignment to minimize communication energy cost. Murali and De Micheli propose an algorithm that maps processing cores onto a mesh NoC architecture under bandwidth constraints [17]. In [18], the same authors introduce SUNMAP, which automatically selects the best topology for a certain application under delay, area and energy constraints. In [19], a topology synthesis process that considers the effect floorplan is proposed. Srinivasan et al. [22, 23] propose an automatic technique to generate floorplan and route for application-specific NoC with irregular topology.

Several works [25, 24, 21] are proposed to solve the hardware-software co-synthesis problem of traditional bus-based distributed embedded systems. In [25], an architectural co-synthesis algorithm that considers PE selection and

task allocation simultaneously is proposed for distributed embedded system; however, their algorithm omits tile mapping and routing path allocation which are specific to the NoC-based systems.

3 System Models

Our system consists of four main models: a *real-time application model*, an *NoC architecture model*, a *PE library* and a *system energy model*.

(1) *Application Model*

We represent a real-time application by a *task graph* $G =< V, E >$, which is a *directed acyclic graph*, where V represents the set of tasks and E represents the set of directed edges between tasks. Each vertex $v_i \in V$ has following properties:

- $d(v_i)$ denotes the deadline of the node v_i which must be met to ensure correct functionality of the application.
- $type(v_i)$ denotes the type of this task node, which can be a general-purpose CPU, DSP, or ASIC.
- An array R^i, where the j-th element $r^i_j \in R^i$ gives the execution time of task v_i if v_i is executed on j-th PE p_j in the PE library.
- An array S^i, where the j-th element $s^i_j \in S^i$ gives the energy consumption of task v_i if v_i is executed on j-th PE p_j in the PE library.

Each $e_{i,j} \in E$ represents a precedence relation (v_i should be executed before v_j) between v_i to v_j and is associated with a value $c(e_{i,j})$ which indicates the amount of communication volume (bits) between v_i and v_j.

(2) *NoC Model*

The NoC architecture under consideration is composed of $m \times n$ tiles interconnected by a 2D mesh network. We model such an NoC-based system with $m \times n$ tiles as an *Architecture Graph* $N =< T, L >$, which is a *directed graph*, where $T = \{t_1, ..., t_{m\times n}\}$ is the set of tiles and L is the set of links between tiles. Each link $l_{i,j} \in L$ represents a link connection between t_i and t_j and is associated with $b(l_{i,j})$ which stands for the bandwidth (bits/second) of $l_{i,j}$.

Similar to [12], we also assume a *static XY* routing scheme [7] as our underlying routing protocol. It first routes packets along the X-axis. Once it reaches the column where the destination tile lies in, the packet is then routed along the Y-axis. Note that the proposed co-synthesis can be easily modified to apply other deterministic routing algorithm.

(3) *PE Model*

We denote the PE library $P = \{p_1, ..., p_n\}$, where p_i indicates the i-th PE in the PE library. We assume that the number of PEs are at least more than the number of tiles. Each p_i are associated with a $type(p_i)$ which

indicates the compatible task type of p_i. The task v_i can execute on p_i if and only if $type(p_i)$ is a general-purpose CPU or $type(p_i) = type(v_i)$.

(4) *Energy Model*

In this paper, we use two energy models to evaluate the energy consumption of synthesis results. The first energy model considers system dynamic energy only. The dynamic energy consumption of a system is composed of computation and communication energy consumption. According to the application model described before, computation energy consumption $E_{comput.}$ can be modelled as

$$E_{comput.} = \sum_{i=1}^{|V|} s_k^i \tag{1}$$

, where task i is mapped to the k-th PE in the PE library, and s_k^i is the energy consumption of task i running on the k-th PE. For the energy consumption of communication, we use $I_{bit}^{t_i,t_j}$ to denote the average energy consumption (in joules) of sending one bit of data from t_i to t_j, including energy consumed in the links and switches. We use the energy model in [26, 12, 11] to calculate $I_{bit}^{t_i,t_j}$. They define $I_{bit}^{t_i,t_j}$ as

$$I_{bit}^{t_i,t_j} = n_{hops} \times E_{S_{bit}} + (n_{hops} - 1) \times E_{L_{bit}} \tag{2}$$

, where $E_{S_{bit}}$ and $E_{L_{bit}}$ represent the energy consumed on the switch and on the link between tiles, respectively. The n_{hops} is the number of routers the bit passes on its way from t_i to t_j.

The second energy model used in this paper considers both dynamic and static energy consumption in the system. The dynamic energy consumption is modelled by Eq.(1) and Eq.(2). The static energy from the leakage of each component is modelled by Eq.(3), where P_{static} is the static power of selected PEs and routers, and T is the total execution time of system. A similar leakage model is also used in [13].

$$E_{static} = P_{static} \times T \tag{3}$$

4 Problem Formulation

For a given task graph $G =< V, E >$, a PE Library $P = \{p_1, p_2, ..., p_n\}$ and an NoC architecture $N =< T, L >$, the problem we want to solve is to find both the hardware and software architectures such that the overall energy consumption is minimized and specified performance constraints are met. For the overall NoC energy consumption, we break down the NoC hardware into two components: PE and interconnection. We can define the NoC co-synthesis problem as follows.

Given $G =< V, E >, P = \{p_1, ..., p_n\}$, and $N =< T, L >$,

Find a subset P' of P and the function ϕ, ω, η such that

$\{ \sum_{\forall v_i \in V} s^i_{\omega(v_i)} + \sum_{\forall e_{i,j} \in E} I^{\phi(\omega(v_i)),\phi(\omega(v_j))}_{bit} \times c(e_{i,j})\}$ is minimized

Subject to $\forall v_i \in V$, $completionTime(v_i) \leq d(v_i)$

In the problem formulation, $\sum_{\forall v_i \in V} s^i_{\omega(v_i)}$ is the total energy consumption on PEs, and $\sum_{\forall e_{i,j} \in E} I^{\phi(\eta(v_i)),\phi(\eta(v_j))}_{bit} \times c(e_{i,j})$ is the total energy consumption on interconnections (routers and links). When static energy consumption is considered, E_{static} defined by Eq.(3) is also evaluated for the overall NoC energy consumption. P' is the result of *PE Selection*, where $|P'| = |T|$. The function ϕ, ω, and η represent steps: *Tile Mapping*, *Task Allocation* and *Task Scheduling* respectively and are defined as below:

- **Tile Mapping**: Map each selected PE in P' onto one of tile of the NoC. We use the function $\phi : P' \rightarrow T$ to represent *"Tile Mapping"* step. Obviously ϕ is a one-to-one and onto function.
- **Task Allocation**: Assign each task node in V into one of compatible PE in P'. We use the function $\omega : V \rightarrow P'$ to represent *"Task Allocation"* step.
- **Task Scheduling**: Determine the execution order of the tasks and communication. For *"Task Scheduling"* in our problem, the set of all possible solutions consists of all the possible permutations of the tasks subject to the additional precedence and exclusion constraints and to their deadlines. We use the function $\eta : V \rightarrow V'$ to represent *"Task Scheduling"* step.

5 SA-based Architectural Co-Synthesis Algorithm

The design space for the NoC hardware-software co-design problem is huge, and all of the steps in the co-design flow, PE Selection, Tile Mapping, Task Allocation and Task Scheduling, actually interplay with one another. Therefore, in this work, we propose a Simulated-Annealing (SA) based hardware-software co-synthesis algorithm. SA is a widely-used non-deterministic algorithm for solving combinatorial optimization problems. Figure 3 shows a generic SA flow. Each iteration of SA is composed of three steps; Perturbation, feasibility test and cost evaluation. Each perturbation results in a new solution through a set of operations. Therefore, after each perturbation, feasibility test is required to verify if the solution violates its constraints or not. The quality of the solution is evaluated with a pre-defined cost function. The whole process is repeated until the SA termination condition is met.

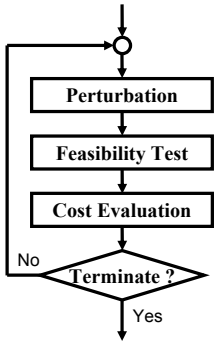


Fig. 3. Generic *Simulated-Annealing* (SA) Flow

In this section, we first describe the baseline SA algorithm. We then present three variations to the baseline algorithm; *Low-Temperature Moves on PE-Selection*(*LTM-PS*), *Greedy PE Selection* and *Two-Stage SA*.

5.1 The Baseline SA Algorithm

The easiest way to adopt the SA approach for the co-synthesis problem is to treat each co-synthesis step as a perturbation operation. We refer to this as the baseline SA algorithm. Figure 4 shows the baseline SA flow. The four steps in the NoC co-design flow (PE Selection, Tile Mapping, Task Allocation and Task Scheduling) are treated as perturbation operations. Some important ingredients of the baseline SA algorithm are defined as follows.

(1) **solution space**: The solution space is the combination of *PE Selection*(PS), *Tile Mapping*(TM), *Task Allocation*(TA), and *Task Scheduling*(TS). If we define solution space as S, then $S = P' \times \phi \times \omega \times \eta$, where P' is the selected PEs for the current solution.
(2) **neighborhood structure**: There are four types of perturbation in our SA engine: PS, TM, TA and TS.
   (a) **PS (PE Selection)**: The PS perturbation is to randomly pick $p_i \in P$ and swap p_i and p_j. Due to the heterogeneity of PE types, we have two cases for the PS perturbation:
      (i) $type(p_i) = type(p_j)$ or $type(p_j) = CPU$: swap p_i and p_j directly. If the newly selected PE is the same type as the replaced PE or is a CPU, which could execute any type of tasks, the tasks running on the replaced PE can run on the new PE directly.
      (ii) $type(p_i) \neq type(p_j)$ and $type(p_j) \neq CPU$: in this case, parts of the tasks running on p_i may not be able to execute on p_j. To handle this case, we select a CPU from P' and migrate these tasks to the selected CPU. If there is no CPU in P', we then redo the PS perturbation.

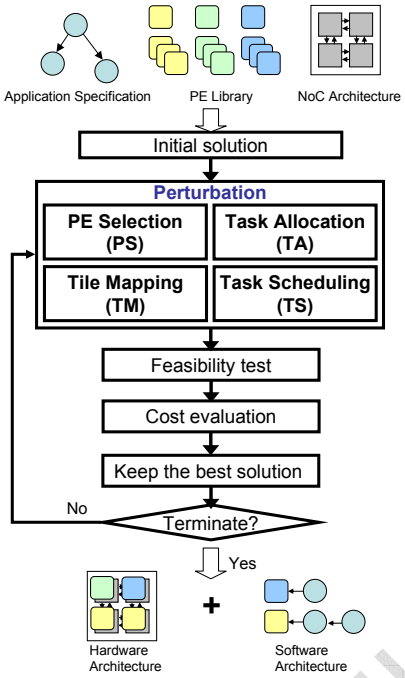


Fig. 4. Overview of the Baseline SA Algorithm for NoC Design

(b) **TM (Tile Mapping)**: TM is to pick $p_i, p_j \in P'$ randomly, where $p_i \neq p_j, t_i = \phi(p_i)$, and $t_j = \phi(p_j)$. Then we change the tile mapping to: $\phi(p_i) = t_j, \phi(p_j) = t_i$.

(c) **TA (Task Allocation)**: TA picks $v_i \in V$ randomly and selects a $p_i \in P'$ randomly, where v_i is compatible with p_i. Then migrate v_i into p_i.

(d) **TS (Task Scheduling)**: We adopt List Scheduling [1, 8] as our baseline scheduler. In List Scheduling, tasks are scheduled according to their precedence relations and priorities. In our SA-based List scheduler, the task priorities are first randomly given, and then we use the TS perturbation to change the priorities of the task set. More specifically, the TS is to randomly select $v_i, v_j \in V, v_i \neq v_j$, and then swap the priority of v_i and v_j. Note that communication traffic is taken into account for task scheduling.

(3) **cost function**: The objective function contains two parts: *energy cost* and *miss deadline penalty*.

$$\Phi = C_{energy} + C_{penalty} \tag{4}$$

, where Φ is the cost of current solution. We normalized both energy term and timing penalty term in the cost function. The energy term (C_{energy}) is the same as the objective function in the problem formulation.

$$\{\sum_{\forall v_i \in V} s^i_{\omega(v_i)} t + \sum_{\forall e_{i,j} \in E} I^{\phi(\omega(v_i)),\phi(\omega(v_j))}_{bit} \times c(e_{i,j})\}$$

and the $C_{penalty}$ is described as following:

(a) $C_{penalty} = 0$, if $T \leq T_d$

(b) $C_{penalty} = T - T_d + \epsilon$, if $T > T_d$

, where T_d is the timing constraint of the application, T is the current completion time of the application and ϵ is a constant.

Recall that our optimization goal is minimizing the total energy consumption while meeting the tight performance constraint. In the first case, when the current solution satisfies the specified timing constraint, we concentrate on energy consumption optimization by setting $C_{penalty}$ to zero. In the second case, the completion time T violates the timing constraint T_d, therefore, both energy consumption and timing factors should be considered in searching for solutions. The $C_{penalty}$ is given more weight as the difference between the timing constraint and the current completion time gets larger. Note that we include ϵ in $C_{penalty}$ to distinguish a feasible solution from an infeasible one. The ϵ is an user-defined experimental parameter. A larger ϵ indicates that an infeasible NoC configuration is less likely to be accepted as the best configuration during the execution of SA. Since both energy term and timing penalty term in the cost function are normalized, ϵ should be $0 < \epsilon < 1$. In this paper, we set ϵ to 0.25 for all experiments.

5.2 LTM-PS: Low Temperature Move on PS

In the SA algorithm, after a perturbation, the derived solution is evaluated using the cost function to decide accepting or rejecting the solution. Since a PS perturbation changes the underlying hardware architecture, it implies that it might require a significant change in software architecture as well. For example, if the newly selected PE has lower computing power than the replaced one, it is very likely that the current schedule is not going to meet the timing constraints with the new set of PEs. Therefore, the new solution will be probably rejected by the SA due to its high cost. However, trivially rejecting this new PE configuration may foreclose possibly attracting PE configurations. In the example mentioned above, if we re-schedule the tasks according to the new hardware configuration, we might be able to find a feasible solution. Therefore, in the LTM-PS scheme, we try to optimize for the PE configuration by performing a low-temperature SA containing only TA, TM and TS perturbations on the new PE configuration before deciding to accept or reject the new PE configuration.

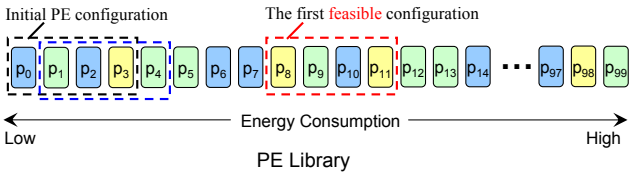


Fig. 5. Greedy PE-Selection Method

5.3 Greedy PE-Selection Method

Instead of randomly choosing a PE in each SA iteration, a heuristic approach is to select PEs in a greedy method as illustrated in Figure 5. We first sort the PEs in a non-decreasing order of their energy consumption [1]. We then choose the first n PEs where n is the number of tiles as our initial hardware configuration (P'). In the example shown in Figure 5, the initial hardware configuration P' contains p_0,p_1,p_2 and p_3 assuming a 2×2 NoC. For a selected PE configuration, we evaluate its feasibility with a low-temperature SA engine with only the TM, TA and TS perturbations. If there exist tasks that cannot be scheduled using P', we replace a PE by the CPU with the lowest energy consumption (CPU_{lowest}) in the sorted PE library. The victim PE (P_v) is the PE with the maximal energy consumption in P', and the new hardware configuration is $P' = (P' - P_v) \bigcup CPU_{lowest}$. If we can not find a feasible solution, we replace the PE with the lowest energy consumption in P' by the PE with the lowest energy among all the PEs that have energy consumptions larger than that of $P' - CPU_{lowest}$. We repeat this process until a feasible solution is found. We then perform a normal SA run on the selected PEs to determine the corresponding tile mapping and software architecture.

We can see that the greedy method only explores a subset of PE combinations. For example, the greedy method does not try the PE combination (p_0, p_3, p_4, p_5) in the example shown in Figure 5. This limitation may lead to a configuration with high energy consumption. To expand the solution space, we propose the Two-Stage SA algorithm described in the next section.

5.4 The Two-Stage SA Algorithm

The Two-Stage SA algorithm contains two stages as shown in Figure 6. The first stage is the aforementioned Greedy PE-Selection. After a feasible solution is found, all PEs with higher energy consumption than those in P' (the set of PEs of the feasible solution) are not considered in the second stage SA. In the

[1] We compute the average energy consumption of PE p_k by $\frac{1}{|P|}\sum_i^{|P|} s_k^i$.

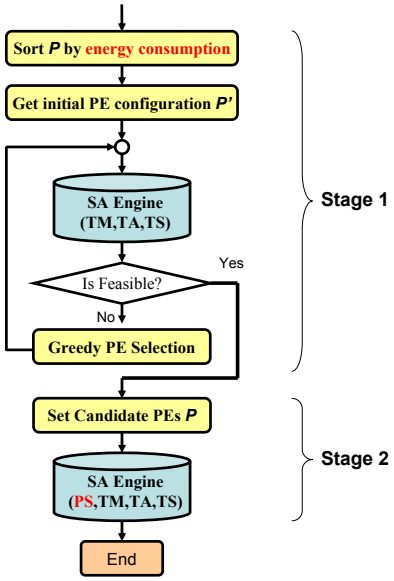


Fig. 6. Overview of Two-Stage SA

example shown in Figure 5, only the set of $\{p_0, p_1..., p_{11}\}$ are selected as the candidate PEs for the second stage SA. The objective of the first stage is to prune the design space. The second stage SA is the LTM-PS scheme described in Section 5.2.

6 Branch-and-Bound and Iterative Synthesis Algorithm

To evaluate the effectiveness of the proposed SA-based co-synthesis algorithms, we also implement a branch-and-bound and iterative algorithm to solve the hardware-software co-synthesis problem of NoC architecture. In this section, we describe how we implement the two algorithms.

6.1 The Branch-and-Bound Algorithm

Branch-and-bound is a general algorithm for finding optimal solutions of various optimization problems. The branch-and-bound method systematically enumerates all candidate solutions (branch method), and discards a branch of candidates by upper and lower estimated bounds of the quantity being optimized (bound method). For the NoC hardware-software co-synthesis problem, we use a search tree as shown in Figure 7 to enumerate all configurations. Each node in the tree is either a root, internal, or leaf node. The root node represents the initial state, which is an empty configuration. The search tree has three kinds of internal nodes: L1, L2, and L3 internal node. An L1 inter-

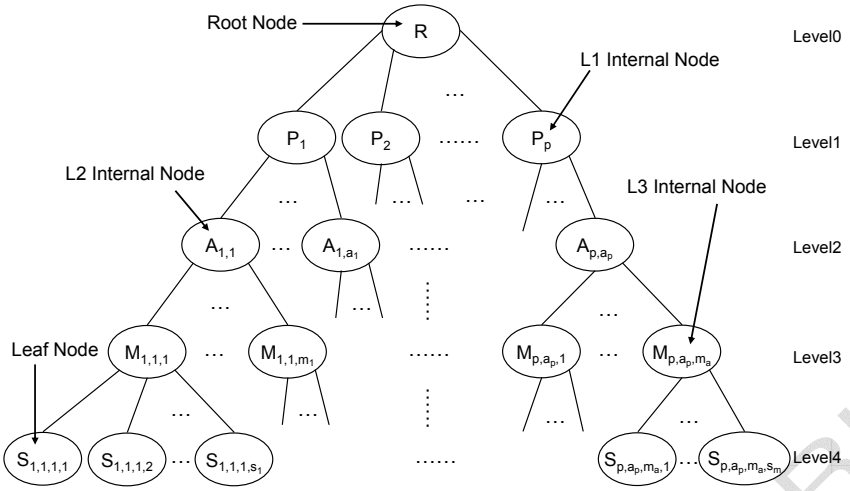


Fig. 7. Search Tree of NoC Design

Synthesis Step	Number of Configurations	Explanation on Parameters
PE Selection	$\binom{\|P\|}{m\times n}$	P:total num. of PEs in the PE library m, n: dimension of the target NoC
Task Allocation	$\|A\|^{m+n}$	A: total num. of tasks in the task set
Tile Mapping	$\|(m \times n)\|!$	m, n: dimension of the target NoC
Task Scheduling	$\|S\|!$	S: total num. of tasks
Total Num. of Config.	$\binom{\|P\|}{m\times n} \times \|A\|^{m+n} \times \|(m \times n)\|! \times \|S\|!$	

Table 1
Number of Configurations to be Explored in the Search Tree

nal node P_i represents a configuration of PE Selection. An L2 internal node $A_{i,j}$ represents the j-th Task Allocation configuration enumerated from the PE Selection configuration P_i. L3 internal nodes are the enumeration of all possible Tile Mapping based on the result of L2 internal nodes. We use $M_{i,j,k}$ to represent the k-th Tile Mapping configuration enumerated from $A_{i,j}$. A leaf node represents a complete system configuration, including Task Scheduling. We use $S_{i,j,k,l}$ to denote a leaf node representing the l-th Task Scheduling configuration enumerated from $M_{i,j,k}$. The search tree is explored by the Depth-First-Search (DFS) method. We can see that, the number of nodes in the search tree increases exponentially as the size of NoC template and task set increase, and the number of configuration to be explored in each synthesis step and the whole search tree are listed in Table 1.

To prune the solution space, we obtain the upper bound of a solution by running the low-temperature baseline SA, which converges quickly. The lower bound of a solution is set to the energy consumption estimated by the node's configuration. For example, the system energy consumption estimated from

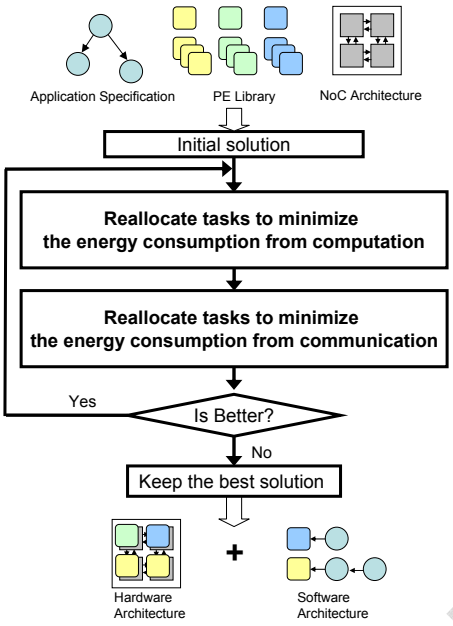


Fig. 8. Overview of the Iterative Algorithm for NoC Design

an L2 internal node is computation energy consumption only since an L2 node considers PE Selection and Task Allocation only. When the lower bound of the node is larger than its upper bound, the branch started from the node is discarded.

6.2 The Iterative Algorithm

The iterative algorithm starts with an initial solution, and iteratively improves the quality of the solution in the following iterations until further improvement can not be achieved. The iterative algorithm evaluated in this paper is an extension of the algorithm proposed in [25], which is targeting at bus-based Multi-Processor SoC. The iterative algorithm proposed in [25] consist of a step of finding the initial solution and an iterative refinement step that is executed repeatedly. In each iteration, the iterative refinement process modifies the system configuration to minimize energy consumption from task computation and communication, respectively. We maintain the structure of the algorithm proposed in [25], and modify the detail steps of finding the initial solution and performing iterative refinement to cope with the NoC co-synthesis problem. The overview of the iterative algorithm for NoC hardware-software co-synthesis is illustrated in Figure 8, and the overview of each operation is as follows.

Initialization Algorithm:
Input: Task graph $G =< V, E >$, NoC platform $N =< T, L >$ and PE library P
Output: An initial configuration with PS, TM, TA and TS are decided

```
    /* ————————Initial PE Selection and Task Allocation———————— */
1   Sort tasks in V according to their workload
2   V_{m×n} = the set of m × n tasks that are with the most workload
3   for each task t ∈ V_{m×n}
4       Selected_PE = Selected_PE ∪ Select_most_efficient(t, P);
5   for each task t ∈ V − V_{m×n}
6       Allocate_to_most_efficient(t, Selected_PE);
    /* ————————————————Initial Tile Mapping———————————————— */
7   Sort PEs∈Selected_PE according to their communication volume;
8   Put p ∈Selected_PE with the most communication volume at the middle of NoC
9   Unmapped_PE = Selected_PE - p;
10  for each p ∈Unmapped_PE
11      for each un-mapped tile position c that is closest to the mapped ones
12          fit_value_{p,c} = calculate_fit_value(p, c);
13      fit_position = {i|max{fit_value_{p,i}, ∀i ∈ un-mapped tile position}};
12      map_PE(p, fit_position);
```

Fig. 9. Initialization Process of the Iterative Algorithm

(1) **Initial solution**: Initial solution generates an initial configuration, including PE Selection, Tile Mapping, Task Allocation and Task Scheduling.
(2) **Iterative refinement**: Two steps are performed in the iterative refinement:
   (a) *Minimization of computation energy consumption*: Configurations of PE Selection and Task Allocation are modified to minimize computation energy consumption.
   (b) *Minimization of communication energy consumption*: Tasks are reallocated to minimize inter-PE communication.

To find a feasible initial solution, the algorithm tends to select PEs with high computation power. Although PEs with high computation power tend to have high energy consumption, these PEs can be replaced by the ones with low energy consumption during the iterative refinement process as long as the timing constraint is met. Therefore, for the initial solution, the algorithm selects PEs according to the tasks with the most workload. Assume an $m \times n$ NoC, the algorithm finds the $m \times n$ tasks with the most workload among the task set. For each task, the PE compatible to the task type and with the highest computation power is selected, and the task is also allocated to the PE. For the tasks other than the $m \times n$ tasks, each of them is allocated to the PE with feasible type and the highest computation power. On deciding the initial Tile Mapping, the algorithm tends to minimize the communication cost by

allocating PEs with high communication demands as close to one another as possible. To achieve this, the algorithm first allocates the PE with the highest communication needs in the middle of the NoC platform. For each un-mapped tile position that is adjacent to a mapped one, we calculate fit value $F_{PE_i,(x,y)}$ [2] of tile position (x, y) for all PE_i that is an un-allocated selected PE. $F_{PE_i,(x,y)}$ represents the average communication load of links between tile position (x, y) with PE_i and other mapped PEs. Therefore, the algorithm tends to map the PE with the highest fit value for an un-mapped tile position. The schedule of tasks allocated on a PE is set according to the precedence constraint of the input task graph. The priorities of tasks at the same level in the task graph are randomly decided. The flow of generating initial solution is summarized in Figure 9.

In the iterative refinement process, the algorithm minimizes system energy consumption in two directions; *computation* and *communication* energy consumption. The computation energy consumption is minimized by (1) replacing an selected PE PE_i by a PE in the PE library that is compatible to PE_i and has energy consumption lower than PE_i, and (2) re-allocating a task originally allocated on PE_i to another selected PE, which is feasible for the task and has energy consumption lower than PE_i. The communication energy consumption comes from the distance and communication volumes between two PEs on the system. To minimize communication energy consumption, the algorithm exhaustively merge tasks with high communication volume to the same PE as long as the merging leads to less on-chip communication.

7 Experimental Results

To evaluate the effectiveness of the proposed NoC hardware-software co-synthesis algorithms, we implement all the proposed SA-based algorithms and perform several experiments on synthetic and real-application task sets. In Section 7.1, the solution quality achieved by different co-synthesis algorithms are discussed. The efficiency of different algorithms is compared in Section 7.2.

[2] $F_{PE_i,(x,y)} = \sum_{\forall mapped\_PE_m} \frac{Commu(PE_i,PE_m)}{link(position(PE_m),(x,y))}$, where $Commu(PE_i, PE_m)$ denotes the communication volume between PE_i and PE_m, and $link(position(PE_m), (x, y))$ denotes the Manhattan-Distance between (x, y) the tile position that PE_m is mapped to.

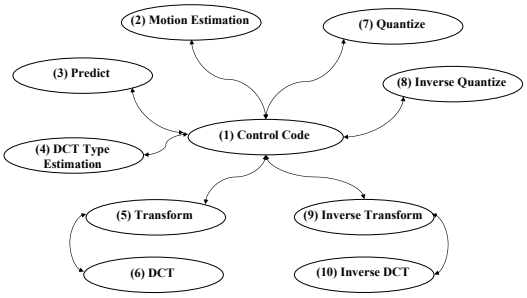


Fig. 10. Partitioning of MPEG2 Encoder

7.1 Comparison of Synthesis Results

For evaluation, we implement all the proposed SA-based co-synthesis algorithms; *baseline SA*, the *Greedy PE-Selection* method, *LTM-PS* and *Two-Stage SA*. We also implement the branch-and-bound and iterative algorithm described in Section 6 for comparison. For each SA-based algorithm, we ran 100 times and pick up the best result among runs. Note that, in Two-Stage SA, the first stage runs only once. The candidate PEs are then passed to the second-stage SA which are performed 100 times to derive the best solution. In the NoC-based architecture, we use a 2×2 tiles interconnected by a 2D mesh network.

We apply all the co-synthesis algorithms on two set of benchmarks; that is, synthetic task sets and a real-world application. The synthetic task sets are generated by the graph generator TGFF [6]. TGFF is a parameterizable generator that can accept user specifications like maximum in-degree and out-degree of the vertices. We generate random task graphs $g1$ to $g15$ which varies in graph size and in-out degree. A synthetic PE library is also generated for this set of tasks. The synthetic PE library contains three types of PEs: ASIC, DSP and CPU. The frequency and voltage of each type of PE is randomly generated, and there are 16 CPUs, 26 ASIC and 39 DSPs in this PE library.

To test the proposed NoC co-design flow on a real-world application, we apply our schemes to MPEG2 encoder [27]. MPEG2 encoder is divided into 10 tasks as shown in Figure 10. Note that we transform the cyclic task graph into an acyclic one by removing the incoming edge of the node that is the entrance node of a strongly connected component [3]. The traffic traces are obtained directly by executing the encoder on the SimpleScalar [30]. The PE library for MPEG2 encoder is constructed from the datasheets of IPs comprising DSP, CPUs, and customized ASICs that support the functionalities required by MPEG2 encoder. We select 29 CPUs from ARM [28], 8 DSPs from TI [31], and 19 ASICs from Philips [29]. Three different kinds of functionalities are provided by the selected ASICs. Because we only get execution cycle of each task from SimpleScalar [30], we set the execution cycle to 1 when a task

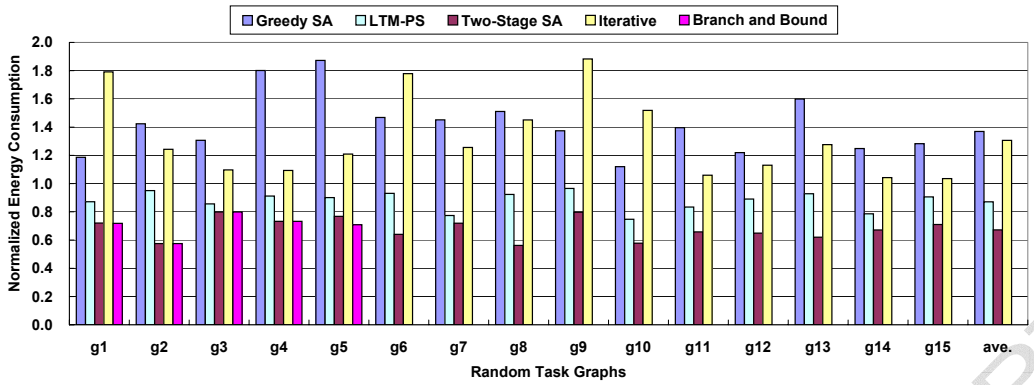


Fig. 11. Dynamic Energy Consumption of Solutions Synthesized by Various Co-Synthesis Algorithms

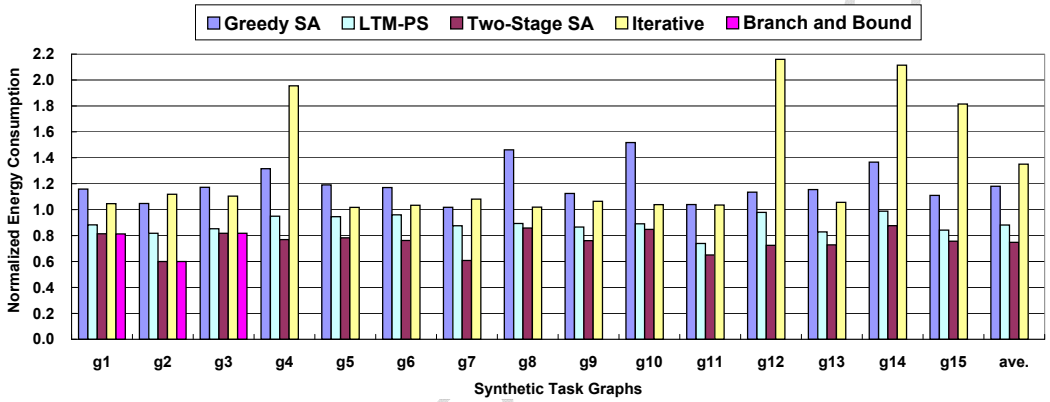


Fig. 12. Dynamic and Static Energy Consumption of Solutions Synthesized by Various Co-Synthesis Algorithms

executing on an ASIC and set the execution cycle to half of the CPU cycle when a task executing on a DSP.

Figure 11 and Figure 12 show the dynamic and dynamic+static energy consumption of systems synthesized by various co-synthesis algorithms, respectively. In both figures, the energy consumption is normalized to that of baseline SA. Among the SA-based methods, the experimental results show that Two-Stage SA performs the best and the Greedy PE-Selection method performs the worst, even worse than baseline SA. When considering dynamic energy consumption only, Greedy PE-Selection has 36.9% more energy consumption than baseline SA on the average. As described in Section 5.3, Greedy PE-selection always pick the n PEs that are sequential in energy consumption, where n is the number of tiles on the NoC platform. Therefore, task sets that have tasks with high computation demand would lead the Greedy PE-Selection method to select high frequency PEs and lose the chance of mapping tasks with low computation demands to PEs with low energy consumption. The Greedy PE-Selection method can be considered as an approach that a de-

signer would adopt without an automatic co-design environment. This result demonstrates the importance of PE Selection. We can also observe that LTM-PS dramatically improves the solution quality over baseline SA. When considering dynamic energy consumption only, LTM-PS achieves 13% less energy consumption than that of baseline SA. Two-Stage SA first uses the Greedy PE-Selection method to prune down the solution space, therefore, it can find better solutions in short time than LTM-PS. On the average, Two-Stage SA achieves 32.9% less energy consumption than baseline SA when considering dynamic energy consumption only.

When comparing SA-based algorithms with the branch-and-bound and iterative algorithm, we can observe that the proposed SA-based algorithms is able to synthesize a solution with good quality in a reasonable time. The branch-and-bound algorithm can always synthesize the optimal solution. When considering dynamic energy only, branch-and-bound achieves 52.5% less energy consumption than baseline SA. However, the execution time of the branch-and-bound algorithm is also extremely long. In this set of experiments, when performing the branch-and-bound algorithm, we can only get synthesis results of task graphs $g1-g5$, which have at most 13 tasks in their task graphs. When comparing the SA-based algorithms and the branch-and-bound algorithm, we can observe that the solution quality of configuration synthesized by Two-Stage SA is close to that of branch-and-bound. When considering dynamic energy only, among the five task graphs that the branch-and-bound algorithm is able to synthesize, Two-Stage SA has at most 5.9% more energy consumption than that of the branch-and-bound algorithm. We can also observe that the iterative algorithm, which only explores a subset of feasible solutions in each iteration, performs worse than baseline SA, LTM-PS and the Two-Stage SA in all cases. However, the iterative algorithm performs better than the Greedy PE-Selection method in some cases. As described earlier, the Greedy PE-Selection method tends to select a set of PEs with high computation power and high energy consumption when a task set needs PEs with high computation power to meet its deadline. In such cases, Greedy PE-Selection tends to perform worse than iterative algorithm.

When comparing Figure 11 and Figure 12, we observe that the differences between SA-based algorithms and iterative algorithm are shortened when considering static power consumption. SA-based methods tend to select PEs with lower voltage levels as long as the timing constraints are met. These PEs also lead to a longer execution time and thus have more leakage energy consumption. However, when considering static power consumption, except for Greedy PE-Selection, the SA-based methods still perform better than the iterative algorithm, and the Two-Stage SA still performs the best in all cases.

Figure 13 and Figure 14 show the dynamic and dynamic+static energy consumption of MPEG2 encoder system, respectively. The energy consumption is

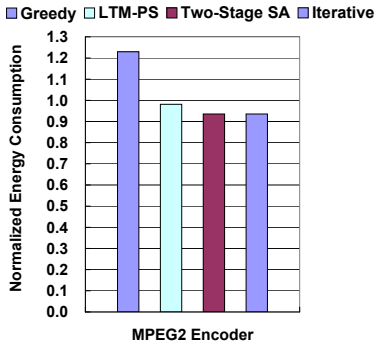


Fig. 13. Dynamic Energy Consumption of MPEG2 Encoder System Synthesized by Various Co-Synthesis Algorithms

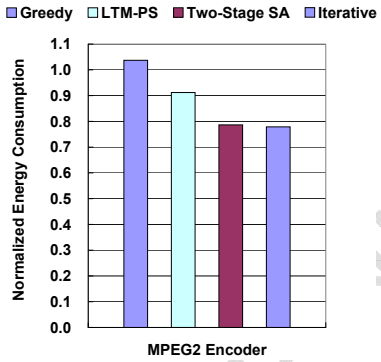


Fig. 14. Dynamic and Static Consumption of MPEG2 Encoder System Synthesized by Various Co-Synthesis Algorithms

also normalized to that of baseline SA. In this set of experiments, we do not show the results of the branch-and-bound algorithm since the communication pattern of MPEG2 encoder is complex and branch-and-bound can not synthesize its result in a reasonable time. In this set of experiments, Two-Stage SA achieves 6.5% less energy consumption than baseline SA. The Greedy PE-Selection method still performs the worst in this set of experiments. However, the results synthesized by the iterative algorithm is almost the same as that of Two-Stage SA. PE library used in this set of experiment has little variation in PE performance and energy consumption. Therefore, the PE library makes the co-synthesis algorithms hard to choose PEs to trade off between performance and energy consumption, and the iterative algorithm is easy to choose good configuration in the initial solution. When considering system leakage energy consumption (Figure 14), the result is similar to that of considering dynamic energy consumption only.

7.2 Comparison of Algorithm Efficiency

Another important metric for evaluating various hardware-software co-synthesis algorithms is how fast they find the synthesis result. Table 2 lists the execu-

Schemes	Running Time
Baseline SA	1
Greedy SA	1.40
LTM-PS	2.20
Two-Stage SA	2.07
Iterative Algorithm	0.35
Branch and Bound Method	6368.8

Table 2
Execution Time Evaluation

tion time of various schemes normalized to baseline SA. The results show that Two-Stage SA derives better solution than LTM-PS without using longer execution time. In Two-Stage SA, the first stage is invoked only once, and the second stage converges faster than LTM-PS because the PE searching space has been reduced. LTM-PS has longer execution time than baseline SA since a low-temperature SA is performed after each PS perturbation. The experimental results also show the iterative algorithm is the fastest among all the evaluated algorithms. Compared to SA-based algorithms, the solution space explored by the iterative algorithm explores is smaller. Therefore, the iterative algorithm tends to sacrifice solution quality to get execution efficiency. The branch-and-bound algorithm is the slowest among all the evaluated algorithms since it needs to exhaustively explore the design space.

8 Conclusion

In this paper, we propose an energy-aware architectural co-synthesis algorithm for Network-on-Chip (NoC) system design which simultaneously optimizes both software and hardware architectures to meet a tight performance constraint. We propose four types of SA-based co-synthesis algorithms. The baseline SA algorithm treats each co-design step as a perturbation; LTM-PS performs a low-temperature SA after each PS perturbation; the Greedy PE-Selection method tries the PE configurations in a non-decreasing order of their energy consumption; Two-Stage SA first uses the Greedy PE-Selection method to prune the design space and then invoke a complete SA to derive final hardware and software architecture. To compare the efficiency of the proposed SA-based algorithms, we also implement the branch-and-bound and iterative algorithm to solve the co-synthesis problem of NoC. Our experimental results show that the Two-Stage SA algorithm achieves the best solution quality in a reasonable execution time. When considering synthetic task set

and dynamic energy only, Two-Stage SA achieves 32.9% less energy consumption than baseline SA on the average.

References

[1] T. L. Adam, K. Chandy, and J. Dickson. A comparison of list schedules for parallel processing systems. *Commun. ACM*, 17(12):685–690, December 1974.

[2] L. Benini and G. De Micheli. Network on chips: A new soc paradigm. *IEEE Computers*, 35:70–78, January 2002.

[3] T. H. Coreman, C. E. Leiserson, R. L. Rivest, and C. Stain. *Introduction to Algoirthms*. McGraw Hill.

[4] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. *Proc. Design Automation Conference (DAC)*, pages 684–689, June 2001.

[5] G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.

[6] R. P. Dick, D. L. Rhodes, and W. Wolf. Tgff: Task graphs for free. *Proc. Intl. Workshop on Hardware/Software Codesign*, pages 97–101, March 1998.

[7] C. J. Glass and L. M. Ni. The turn model for adaptive routing. *Proc. international Symposium on Computer Architecture (ISCA)*, pages 278–287, May 1992.

[8] M. Grajcar. Strengths and weakness of genetic list scheduling for heterogeneous systems. *Proc. International Conference on Application of Concurrency to System Design (ACSD)*, pages 123–132, June 2001.

[9] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Öberg, M. Millberg, and D. Lindqvist. Network on a chip: An architecture for billion transistor era. *Proc. of the IEEE NorChip*, 220(4598):671–680, November 2000.

[10] W.-H. Hung, Y.-J. Chen, C.-L. Yang, Y.-S. Chang, and A. P. Su. An architectural co-synthesis algorithm for energy-aware network-on-chip design. *Proc. SAC*, March 2007.

[11] J. Hu and R. Marculescu. Energy-aware mapping for tile-based noc architectures under performance constraints. *IEEE ASP-DAC*, January 2003.

[12] J. Hu and R. Marculescu. Energy-aware communication and task scheduling for network-on-chip architecture under real-time constraints. *Proc. Design, Automation and Testing in Europe Conference and Exhibition (DATE)*, 2004.

[13] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *Proceedings of the 28th annual international symposium on Computer architecture 2001(ISCA' 01)*, 2001.

[14] S. Kirkpatrick, C. D. G. Jr., and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[15] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Öberg, K. Tiensyrjä, and A. Hemani. A network on chip architecture and design methodology. *Proc. Symposium on VLSI*, pages 117–124, April 2002.

[16] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint netwroks within the nostrum network on chip. *Proc. of 2004 Desitn, Automation and Test in Europe (DATE '04)*, March 2004.

[17] S. Murali and G. De Micheli. Bandwidth-constrained mappings of cores onto noc architectures. *Proc. 2004 Design, Automation and Test in Europe (DATE '04)*, March 2004.

[18] S. Murali and G. De Micheli. SUNMAP: A tool for automatic topology selection and generation for nocs. *Proc. 2004 Design Automation Conference (DAC '04)*, pages 914–919, 2004.

[19] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, G. De MiCheli, and L. Raffo. Designing application-specific networks on chips with floorplan information. *Proc. 2006 International Conference on Computer-Aided Design (ICCAD '06)*, 2006.

[20] D. Shin and J. Kim. Power-aware communication optimization for network-on-chips with voltage scalable links. *Proc. CODES+ISSS*, September 2004.

[21] G. C. Sih and E. A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Transactions on Parallel and Distributed Systems*, 4(2):175–187, February 1993.

[22] K. Srinivasan, K. S. Chatha, and G. Konjevod. An automated technique for topology and route generation of application specific on-chip interconnection netowkrs. *Proc. 2005 International Conference on Computer-Aided Design (ICCAD '05)*, 2005.

[23] K. Srinivasan, K. S. Chatha, and G. Konjevod. Linear-programming-based atechniques for synthesis of network-on-chip architectures. *IEEE Transactions on Very Large Scale Intergration (VLSI) Systems*, 14(4):407–420, April 2006.

[24] W. H. Wolf. Hardware-software codesign of embedded systems. *Proceedings of the IEEE*, 82(7):967–989, July 1994.

[25] W. H. Wolf. An architectural co-synthesis algorithm for distributed, embedded computing systems. *IEEE Transaction on Very Large Scale Integration (VLSI) Systems*, 5, June 1997.

[26] T. T. Ye, L. Benini, and G. De Micheli. Analysis of power consumption on switch fabrics in network routers. *Proc. of Design Automation Conference (DAC)*, pages 524–529, June 2002.

[27] MPEG2 video. IS standard. I. D. 13818-2, 2001.

[28] *ARM Processor cores.* http://www.arm.com/products/CPUs/.

[29] *Electronics. Philips' IP portfolio.* http://www.semiconductors.philips.com.

[30] *SimpleScalar*. http://www.simplescalar.com/.
[31] *Texas Instruments. Digital Signal Processing*
. http://focus.ti.com/dsp/docs/dsphome.tsp?sectionId=46.

Yi-Jung Chen received the B.S. and M.S. degrees from the Department of Computer Science and Information Engineering at National Chi Nan University, Nantou, Taiwan in 2000 and 2002, respectively. She is currently working toward the Ph.D. degree in Department of Computer Science and Information Engineering at National Taiwan University, Taipei, Taiwan.
Her research interests include high-level synthesis, Network-on-Chip design and memory hierarchy design.

Chia-Lin Yang received the B.S. degree from the National Taiwan Normal University, Taiwan, R.O.C., in 1989, the M.S. degree from the University of Texas at Austin in 1992, and the Ph.D. degree from the Department of Computer Science, Duke University, Durham, NC, in 2001.
In 1993, she joined VLSI Technology Inc. (now Philips Semiconductors) as a Software Engineer. She is currently an Associate Professor in the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan. Her research interests include energy-efficient microarchitectures, memory hierarchy design, and multimedia workload characterization.
Dr. Yang is the recipient of a 2000–2001 Intel Foundation Graduate Fellowship Award and 2005 IBM Faculty Award.

Yen-Sheng Chang received the B.S. degree in computer science and engineering from National Dong Hwa University, Hualien, Taiwan, in 2003, and the M.S. degree in computer science and engineering from National Taiwan University, Taipei, Taiwan, in 2005. His research interests include hardware-software co-design and Network-on-Chip design.

Yi-Jung Chen

Chia-Lin Yang

Yen-Sheng Chang