Power-Aware Communication Optimization for Networks-on-Chips with Voltage Scalable Links[*]

Dongkun Shin
School of Computer Science and Engineering
Seoul National University
sdk@davinci.snu.ac.kr

Jihong Kim
School of Computer Science and Engineering
Seoul National University
jihong@davinci.snu.ac.kr

ABSTRACT

Networks-on-Chip (NoC) is emerging as a practical development platform for future systems-on-chip products. We propose an energy-efficient static algorithm which optimizes the energy consumption of task communications in NoCs with voltage scalable links. In order to find optimal link speeds, the proposed algorithm (based on a genetic formulation) globally explores the design space of NoC-based systems, including task assignment, tile mapping, routing path allocation, task scheduling and link speed assignment. Experimental results show that the proposed design technique can reduce energy consumption by 28% on average compared with existing techniques.

Categories and Subject Descriptors: C.3 [Special-purpose and application-based systems]: Real-time and embedded systems

General Terms: Algorithms, Design

Keywords: Network-on-chip, Real-time systems, Low-power design

1. INTRODUCTION

Networks-on-Chip (NoC) have recently been proposed as a practical development platform for systems-on-chip (SoC) products [1, 3]. NoCs are especially useful in overcoming complex on-chip communication problems by providing a more structured and modular network interface. Since networks are structured and wired beforehand, their electrical parameters can be well controlled and optimized, which makes it possible to use aggressive signaling circuits thus significantly reducing power dissipation and propagation delay. And a standard interface between modules facilitates reusability and interoperability.

As shown in Figure 1(a), an NoC-based system is typically divided into regular tiles, where each tile might be a programmable microprocessor, an ASIC, or an FPGA. Instead of being connected by dedicated wires, each of these tiles is connected to an interconnection network that routes packets between tiles. As shown in Figure 1(b), the router in NoCs consists of input and output links, buffers and a crossbar switch.

[*]This research was supported by University IT Research Center Project.

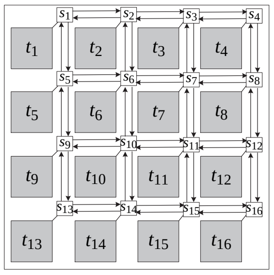
In NoC-based systems, on-chip networks take up a substantial portion of system power budget, e.g. the MIT Raw on-chip network in which the communication between 16 tiles containing processing elements consumes 36% of the total chip power [14]. And on the Alpha 21364 processor, 20% of the total chip power is consumed by the router and links [11].

One promising low-power technique for energy-efficient NoCs is to scale the speeds of the communication links with the corresponding voltage level [6]. There are two kinds of speed scaling techniques. One is an *on-line* scheme, which adjusts the communication speed dynamically, based on variations in the run-time communication traffic. The other is an *off-line* scheme which assigns an appropriate fixed communication speed to each link, based on the communication patterns of target applications. The *off-line* scheme is better suited to real-time applications since system designers are able to predict communication delays at the design time.

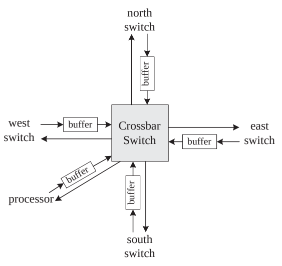
In this paper, we propose an *off-line* link speed assignment algorithm for energy-efficient NoCs with voltage scalable links. Given the task graph of a periodic real-time application, our proposed algorithm assigns an appropriate communication speed to each link, which minimizes the energy consumption of the NoC while guaranteeing the timing constraints of the real-time application. In addition, the proposed algorithm turns off links *statically* when no communications are scheduled because the leakage power of an interconnection network is not negligible (for example, 21% of the total (leakage+switching) power consumption in 0.07μm technology [2]).

As with other multiprocessor-based systems, the design flow of NoC-based systems involves several (interacting) steps, of which link speed assignment is the last step. In a typical multiprocessor system, the design flow includes two key steps, *task assignment* and *task scheduling*. Given a task graph with design constraints (e.g. the execution time and the power consumption) and processing elements (PEs), we first assign each task to an appropriate PE (*task assignment*). Then, each task is scheduled for execution within the PE (*task scheduling*). However, in NoC-based systems, two additional steps are necessary, *tile mapping* and *routing path allocation*. The tile mapping step maps a PE to one of the tiles in an NoC. The routing path allocation step determines communication paths between tiles. For example, if an NoC has sixteen PEs, as shown in Figure 1(a), then we need to decide on which tile each PE will be located. If data has to be transferred from the tile t_1 to the tile t_{16}, then we must determine which switches among $s_1, \cdots, s_{16}$ will forward the data. (In this paper, we use the term *network assignment* to refer to both the tile mapping and routing path allocation steps.)

In an NoC-based system, design decisions made in the network assignment step, as well as the task assignment and the task scheduling steps, can significantly affect the communication speed of each link, because communication traffic patterns vary according to the result of the design steps. Therefore, in order to make an NoC-



(a) An NoC with 16 tiles



(b) Structure of a router

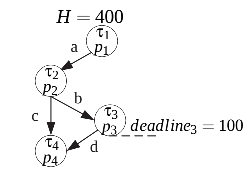
Figure 1: An architectural overview of an NoC-based system.

based system energy-efficient, each step should be taken with an awareness of its implication for energy consumption in links.

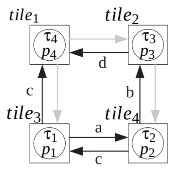
For example, consider the task graph g shown in Figure 2(a), where the tasks τ_1, τ_2, τ_3 and τ_4 are assigned to the PEs p_1, p_2, p_3 and p_4, respectively. A tile mapping algorithm might generate the network assignment shown in Figure 2(b), where a, b, c and d indicate the corresponding communication costs for the links. Assuming that the routing paths are allocated by the XY-routing algorithm [4], packets are first routed along the X-axis. Once a packet reaches the column under which the destination tile is located, it is then routed along the Y-axis. But if we change the tile mapping to the network assignment NA_2 (as shown in Figure 2(c)), we get a different energy consumption. If $b < c$, the network assignment NA_2 consumes less energy than the network assignment NA_1, because the energy consumed in the communication links are given as $(a+b+2c+d)$ and $(a+2b+c+d)$ for NA_1 and NA_2, respectively. Therefore, we can say that the network assignment NA_2 is better than NA_1.

Now let us consider how to assign the communication speed of each link in the network assignment NA_2. If the task τ_3 has a hard deadline, then the link from $tile_1$ to $tile_2$ should have a high speed because the link is on the critical path and it has to transfer the data between τ_2 and τ_4 as well as τ_2 and τ_3. This suggests that the network assignment NA_2 could be improved so that the critical path does not share links with non-critical paths. Therefore, the routing path allocation should be performed with an awareness of its effect on link speed scaling.

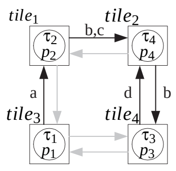
If we change the routing path of the edge (τ_2, τ_3) in NA_2 from $tile_1 \rightarrow tile_2 \rightarrow tile_4$ to $tile_1 \rightarrow tile_3 \rightarrow tile_4$, we get the network assignment NA_3, shown in Figure 2(d). Although the network schedule NA_3 has the same amount of communication traffic as the network assignment NA_2, we can now assign a lower speed to the link from $tile_1$ to $tile_2$.



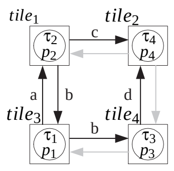
(a) an example task graph



(b) network assignment NA_1



(c) network assignment NA_2



(d) network assignment NA_3

Figure 2: A motivational example.

In this paper, we show that the existing design algorithm for NoCs is inappropriate for NoCs with voltage scalable links, and we go on to propose a novel optimization algorithm (based on a genetic formulation) which explores the overall design space efficiently. Experimental results show that the proposed design algorithm can reduce the energy consumption by 28% on average compared with the existing algorithm.

The rest of the paper is organized as follows. In Section 2, we briefly review the related works on NoC design techniques. The overall design flow and problem formulation are presented in Section 3. The detailed design techniques are described in Section 4. We present experimental results in Section 5. Section 6 concludes with a summary and directions for future work.

2. RELATED WORKS

Several research groups have investigated design techniques for minimizing the energy consumption in NoC-based systems. For example, Simunic and Boyd [12] proposed a power management technique for NoCs. Based on a network-centric power management scheme, their technique makes better predictions of future workload than techniques based on a node-centric power management approach. While that work focused on PEs, other techniques [6, 11, 13, 15, 5] have been developed with the aim of reducing the energy consumption of communication links in NoCs, because they are such a significant energy consumer [11]. Kim and Horowitz [6] proposed variable-frequency links, which can track and adjust their voltage level to the minimum supply voltage as the link frequency is changed, thus reducing the power dissipation.

Based on the variable-frequency links proposed by Kim *et al.* [6], Sang *et al.* [11] developed a history-based dynamic voltage scaling (DVS) policy which adjusts the operating voltage and clock frequency of a link according to the utilization of the link and the input buffer. Soterious *et al.* [13] proposed a simple dynamic power management technique for communication links based on the communication traffic variance to reduce the leakage power consumption. Worm *et al.* [15] proposed an adaptive low-power transmission scheme for on-chip networks. They minimized the energy required for reliable communications, while satisfying a QoS constraint by dynamically, varying the voltage on the links.

Unlike these existing techniques, that are all *on-line* schemes, our proposed technique is an *off-line* technique which assigns the appropriate constant speeds to each link. We exploit the information on communication patterns from a task graph. Hu and Marculescu [5] tackled a similar problem. Their algorithm determines a network assignment which is designed to minimize the dynamic power consumption by reducing the communication traffic. However, they did not address the issue of link speed scaling, but only the network assignment problem, assuming task assignment and task scheduling had been completed. Lei and Kumar [7] has also used the communication patterns of a task graph in tile mapping. But the objective of their algorithm is to find a tile mapping that minimizes the overall execution time of the task graph.

3. OVERALL DESIGN FLOW FOR NOCS

3.1 Specification and Architectural Model

We represent a periodic real-time application by a task graph (TG) $G =< V, E >$, which is a directed acyclic graph, where V is the set of tasks and E is the set of directed edges between tasks. In a TG, each directed edge $e(\tau_i, \tau_j)$ represents a precedence relation between τ_i and τ_j. That is, $e(\tau_i, \tau_j)$ means that the task τ_i must complete its execution before the task τ_j starts its execution. (For descriptive purposes, we will denote $e(\tau_i, \tau_j)$ by $e_{i,j}$.) The TG has a period H. A task τ_i in TG may have a deadline d_i, which must be met to ensure correct functionality of the application. Each edge $e_{i,j}$ is associated with a value $w(e_{i,j})$ which indicates the amount of communication data required between τ_i and τ_j, in the case that τ_i and τ_j are allocated in different PEs. Figure 2(a) shows an example of a task graph. Each edge e has a value of $w(e)$, and the task τ_3 has a deadline.

We denote an NoC-based system N with $m \times m$ tiles as a tuple $< T, L >$, where $T = \{t_1, \cdots, t_m, \cdots, t_{m^2}\}$ is the set of tiles and $L = \{\ell_1, \cdots, \ell_{4m(m-1)}\}$ is the set of links between tiles. All tiles are assumed to have the same area $\mathbb{A}$. We will denote the link between t_i and t_j by $\ell_{i \to j}$. We also use the notation $src(\ell_{i\to j})$ and $dst(\ell_{i\to j})$ to represent the source and destination of $\ell_{i\to j}$ respectively. For a link ℓ_i, $W(\ell_i)$ indicates the total amount of data which is transferred across the link. We denote the set of PEs as $R = \{r_1, \cdots, r_n\}$, where r_i indicates the i-th PE. We assume that the number of PEs and the number of tiles are the same, i.e., $|R| = |T|$. Each tile has associated coordinate values, $t_i.x$ and $t_i.y$ which specify row and column. (In this paper, we set $t_i.x$ and $t_i.y$ to be the quotient and the remainder of $(i-1)/m$ respectively.)

3.2 Problem Formulation

For a given task graph $G =< V, E >$, an initial step assigns each task in G into one of the available PEs. We use the function $\Phi : V \to R$ to represent this task assignment step. Task assignment affects the total communication load because only the tasks assigned into different PEs generate communication loads. Each PE is then assigned to one of tiles in the NoC. The function $\Psi : R \to T$ is used to represent this tile mapping step. The mapping also affects the total communication load because the distances between tiles are changed. The routing path between tiles is then allocated, and the function $\Omega : E \to P$ is used to denote this routing path allocation step, where P is the set of link sequences. After the routing path allocation, we set $W(\ell_i)$ for all ℓ_i in L to be $\sum_{\forall e_j, \ell_i \in \Omega(e_j)} w(e_j)$.

In this paper, we only consider a static minimal-path routing algorithm, because that is more suitable for an on-chip network and generates less communication traffic than non-minimal routing paths. The routing path allocation step does not affect the total communication load because we consider only the minimal routing path. However, since the allocation affects the communication load of each link $W(\ell)$, it also affects the speed of the links. The task scheduling step determines the execution order of tasks assigned to the same PE. Since the task scheduling step converts the task graph G to G' by inserting additional edges, we describe it with the function $O : G \to G$. Because we need to know the communication delay between tasks for task scheduling, the tile mapping and the routing path allocation steps are executed first. The link speed assignment step decides the clock speed of each link to reduce the energy consumption by utilizing what would otherwise be slack time. We use function $S : L \to C$ to denote the link speed assignment step, where C is the set of possible clock speeds for the links.

We can define the link energy optimization problem for energy-efficient NoCs as follows:

Link Energy Optimization Problem

Given $G =< V, E >, R$ and $N =< T, L >$,

find the functions Φ, Ψ, Ω, O and S such that

$$E = \sum_{\ell_i \in L} (C_L \cdot W(\ell_i) \cdot f(\ell_i)^2 + H \cdot P_{leakage}(\ell_i)) \text{ is minimized}$$

subject to $\forall \tau_i \in V,\ \theta(\tau_i) \le d_i$.

C_L is the average switching capacitance of the links. $f(\ell_i)$ and $P_{leakage}(\ell_i)$ are the clock speed and the leakage power of a link ℓ_i. For a link ℓ_i with $W(\ell_i) = 0$, $P_{leakage}(\ell_i)$ is 0, because we can turn off an inactive link. $\theta(\tau_i)$ is the end time of τ_i.

Since the task assignment may change the total computation energy consumption on PEs due to the heterogeneous architecture, we should consider both the computation energy of PEs and the communication energy of links. However, in this paper, we consider only the communication energy assuming the homogeneous PEs to concentrate on the network assignment problem.

4. ENERGY-EFFICIENT NOC DESIGN

Figure 3 shows the design flow for NoC-based systems, which consists of five optimization steps. Unfortunately, this design problem has a very large solution space, as the search space increases factorially with the number of tiles in an NoC [5]. Moreover, since all the design steps are closely related to the link speed assignment step, it is unlikely that a good speed assignment will be found by optimizing each step independently. We use three nested genetic algorithms (GAs) to explore the design space efficiently. There are a GA-based task assignment algorithm (GA-TA), a GA-based tile mapping algorithm (GA-TM) and a GA-based routing path allocation algorithm (GA-RPA).

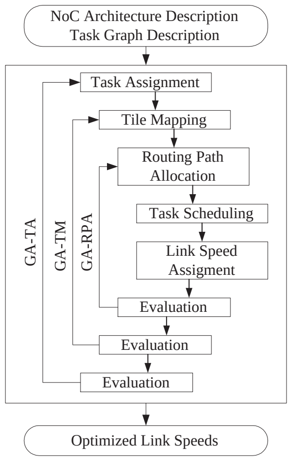


Figure 3: An overall design flow for NoCs.

Genetic algorithms imitate the principles of natural evolution to solve search and optimization problems, and are a promising technique for system-level design which has a large solution space. GA is especially suitable for multiple-objective optimization. Figure 4 shows a typical structure of a genetic algorithm. Starting with an initial population, a genetic algorithm evolves a population using the crossover and mutation operations. Since the performance of a genetic algorithm depends on the encoding, the crossover and the mutation schemes, we need to select these schemes carefully.

```
Genetic Algorithm (CTG G)
1: initialize a population with n individuals;
2: ranking each individual according to its fitness;
3: repeat {
4:     select two individuals, $p_1$ and $p_2$;
5:     child = crossover($p_1$,$p_2$);
6:     child = mutation(child);
7:     replace the lowest ranked individual with child;
8: }  until (there is no improvement during m-iterations).
```

Figure 4: A typical structure of a genetic algorithm.

4.1 GA-based Task Assignment

The more tasks that are assigned to a PE, the larger its area becomes, because it requires more memory or gates. Since each tile has the same size, the area constraint may be expressed as $A_\Phi(r_i) \le \mathbb{A}$, where $A_\Phi(r_i)$ means the area of a PE r_i under the task assignment function Φ. If there is an edge e between two tasks assigned to the same PE, its value $w(e)$ is changed to 0. This task assignment step affects the total communication load. We will represent a task assignment solution as an array of integers. For example, in Figure 5(a), the task τ_1 is mapped to r_8 in the individual p_1. Our crossover operation is the two-point crossover, which is widely used in GAs. Both parent individuals p_1 and p_2 are divided at the same two points and the child individual c_1 is generated from the first part of p_1, the second part of p_2 and the third part of p_1. Our mutation operation changes the values of randomly selected genes into new values, which make up a new individual.

4.2 GA-based Tile Mapping

We encode a tile mapping solution as an array of integers. For example, in Figure 5(b), the PE r_1 is mapped to t_8 in the individual p_1. To achieve GA-based tile mapping, we need to be careful in designing the crossover operation. If we were to use a two-point crossover for tile mapping, we would obtain illegal solutions because different PEs might be allocated into the same tile. Therefore, we use the cycle crossover [8], which is appropriate when the encoding represents a sequence. Figures 5(b)-(d) show how to make child individuals using the cycle crossover. In the mutation operation in the mapping, two randomly selected genes are exchanged to make a new individual.

For each individual, we perform routing path allocation, task scheduling and link speed assignment steps to evaluate the fitness value. To reduce the computation time, we check whether one individual is topologically identical to another individual, and omit the evaluation step if an identical individual has already been evaluated. To achieve this operation, we first transform each individual into an ordered form and compare the ordered forms of individuals. The ordered form has the following two properties: (1) the PE $\Psi^{-1}(t_1)$ has a smaller index than the indices of $\Psi^{-1}(t_m)$, $\Psi^{-1}(t_{m^2-m+1})$ and $\Psi^{-1}(t_{m^2})$, where t_1, t_m, t_{m^2-m+1} and t_{m^2} are four corner tiles; (2) the PE $\Psi^{-1}(t_m)$ has a smaller index than the index of $\Psi^{-1}(t_{m^2-m+1})$. We can transform a tile mapping into the ordered form by rotating or mirroring the tile mapping structure.

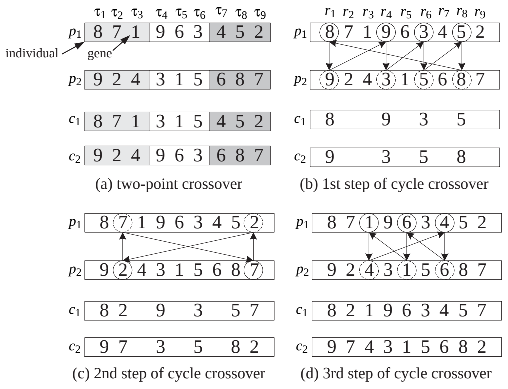


Figure 5: Crossover operations in GA-TA and GA-TM.

After tile mapping, we compose the set of communication loads, CL. For each edge $e_{i,j}$ for which $w(e_{i,j}) > 0$, we make a communication load υ which has three properties: $\upsilon_{src} = \Psi(\Phi(\tau_i))$, $\upsilon_{dst} = \Psi(\Phi(\tau_j))$, and $\upsilon_{data} = w(e_{i,j})$.

4.3 GA-based Routing Path Allocation

In routing path allocation, we assume that the source tile sends the data packet with the routing information represented by a binary number. Each bit of the binary number represents routing direction, i.e. bits 1 and 0 correspond to moves along the X-direction and Y-direction respectively. (Although there are two output links in each direction, there is no ambiguity in practice, because we assume a minimal path routing.) The intermediate routers between the source tile and the destination tile determine the forward direction from the most significant bit of routing information and forwards the routing information after shifting it by one bit. The routing path allocation step determines n directions, where the Manhattan distance between two tiles is n.

The individuals in the GA-based routing path are represented by one-dimensional arrays of binary numbers. Each binary number represents the routing path for a communication load $\upsilon \in CL$. For example, Figure 6 shows two different routing paths for the communication load υ where $\upsilon_{src} = t_1$ and $\upsilon_{dst} = t_{16}$. The routing paths p_1 and p_2 can be represented as follows:

$$p_1 = \{\ell_{1\to2}, \ell_{2\to3}, \ell_{3\to7}, \ell_{7\to11}, \ell_{11\to15}, \ell_{15\to16}\}$$
$$p_2 = \{\ell_{1\to2}, \ell_{2\to6}, \ell_{6\to7}, \ell_{7\to11}, \ell_{11\to12}, \ell_{12\to16}\}.$$

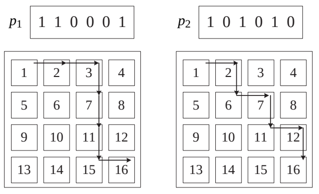


Figure 6: Routing path encoding.

By representing the routing path in this way, we can use an effective crossover and mutation operation as well as reducing the memory required for a population. When the X-distance and the Y-distance of a communication load are n and m respectively, the initial population is generated using random binary numbers which has n-number of **1**s and m-number of **0**s[1].

Routing path allocation also requires a special crossover operation. As we can see in Figure 7(a), if we were to use one-point crossover for the path routing, we might get illegal solutions. The crossover operation should guarantee that it generates only legal solutions while passing on properties from parent individuals to child individuals. In order to satisfy these requirements, we have invented a special crossover, called the coordinate crossover, for path routing. Figure 7(b) shows the coordinate crossover operation. First, we find the meeting points from two parent individuals, where two different routing paths represented by the parents meet at the meeting points. For example, in Figure 6, two routing paths represented by p_1 and p_2 meets at the tiles t_1, t_7 and t_{16}. The meeting points can be easily found by calculating the partial sum $s_i(k)$ from the first bit to the kth bit in the parent individual p_i. If two partial sums $s_1(k)$ and $s_2(k)$ of p_1 and p_2 are equal but $s_1(k-1)$ and $s_2(k-1)$ are different, the location k is a meeting point as shown in Figure 7(b). (The source and destination tiles are always meeting points.) Second, both parent individuals are divided at the meeting points and child individuals are created by mixing parts from two parents like the multi-point crossover operation. The coordinate crossover only generates legal child individuals because children have the same number of 1s as their parents. The child individuals are also guaranteed to inherit the links which both parents share. The turn model which prevents a deadlock can be sustained with the coordinate crossover.

For the mutation operation, we exchange the locations of two bits where one bit is 1 and another is 0. Using these specially designed crossover and mutation operations, we are sure to explore only the legal solution space.

[1]To prevent a deadlock in the routing path, we can use a turn model such as *west-first, north-last and negative-first* [4]. A routing path can be represented so that it follows a specific turn model.

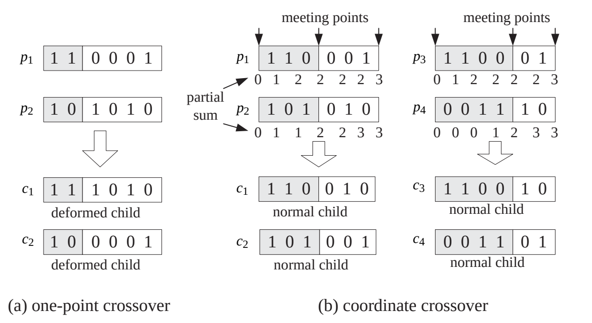


Figure 7: Crossover operation for routing.

4.4 Task Scheduling

For task scheduling, we adopted a list scheduling algorithm which uses the mobility of each task to determine its priority. The mobility of a task is defined as the difference between the ASAP start time and the ALAP end time. To get these times, we need to know the communication delay of an edge. However, we cannot know its exact value due to the asynchronous communication protocol of an NoC. Moreover, the communication delay at a link is dependent on how many communication loads share the link. So, we use the worst-case communication delay of each edge to satisfy the hard real-time constraint. To estimate the worst-case communication delay, we assumed that the worst-case delay at each link is the time required to transfer all communication loads assigned the link[2]. We can now calculate the worst-case communication delay of an edge e_i as follows:

$$\delta(e_i) = \sum_{\ell_j \in \Omega(e_i)} \frac{W(\ell_j)}{f(\ell_j) \cdot B}$$

, where B is the bandwidth of the communication links (in bits/sec). The clock speed $f(\ell_j)$ is set to maximum but a lower speed can be obtained by the following link speed assignment step.

4.5 Link Speed Assignment

For the link speed assignment, we have used a similar idea to the voltage and clock speed selection algorithm proposed by Schmitz and Al-Hashimi [10]. Their algorithm first estimates the slack time of each task considering the deadline and precedence constraint. It then calculates $\Delta E(\tau_i)$ for a task τ_i which has slack time. $\Delta E(\tau_i)$ is the energy gain when the time slot for τ_i is increased by Δt (with a lower clock speed). After increasing the time slot for the task τ_i with the largest $\Delta E(\tau_i)$, by a time increment Δt, it repeats the same sequence of steps until there is no task with slack time.

While this algorithm determines the operating speed of each task assigned on the DVS-enabled PE, our link speed assignment algorithm determines an operating speed for each link which does not change dynamically at run time. To estimate the slack time for each link, we need to consider the edges whose communication loads share that link. The slack time of a link is the minimum value among the slack times of the edges, i.e., $\xi(\ell_i) = \min_{\ell_i \in \Omega(e_j)}(\xi(e_j))$, where $\xi(\ell_i)$ and $\xi(e_j)$ are the slack times of ℓ_i and e_j respectively.

5. EXPERIMENTAL RESULTS

We started by estimating the efficiency of each optimization technique at the task assignment, tile mapping, routing path allocation and link speed assignment steps. For our experiments, we generated random task graphs $g1$ to $g16$. Figure 8 shows the energy consumptions of the communication links under various optimization configurations. The results were normalized against the energy consumption obtained by a design technique which uses random task assignment, random tile mapping, XY-routing and no link speed scaling.

The first bar for each task graph represents the result when we applied only the link speed scaling technique (opt_{LS}). The second bar represents the result when we used the GA-RPA algorithm for routing path allocation, as well as link speed scaling (opt_{LS+R}). The third bar shows the result when the GA-TM algorithm for tile mapping is also applied ($opt_{LS+R+TM}$). The fourth bar shows the energy efficiency when all optimization algorithms are used ($opt_{LS+R+TM+TA}$). The energy consumption is reduced by 8%, 17%, 27% and 43% on average by the opt_{LS}, opt_{LS+R}, $opt_{LS+R+TM}$ and $opt_{LS+R+TM+TA}$ techniques, respectively. These reduction ratios are dependent on the characteristics of the task graph (e.g., slack time and communication load) and the performance of the random configurations. For example, the link speed scaling (opt_{LS}) showed small energy reductions because the random task assignment, the random tile mapping and the XY-routing generate little slack time. From these results, we realize that all design steps have large effects on communication energy, and it is necessary to optimize the energy consumption at all design steps. (We did not compare the task scheduling step with other techniques because the list scheduling is universally popular.)

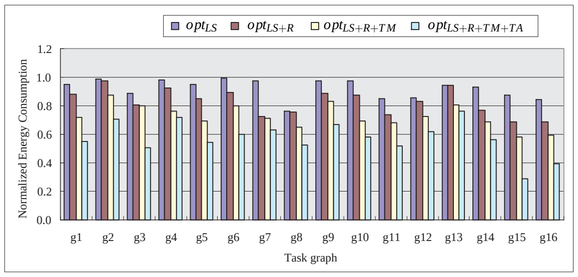


Figure 8: Effects of the nested optimization techniques.

We also compared our GA-based NoC design technique with previous techniques. Hu and Marculescu proposed a tile mapping technique that uses a branch-and-bound (B&B) algorithm and a routing path allocation algorithm that balances the communication workload across the links [5]. We expanded their algorithm by integrating it with our task assignment, task scheduling and link speed assignment algorithms. We denote the integrated algorithm as **BB** in this paper.

In **BB**, the cost of a solution is estimated from the amount of traffic, which is proportional to the dynamic power consumption of the communication links. We improved the **BB** algorithm by assuming that we can turn off a link which has no communication traffic. We call this technique as **BB**$^+$ to distinguish it from **BB**. The **BB**$^+$ technique takes into account the leakage power in estimating the cost of a solution.

The heuristic for routing path allocation is also improved in **BB**$^+$, which gives priority to the links with non-zero traffic. Using this algorithm, the number of links without traffic can be increased.

In [5], the authors showed that their algorithm can find an optimal solution within a short time. However, if we consider task scheduling and link speed assignment, **BB** and **BB**$^+$ cannot find an optimal solution because the exact cost of an internal node[3] cannot be estimated before task scheduling and link speed assignment.

[2]Although there are special on-chip network architectures for hard real-time systems [9], we have assumed no special architecture and have used an aggressive communication delay model. However, such a network architecture could be combined with our design technique by modifying the communication delay model.

[3]In the branch-and-bound technique, the *bound* step compares *UBC*, the

Figure 9 shows experimental results that compare our GA-based algorithm with **BB** and **BB**$^+$. To compare only the network assignment step, all the algorithms were executed with the same predetermined task assignment. The results were normalized againt the energy consumption of the random tile mapping and XY-routing technique. As we can see from the results, **BB** reduces the energy consumption by 16% on average but sometimes generates worse results than the random mapping (e.g., the task graph $g15$). This is because **BB** does not consider leakage power or link speed assignment. The **BB**$^+$ technique and the GA-based technique reduced the energy consumption by 22% and 39% on average, compared with the random mapping and XY-routing technique. The GA-based technique reduced the energy consumption by 28% on average, compared with the **BB** technique.

We also experimented with the task graph of the real application (a multimedia system with an H.263 encoder/decoder and an MP3 encoder/decoder) introduced previously [5]. Since each task is assigned to a processing element in the task graph, we only evaluated the tile mapping and routing path allocation steps. Our GA-based algorithm reduced the energy consumption by 35% compared with the random tile mapping and XY-routing technique.

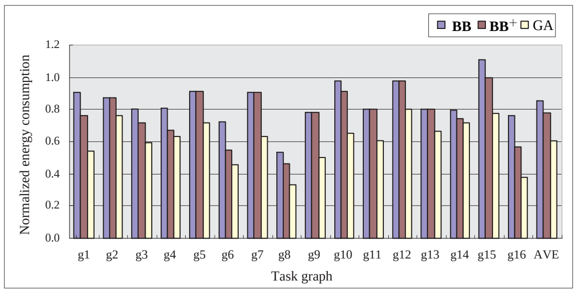


Figure 9: Performance comparison between the B&B and GA techniques.

Table 1 shows the features of the task graphs and the execution times of the $opt_{LS+R+TM}$ algorithm running on a Pentium-III 500 MHz Linux machine. The execution time of the genetic algorithm depends on various design factors, such as the population size, the mutation probability and the termination condition. We initialized the population sizes as $|T|^2/2$ and $|T|$ in GA-TM and GA-RPA respectively. For our experiments, the number of tiles, $|T|$, was 9 (3×3) or 16 (4×4). From Table 1, we see that the GA-based algorithm takes a long time when a task graph has a large number of edges, i.e., there are many communication loads. Exceptions such as the task graphs $g3$ and $g14$ are due to the nature of the link speed assignment algorithm. The link speed assignment algorithm iterates, increasing the delay time of a link by Δt until there is no slack time. So it takes longer as a task graph has long slack times. So we can improve the speed of the algorithm with a fast link speed assignment module. The **BB** and **BB**$^+$ performed well when $|T| = 9$, but become very slow when $|T| = 16$. Without special speedup techniques, **BB** takes over 1 hour for the graph $g2$ when $|T| = 16$. But our GA-based algorithm take under a minute, even for quite complex task graphs, which makes it acceptable for a design methodology.

6. CONCLUSIONS

In this paper, we have proposed an energy-efficient algorithm to optimize the communication energy consumption in NoC-based

time (sec)

TG	N_{node}/N_{edge}	3×3	4×4	TG	N_{node}/N_{edge}	3×3	4×4
g1	26/43	5.2	8.4	g9	30/29	6.5	43.9
g2	40/77	9.3	35.5	g10	36/50	12.5	57.9
g3	20/33	9.6	38.8	g11	37/36	9.0	26.3
g4	40/77	11.6	38.6	g12	24/33	5.0	9.6
g5	20/26	5.7	20.1	g13	31/56	9.9	58.6
g6	20/27	3.6	13.8	g14	29/56	6.2	19.3
g7	18/26	5.1	15.8	g15	12/15	3.2	9.4
g8	16/15	3.2	12.2	g16	14/19	2.8	4.2

* N_{node}= # of nodes and N_{edge}= # of edges

Table 1: Execution times of $opt_{LS+R+TM}$ algorithm.

systems with voltage scalable links. The proposed algorithm optimizes communication energy at the various design steps, i.e., task assignment, tile mapping, routing path allocation and link speed assignment. We used genetic algorithms to explore the large design space of energy-efficient NoCs effectively. Our algorithm reduced the energy consumption of on-chip network by 39% on average compared with an algorithm which uses random tile mapping and XY-routing.

Our work can be extended in several directions. Due to the asynchronous communication protocol, we estimated the worst-case communication delay pessimistically. However, by considering the precedence dependency in the task graph, we could obtain a tighter bound on the communication delay. In particular, we can find communication loads that never overlap by analyzing the task graph. Another issue is the buffer size of each router. Since the required buffer size changes depending on the tile mapping and the routing path allocation, it is necessary to design NoC-based systems under a buffer size constraint.

7. REFERENCES

[1] L. Benini and G. De Micheli. Networks on Chip: A New SoC Paradigm. *IEEE Computer*, 35(1):70–78, 2002.

[2] X. Chen and L.-S. Peh. Leakage Power Modeling and Optimization in Interconnection Networks. In *Proc. International Symposium on Low Power Electronics and Design*, pages 90–95, 2003.

[3] W. J. Dally and B. Towles. Route Packets, Not Wires: On-chip Interconnection Networks. In *Proc. Design Automation Conference*, pages 684–689, 2001.

[4] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks*. Morgan Kaufmann, 1997.

[5] J. Hu and R. Marculescu. Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures. In *Proc. Design, Automation and Test in Europe Conference*, pages 10688–10693, 2003.

[6] J. Kim and M. Horowitz. Adaptive Supply Serial Links with Sub-1V Operation and Per-pin Clock Recovery. In *Proc. International Solid-State Circuits Conference*, 2002.

[7] T. Lei and S. Kumar. A Two-step Genetic Algorithm for Mapping Task Graphs to a Network on Chip Architecture. In *Proc. Euromicro Symposium on Digital Systems Design*, pages 180–187, 2003.

[8] I. Oliver, D. Smith, and J. Holland. A Study of Permutation Crossover Operations on the Traveling Salesman Problem. In *Proc. International Conference on Genetic Algorithm*, pages 224–230, 1987.

[9] J. Rexford, J. Hall, and K. G. Shin. A Router Architecture for Real-Time Point-to-Point Networks. In *Proc. International Symposium on Computer Architecture*, pages 237–246, 1996.

[10] M. T. Schmitz and B. M. Al-Hashimi. Considering Power Variations of DVS Processing Elements for Energy Minimisation in Distributed Systems. In *Proc. International Symposium on System Synthesis*, pages 250–255, 2001.

[11] L. Shang, L.-S. Peh, and N. K. Jha. Dynamic Voltage Scaling with Links for Power Optimization of Interconnection Networks. In *Proc. International Symposium on High-Performance Computer Architecture*, 2003.

[12] T. Simunic and S. Boyd. Managing Power Consumption in Networks on Chips. In *Proc. Design, Automation, and Test in Europe*, pages 110–116, 2002.

[13] V. Soteriou and L.-S. Peh. Dynamic Power Management for Power Optimization of Interconnection Networks Using On/Off Links. In *Proc. Symposium on High Performance Interconnects*, pages 15–20, 2003.

[14] H.-S. Wang, L.-S. Peh, and S. Malik. Power-Driven Design of Router Microarchitectures in On-Chip Networks. In *Proc. International Symposium on Microarchitecture*, pages 105–116, 2003.

[15] F. Worm, P. Ienne, P. Thiran, and G. De Micheli. An Adaptive Low Power Transmission Scheme for On-chip Networks. In *Proc. International System Synthesis Symposium*, pages 92–100, 2002.

best cost of leaf nodes generated up to the current time, with *LBC*, the lower-bound cost of the leaf nodes which are still to be generated from the current internal node.