Energy-Aware Mapping for Tile-based NoC Architectures Under Performance Constraints *

Jingcao Hu Radu Marculescu
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213-3890, USA
e-mail: {jingcao, radum}@ece.cmu.edu

Abstract — In this paper, we present an algorithm which automatically maps the IPs/cores onto a generic regular Network on Chip (NoC) architecture such that the total communication energy is minimized. At the same time, the performance of the mapped system is guaranteed to satisfy the specified constraints through bandwidth reservation. As the main contribution, we first formulate the problem of energy-aware mapping, in a topological sense, and then propose an efficient branch-and-bound algorithm to solve it. Experimental results show that the proposed algorithm is very fast and robust, and significant energy savings can be achieved. For instance, for a complex video/audio SoC design, on average, 60.4% energy savings have been observed compared to an ad-hoc implementation.

I. Introduction

With the advance of the semiconductor technology, the enormous number of transistors available on a single chip allows designers to integrate dozens of IP blocks together with large amounts of embedded memory. These IPs can be CPU or DSP cores, video stream processors, high-bandwidth I/O, *etc*[1]. The richness of the computational resources places tremendous demands on the communication resources as well. Additionally, the shrinking feature size in the *deep-sub-micron* (DSM) era is continuously pushing interconnection delay and power consumption as the dominant factors in the optimization of modern systems. Another consequence of the DSM effects is the difficulty in optimizing the interconnection because of the ensued worsening effects such as crosstalk, *etc*.

To mitigate these problems, Dally and Towles [2] have recently proposed a *regular* tile-based architecture where communication can be efficiently realized using an on-chip network (Fig. 1)[1]. As shown in the left part of Fig. 1, the chip is divided into regular tiles where each tile can be a general-purpose processor, a DSP, a memory subsystem, *etc*. A router is embedded within each tile with the objective of connecting it to its neighboring tiles. Thus, instead of routing design-specific global wires, the inter-tile communication can be achieved by routing packets via these embedded routers.

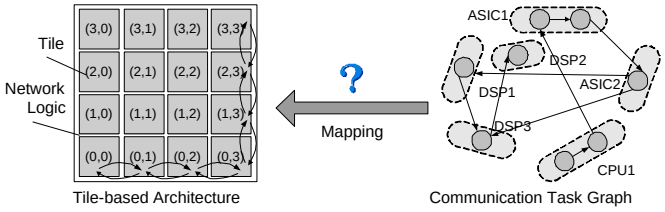

Fig. 1. Tile-based architecture and the mapping problem

Three key concepts come together to make this tile-based architecture very promising: structured network wiring, modularity and standard interfaces. More precisely, since the network wires are structured and wired beforehand, their electrical parameters can be very well controlled and optimized. In turn, these controlled electrical parameters make possible to use aggressively signaling circuits which reduce power dissipation and propagation delay significantly. Modularity and standard network interfaces facilitate re-usability and interoperability of the modules. Moreover, since the network platform can be designed in advance and later used for many applications, it makes sense to highly optimize this platform as its development cost can be amortized across many applications.

To exploit this regular tile-based architecture, the design flow needs the following three steps: First, the application needs to be divided into a graph of concurrent tasks. Second, using a set of available IPs, the application tasks are assigned and scheduled. Finally, the designer needs to decide to which tile each selected IP should be mapped such that the metrics of interest are optimized. More precisely, given the assigned/scheduled task graph which has been generated from previous two steps, this last phase determines the *topological placement* of these IPs onto different tiles. For instance, referring to Fig. 1, this step determines onto which tile (e.g. (3,0), (2,1), (1,3) *etc.*) each IP (e.g. ASIC2, DSP3, CPU1, *etc.*) should be placed.

The first two steps described above are not new to the CAD community, as they have been addressed in the area of hardware/software co-design and IP-reuse [3]. However, the *mapping* phase (that is, the *topological placement* of the IPs onto the on-chip tiles) represents a new problem, especially in the context of the regular tile-based architecture, as it significantly

*Research supported by NSF CCR-00-93104 and DARPA/Marco Gigascale Research Center (GSRC), and SRC 2001-HJ-898.

[1]This implementation is slightly different from the example implementation given in [2], where a torus topology is adopted.

impacts the energy and performance metrics of the system. In this paper, we address this very issue. To this end, we first formulate the mapping problem and show the impact of different mappings on the communication energy consumption of a given system. An efficient branch-and-bound algorithm is then proposed to solve this problem under tight performance constraints. Experimental results show that significant energy savings can be achieved, while guaranteeing the specified system performance. Compared to a simulated annealing algorithm, our algorithm is orders of magnitude faster, while the energy consumption of the solution is almost the same (less than 10% difference).

The paper is organized as follows: Section II briefly introduces the related work. The platform of the targeted system and its associated power model are described in Section III. Sections IV and V illustrate the energy-aware mapping algorithm. Experimental results are shown in Section VI. Finally, Section VII summarizes our contribution and outlines some directions for future work.

II. Related Work

In their paper [2], Dally *et al.* suggest using the on-chip interconnection networks instead of ad-hoc global wiring to structure the top-level wires on a chip and facilitate modular design. In [4], Hemani *et al.* present a honeycomb structure in which each processing core (resource) is located on a regular hexagonal node connected to three switches while these switches are directly linked to their next nearest neighbors. In [5], Kumar *et al.* describe a physical NoC architecture implemented by a direct layout of a $2D$ mesh of switches and resources.

Although different in topology and some other aspects, all the above papers essentially advocate the advantages of using NoCs and regularity as effective means to design high performance SoCs. While these papers mostly focus on the concept of regular NoC architecture (discussing the overall advantages and challenges), to the best of our knowledge, our work is the *first* to address the mapping problem for tile-based architecture and provide an efficient way to solve it.

III. Platform Description

In this section, we describe the regular tile-based architecture and the power model associated to the communication network.

A. The Architecture

The chip under consideration in this paper is composed of $n \times n$ tiles which are inter-connected by a $2D$ mesh network[2]. Fig. 2 shows an abstract view of a tile in this architecture.

As shown in Fig. 2, each tile is composed of a *processing core* and a *router*. The router embedded onto each tile

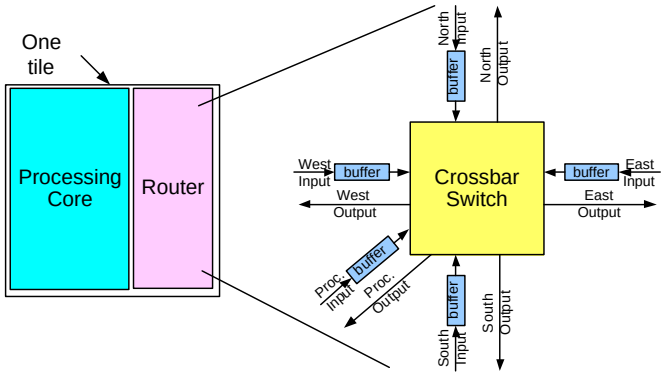

Fig. 2. The typical structure of a tile

is connected to the four neighboring tiles and its local processing core via channels. Each channel consists of two *one-directional point-to-point* links between two routers or a router and a local processing core.

Compared to typical macro-networks, an on-chip network is by far more resource limited. To minimize the implementation cost, the on-chip network should be implemented with very little area overhead. This is especially important for those architectures composed of tiles with fine-level granularity. Thus, instead of having huge memories (e.g. SRAM or DRAM) as buffer space for those routers/switches in the macro-network, it's more reasonable to use registers as buffers for on-chip routers[3]. For the architecture in Fig. 2, a 5×5 crossbar switch is used as the switching fabric because of its nice cost/performance trade-offs for switches with small number of ports.

To be able to direct the information appropriately, a tile-based architecture requires a method of routing the data packets through the network. There are quite a few routing algorithms proposed so far. In general, they can be divided into two categories: static routing and adaptive routing [6]. For the tile-based architecture, we believe that the static routing is more suitable than the adaptive routing because:

1. Compared to static routers, implementing adaptive routers requires by far more resources because of their complexity.
2. Since in adaptive routing packets may arrive out of order, huge buffering space is needed to reorder them. This, together with the protocol overhead, leads to prohibitive cost overhead, extra delay and jitter.

Based on the above considerations, *static XY* routing is assumed for the on-chip network. In a few words, for $2D$ mesh networks, *XY* routing first routes packets along the *X*-axis. Once it reaches the column wherein lies the destination tile, the packet is then routed along the *Y*-axis. Obviously, *XY* routing is a *minimal* path routing algorithm and is *free* of deadlock and livelock [6].

[2]We use the $2D$ mesh network simply because it naturally fits the tile-based architecture. However, our algorithm can be extended for other topologies.

[3]As we will see later, this leads to a much simpler power model compared to its macro-network peer.

B. The Energy Model

In [7], Ye *et al.* propose a new model for evaluating the power consumption of switch fabrics in network routers. To this end, the *bit energy* (E_{bit}) metric is defined as the energy consumed when one bit of data is transported through the router. E_{bit} can be calculated by the following equation:

$$E_{bit} = E_{S_{bit}} + E_{B_{bit}} + E_{W_{bit}} \quad (1)$$

where $E_{S_{bit}}$, $E_{B_{bit}}$ and $E_{W_{bit}}$ represent the energy consumed by switch, buffering and interconnection wires, respectively. (Note that the authors in [7] assume the buffers are implemented in SRAM or DRAM.)

Although the above power model is targeted for network routers where the *entire* chip is occupied by just *one* router, it can be adapted to the tile-based architecture with the following modifications:

• First, in [7], $E_{B_{bit}}$ becomes dominant when congestion happens since accessing and refreshing the memory are very expensive in terms of power consumption. This is no longer true for on-chip networks where the buffers are implemented using regular registers.

• Second, in [7], $E_{W_{bit}}$ is the energy consumed on the wires inside the switch fabric. For the on-chip network, the energy consumed on the links between tiles should also be included; in the following this is denoted by $E_{L_{bit}}$. Thus, the average energy consumed in sending one bit of data from a tile to its neighboring tile can be calculated as:

$$E_{bit} = E_{S_{bit}} + E_{B_{bit}} + E_{W_{bit}} + E_{L_{bit}} \quad (2)$$

Since the link between each pair of nodes is typically in the order of mm, the energy consumed by buffering and internal wires is negligible[4] compared to $E_{L_{bit}}$ ($E_{B_{bit}} + E_{W_{bit}} \ll E_{L_{bit}}$). Thus, Eq. (2) reduces to:

$$E_{bit} = E_{S_{bit}} + E_{L_{bit}} \quad (3)$$

Consequently, the average energy consumption of sending one bit of data from tile t_i to tile t_j can be calculated as:

$$E_{bit}^{t_i,t_j} = n_{hops} \times E_{S_{bit}} + (n_{hops} - 1) \times E_{L_{bit}} \quad (4)$$

where n_{hops} is the number of routers the bit passes on its way along a path.

Eq. (4) gives the energy model for the regular tile-based NoC architecture. Without loss of generality, in what follows, we focus on $2D$ mesh network. Note that, for the $2D$ mesh network with *XY* routing, Eq. (4) shows that the average energy consumption of sending one bit of data from tile t_i to tile t_j is determined by the *Manhattan* distance between these two tiles.

IV. The Problem of Energy-Aware Mapping

A. Problem Formulation

Simply stated, our objective is to figure out, after the designer has selected a set of IPs and assigned/scheduled the tasks onto these IPs, how to map these IPs onto different tiles such that the total communication energy consumption is minimized, while guaranteeing the performance of the system. To formulate this problem in a more formal way, we need to first introduce the following two new concepts:

Definition 1 An *Application Characterization Graph* (APCG) $\mathcal{G} = G(C, A)$ is a *directed* graph, where each vertex c_i represents one selected IP/core, and each directed arc $a_{i,j}$ represents the communication from c_i to c_j. The following quantities are associated with each $a_{i,j}$ as arc properties:

- $v(a_{i,j})$: arc volume from vertex c_i to c_j, which stands for the communication volume (*bits*) from c_i to c_j.
- $b(a_{i,j})$: arc bandwidth requirement from vertex c_i to c_j, which stands for the minimum bandwidth (*bits/sec.*) that should be allocated by the communication network.

Definition 2 An *Architecture Characterization Graph* (ARCG) $\mathcal{G}' = G(T, P)$ is a *directed* graph, where each vertex t_i represents one tile in the architecture, and each directed arc $p_{i,j}$ represents the routing path from t_i to t_j[5]. The following quantities are associated with each $p_{i,j}$ as arc properties:

- $e(p_{i,j})$: arc cost from vertex t_i to t_j, which represents the average energy consumption (*joule*) of sending one bit of data from tile t_i to t_j, i. e., $E_{bit}^{t_i,t_j}$.
- $L(p_{i,j})$: the set of links that make up the path $p_{i,j}$.

Using the above graph representations, the problem of minimizing the communication energy consumption under performance constraints can be formulated as:

Given an APCG and an ARCG that satisfy

$$size(APCG) \leq size(ARCG) \quad (5)$$

find a mapping function $map()$ from APCG to ARCG which minimizes:

$$\min\{Energy = \sum_{\forall a_{i,j}} v(a_{i,j}) \times e(p_{map(c_i),map(c_j)})\} \quad (6)$$

such that:

$$\forall c_i \in C, \quad map(c_i) \in T \quad (7)$$

$$\forall c_i \neq c_j \in C, \quad map(c_i) \neq map(c_j) \quad (8)$$

$$\forall \text{ link } l_k, B(l_k) \geq \sum_{\forall a_{i,j}} b(a_{i,j}) \times f(l_k, p_{map(c_i),map(c_j)}) \quad (9)$$

where $B(l_k)$ is the bandwidth of link l_k, and:

$$f(l_k, p_{m,n}) = \begin{cases} 0 & : \quad l_k \notin L(p_{m,n}) \\ 1 & : \quad l_k \in L(p_{m,n}) \end{cases}$$

Conditions (7) and (8) mean that each IP should be mapped to exactly one tile and no tile can host more than one IP. Eq. (9) guarantees that the load of any link will not exceed its bandwidth.

[4] We implemented a 4×4 crossbar switch and then evaluated its power consumption with Synopsys design compiler for a $0.18\mu m$ technology. The results show that $E_{B_{bit}} = 0.075pJ$, which is indeed negligible compared to $E_{L_{bit}}$ (typically in the order of pJ).

[5] For $2D$ mesh network with static routing, this suggests a *complete* connected graph with *exactly one* arc from each vertex to any other vertex.

B. Significance of the Problem

To prove that the mapping heavily affects the communication energy consumption, we carried out the following experiment. A series of task graphs are generated using TGFF [8]. Then the output graph is randomly assigned to a given number of IPs, with the computational times and communication volumes randomly generated according to the specified distribution. Our tool is then used to pre-process and annotate these task graphs and build the *Communication Task Graphs* (CTG), which characterizes the partitioning, task assignment, scheduling, communication patterns, task execution time, of the application. Also, the bandwidth requirements between any communicating IP pairs are calculated.

The number of IPs used in the experiment ranges from 3×3 to 13×13. For each benchmark, we randomly generate 3000 mapping configurations and the corresponding energy consumption values are calculated. In parallel, an optimizer using *simulated annealing* (SA) was also developed with the goal of finding a legal mapping which consumes the least amount of communication energy. The resulting energy ratios are plotted in Fig. 3.

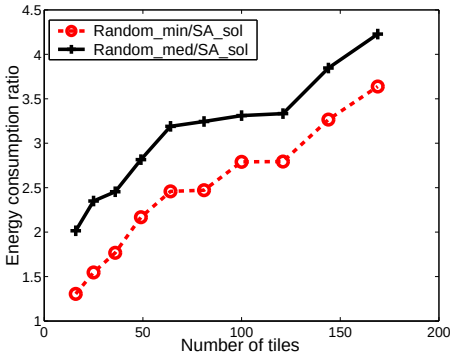

Fig. 3. The impact of mapping on energy consumption

The dashed line in Fig. 3 shows the energy consumption ratio of the best solution among the 3000 random mappings (*Random_min*) to the solution found by the simulated annealing (*SA_sol*). The solid line shows the ratio of the median solution among the 3000 random mappings (*Random_med*) to *SA_sol*.

As we can see, although the simulated annealing optimizer does not necessarily find the optimal solution, it still saves around 50% energy compared to the median solution for the system with 3×3 tiles. Moreover, the savings increase as the system size scales up. For systems with 13×13 tiles, the savings can be as high as 75%. Another observation is that the best solution among the 3000 random mappings is far from satisfactory, even with a system as small as 3×3 tiles.

Unfortunately, the mapping problem is an instance of constrained *quadratic assignment problem* which is known to be *NP-hard* [9]. The search space of the problem increases *factorially* with the system size. Even for a system with 4×4 tiles, there can be 16! mappings which are already impossible to enumerate, not to mention systems with 10×10 tiles that are anticipated in five years or less [5]. In the following section, we propose an efficient *branch-and-bound* algorithm which can be used to find nearly optimal solutions in reasonable run times.

V. The Algorithm of Energy-Aware Mapping

A. The Data Structure

Our approach is based on a *branch-and-bound* algorithm. The algorithm is used to efficiently walk through the *search tree* which represents the whole searching space. Fig. 4 shows an example of the searching tree for mapping an application with 4 IPs onto a 2×2 tile architecture. To keep the figure simple, we do not show all the nodes.

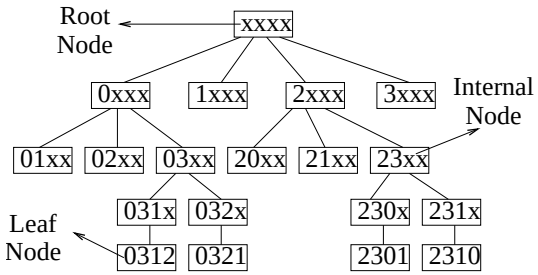

Fig. 4. An example search tree

In Fig. 4, each node belongs to one of the three categories: *root* node, *internal* node, and *leaf* node. The root node corresponds to the state where *no* IP has been mapped. Each internal node represents a *partial* mapping which is tagged by a label. Each number in the label represents which tile the corresponding IP is mapped to. For example, the node with the label "$23xx$" represents a partial mapping where IP_0 and IP_1 are mapped to $Tile_2$ and $Tile_3$ respectively, while IP_2 and IP_3 are still unmapped. Each leaf node represents a *complete* mapping of the IPs to the tiles.

To explain how our algorithm works, the following terms need to be defined:

Definition 3 The *cost* of a node is the energy consumed by the communication among those IPs that have already been mapped.

For instance, the cost of the node labeled "$23xx$" can be calculated as $v(a_{0,1}) \times e(p_{2,3}) + v(a_{1,0}) \times e(p_{3,2})$.

We can infer from definition 3 that any child node's cost is no less than its parent node's cost. This property will later be used in the algorithm to trim away unqualified sub-trees.

Definition 4 Let $\mathcal{M}$ be the set of vertices in the APCG that have already been mapped. A node is called a *legal* node if and only if, for any link l_k, it satisfies the following condition:

$$B(l_k) \geq \sum_{\forall a_{i,j}, c_i c_j \in \mathcal{M}} b(a_{i,j}) \times f(l_k, p_{map(c_i),map(c_j)}) \quad (10)$$

Eq. (10) guarantees that all the bandwidth requirements between the currently mapped IPs are satisfied. Also, if a node is illegal, then all of its descendant nodes are illegal.

Definition 5 The *Upper Bound Cost* (UBC) of a node is defined as a value that is no less than the *minimum* cost of its *legal*, descendant *leaf* nodes.

Definition 6 The *Lower Bound Cost* (LBC) of a node is defined to be the *lowest* cost that its descendant leaf nodes can *possibly* achieve.

Differently stated, this means that if a node has the LBC equal to x, then each of its descendant leaf nodes has at least a cost of x.

B. The Branch-and-Bound Algorithm

Given the above definitions, finding the optimal mapping is equivalent to finding the *legal leaf* node which has the *least* cost[6]. To achieve this, our algorithm searches the optimal solution by alternating the following two steps:

Branch: In this step, an unexpanded node is selected from the tree, the next *unmapped* IP is enumeratively assigned to the set of remaining *unoccupied* tiles and then the corresponding new child nodes are generated.

Bound: Each of the newly generated child nodes are inspected to see if it is possible to generate the best leaf nodes later. A node can be trimmed away without further expansion if either its cost or its LBC is higher than the *lowest* UBC that has been found during the searching (since it is guaranteed that other nodes will eventually lead to a better solution).

Obviously, the calculation of the UBC and LBC significantly impacts the speed of the algorithm. Primarily, we want to have tight UBC and LBC for each node so that more non-promising nodes can be detected and trimmed away early on during the search. Unfortunately, calculating a tight UBC or LBC usually demands more computational time. Next, we describe our method for computing UBC and LBC, which offers a satisfactory trade-off between the average time for processing one node and the number of nodes that need to be processed.

• *UBC calculation*

By definition 5, the cost of any *legal* descendant leaf node can be used as the UBC of that node. Since we want to select the *legal* descendant leaf node with the *smallest* cost, we choose the descendant leaf node using a greedy method for mapping the remaining unmapped IPs to the unoccupied tiles. For each step in the greedy mapping procedure, the next unmapped IP c_k with the highest communication demand is selected and its *ideal* topological location (x, y) on the chip is calculated as:

$$x = \frac{\sum_{\forall c_i \in \mathcal{M}} \left(v(a_{k,i}) + v(a_{i,k})\right) \times c_i^x}{\sum_{\forall c_i \in \mathcal{M}} \left(v(a_{k,i}) + v(a_{i,k})\right)} \quad (11)$$

$$y = \frac{\sum_{\forall c_i \in \mathcal{M}} \left(v(a_{k,i}) + v(a_{i,k})\right) \times c_i^y}{\sum_{\forall c_i \in \mathcal{M}} \left(v(a_{k,i}) + v(a_{i,k})\right)} \quad (12)$$

where c_i^x and c_i^y represent the row id and column id of the tile that c_i is mapped onto, respectively, and $\mathcal{M}$ is the set of mapped IPs which is updated at each step. c_k is then mapped to an unoccupied tile whose topological location has the smallest *Manhattan* distance to (x, y).

[6]The performance constraints are guaranteed to be satisfied by the legality of the node.

This step is repeated until all IPs have been mapped. This leads to a complete mapping and thus identifies a single descendant leaf node. If this leaf node is illegal, then the UBC of the node under inspection is set to be infinitely large; otherwise, it is set to be the cost of that descendant leaf node.

• *LBC calculation*

The LBC cost of a node n can be decomposed into *three* components, as shown in Eq. (13):

$$LBC = C_{m,m} + C_{u,u} + C_{m,u} \quad (13)$$

$C_{m,m}$ is the cost of the intercommunication among mapped IPs. Since the location of these IPs is known, $C_{m,m}$ can be calculated exactly. $C_{u,u}$ is the cost of the intercommunication among unmapped IPs. Eq. (14) is used to calculate $C_{u,u}$, where $\bar{\mathcal{M}}$ stands for the set of unmapped IPs and $\bar{\mathcal{O}}$ stands for the set of tiles that have not been occupied yet.

$$C_{u,u} = \frac{1}{2} \times \sum_{\forall c_i \in \bar{\mathcal{M}}} \sum_{\forall c_j \in \bar{\mathcal{M}}} v(a_{i,j}) \times \min_{\forall t_m t_n \in \bar{\mathcal{O}}} e(p_{m,n}) \quad (14)$$

The last item $C_{m,u}$ stands for the cost of the intercommunication between the mapped IPs and the unmapped IPs. Let $\mathcal{M}$, $\bar{\mathcal{M}}$ and $\bar{\mathcal{O}}$ be the sets of mapped IPs, unmapped IPs and unoccupied tiles, respectively. $C_{m,u}$ can be derived by:

$$\begin{aligned} C_{m,u} = & \sum_{\forall c_i \in \mathcal{M}} \sum_{\forall c_j \in \bar{\mathcal{M}}} v(a_{i,j}) \times \min_{\forall t_k \in \bar{\mathcal{O}}} e(p_{map(c_i),k}) \\ & + \sum_{\forall c_i \in \bar{\mathcal{M}}} \sum_{\forall c_j \in \mathcal{M}} v(a_{i,j}) \times \min_{\forall t_k \in \bar{\mathcal{O}}} e(p_{k,map(c_j)}) \end{aligned} \quad (15)$$

C. Speed-up Techniques

In order to speed up the searching process, it is critical to trim away as many non-promising nodes as possible, as early as possible during the search process. We propose the following techniques for this purpose.

• *IP ordering*: We can sort the IPs according to their communication demand[7] so that the IPs with higher demand will be mapped earlier. Since the positions of the IPs with higher demand generally have a larger impact on the overall communication energy consumption than those of IPs with lower demand, fixing their positions earlier helps exposing those non-promising internal nodes at earlier times in the searching; this reduces the number of nodes to be expanded. As most applications have non-uniform traffic patterns, this heuristic is quite useful in practice.

• *Priority queue (PQ)*: A priority queue is used to sort those nodes that are waiting to be branched based on their cost. The lower the cost of the node, the higher the priority the node has for branching. Intuitively, expanding a node with lower cost will more likely decrease the minimum UBC so that more non-promising nodes may be detected.

• *Symmetry Exploitation*: To further speed up our algorithm, the symmetry property of the architecture is exploited. Considering the system with 16 tiles and nodes of depth 1 in the search tree as an example, we only need to investigate those nodes

[7]For IP c_i, this is calculated as $\sum_{\forall j \neq i} \{v(a_{i,j}) + v(a_{j,i})\}$

which map the first IP to the tiles denoted by $(0,0)$, $(0,1)$ and $(1,1)$ (see Fig. 1), as the other nodes are just mirrors of these nodes.

D. The Pseudo code

Fig. 5 gives the pseudo code of our algorithm which also shows how the above speed-up techniques are employed.

```
Sort the IPs by communication demand
root_node = new node(NULL)
min_UBC = +∞, best_mapping_cost = +∞
PQ.Insert(root_node)
while(!PQ.Empty()) {
   cur_node = PQ.Next()
   for each unoccupied tile t_i {
     generate child node n_new
     if(n_new's mirror node exists in the PQ)
       continue
     if(n_new.LBC>min_UBC) continue
     if(n_new.isLeafNode) {
       if(n_new.cost < best_mapping_cost) {
         best_mapping_cost = n_new.cost
         best_mapping = n_new }}
     else {
       if(n_new.UBC<min_UBC) min_UBC = n_new.UBC
       PQ.insert(n_new) }}
}
```

Fig. 5. The pseudo code of the algorithm

Obviously, the code shown in Fig. 5 will always find the optimal solution. However, as the system size scales up, the run time of this algorithm will also increase drastically. Thus, the following heuristic needs to be used to trade-off the solution quality with run time.

The length of the PQ is monitored during the process. When it reaches a threshold value, strict criteria are applied to select the child nodes for insertion into the queue. Suppose we are currently expanding node n. If n is the node in the PQ which has the minimal UBC, then all of its child nodes will be evaluated by the code in Fig. 5 for insertion into the PQ. Otherwise, only the child with the *lowest* cost among its siblings and the child generated by the greedy mapping will be evaluated for insertion.

VI. Experimental Results

A. Evaluation Experiments

We first compare the run-time and quality of the solution generated by our algorithm to a *simulated annealing* optimizer (SA)[8]. Four categories of benchmarks are generated by the technique described in subsection IV.B. Categories I, II, III and IV contain 10 applications with 9, 16, 25 and 36 IPs, respectively. These need to be mapped onto architectures with the same number of tiles.

[8]To make the comparison fair, SA was optimized by carefully selecting parameters such as number of moves per temperature, cooling schedule, *etc*.

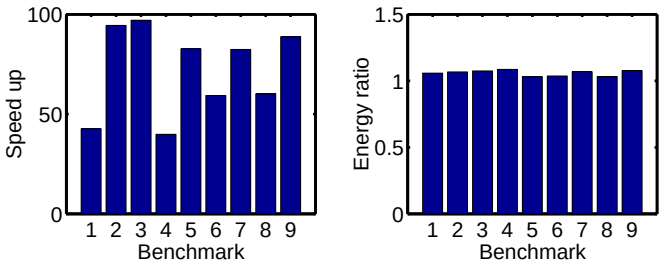

Fig. 6. Comparison between SA and our algorithm for category III benchmarks

Fig. 6 shows the comparison between our algorithm and simulated annealing for the benchmarks in category III. The left figure gives the speed up ratios of our algorithm over the simulated annealing algorithm. The right figure shows the energy ratios of the solutions provided by our algorithm to that generated using simulated annealing. Note that although we have 10 benchmarks for category III, we only show the results for 9 benchmarks here since neither of them can find a mapping solution which meets the specified performance constraints for one of the benchmarks.

As it can be seen, our algorithm runs much faster (72 times on average) over SA with very competitive solutions (the difference of the communication energy consumption between the solutions generated by these two algorithms are within 6%, on average).

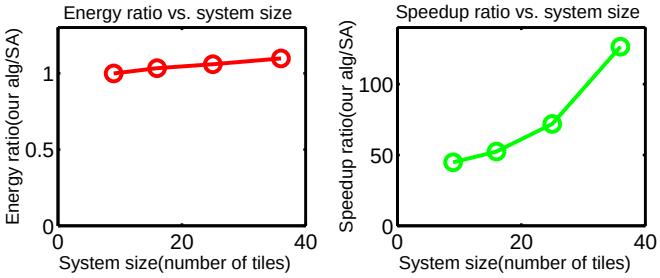

Fig. 7. Comparison between SA and our algorithm with system size scales up

Fig. 7 shows how our algorithm performs compared to SA as the system size scales up. For benchmark applications using 36 tiles, our algorithm runs 127 times faster than SA, on average. Meanwhile, the solutions produced by our algorithm remain very competitive compared to those generated by SA. On average, the energy consumption of the solution generated by our algorithm is only 3%, 6% and 10% for category II, III and IV, respectively. For category I, our algorithm can even find better solutions than SA because it can in general walk through the whole search tree due to the small size of the problem.

B. A Video/Audio Application

To evaluate the potential of our algorithm for *real* applications, we applied this algorithm to a generic MultiMedia System (MMS). MMS is an integrated video/audio system which includes an *h263* video encoder, an *h263* video decoder, an *mp3* audio encoder and an *mp3* audio decoder. We partitioned the application into 40 distinct tasks and then these tasks were

assigned and scheduled onto 25 selected IPs [10]. These IPs range from DSPs, generic processors, embedded DRAMs to customized ASICs. Real video and audio clips are then used as inputs to derive the communication patterns among these IPs. Fig. 8 shows the derived CTG of this system based on the simulation result.

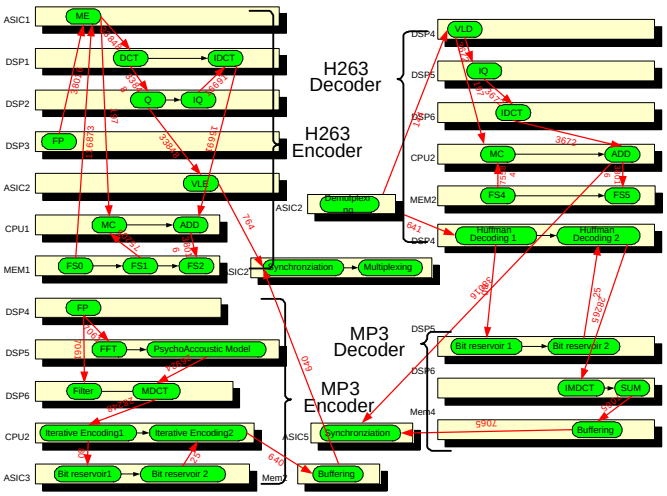

Fig. 8. Communication Task Graph for MMS

Applying our algorithm to MMS, the solution is found in less than 7 seconds CPU time. An ad-hoc implementation[9] was also developed to serve as reference. The results are shown in Table I.

TABLE I
POWER COMPARISON BETWEEN AD-HOC SOLUTION AND OUR SOLUTION

Movie clips	Ad-hoc(mW)	Our alg(mW)	Savings
box/hand	374.5	148.3	60.4%
akiyo/cup	440.5	187.3	57.5%
man/phone	327.5	120	63.4%

In Table I, each row represents the power consumption of using two movie clips as simulation inputs, with one clip for the video/audio encoder and the other for the video/audio decoder. Compared to the ad-hoc solution, we observe around *60.4%* energy savings, on average, which demonstrates the effectiveness of our algorithm.

Simulated annealing is also applied to MMS, and the power consumption of the solution and the run time are shown in Table II together with the results of our algorithm for comparison.

TABLE II
COMPARISON BETWEEN SA AND OUR ALGORITHM USING MMS

	SA	Our alg.	Ratio
Run Time (sec)	181.67	6.52	27.864
Power (mW)	148.8	148.3	0.997

As shown in Table II, for this complex application, our algorithm generates a better solution with significantly less run time compared to the simulated annealing. We should point out that although for this system which has only 5×5 tiles, the solving time of SA is affordable, the run time of SA increases dramatically as the system size scales up. For systems with 7×7 tiles, the average run time of SA increases to 2.2 hours, on average. For systems with 10×10 tiles (which will be available in the near future), our algorithm needs just a few minutes to complete while the run time of SA becomes prohibitive (in our experiments on systems with 10×10 tiles, the SA did not finish in 40 hours of CPU time).

[9]We randomly generated 3000 solutions, from which the one that consumes median energy consumption was chosen as the ad-hoc implementation.

VII. CONCLUSION AND FUTURE WORK

In this paper, we addressed the mapping problem for regular tile-based architectures. An efficient algorithm which automatically maps the IPs to the tiles so that the total communication energy consumption is minimized under specified performance constraints was proposed. By using simulated annealing as reference, we have shown that we can generate high quality solutions with significantly less computational time. Although in this paper we focus on the tile-based architecture interconnected by $2D$ mesh network with *XY* routing, our algorithm can be adapted to other *regular* architectures with different network topologies and different *static* routing schemes.

We plan to advance this research in several directions. Due to the short run time of our algorithm, one possible extension is to combine the IP selection and the task partitioning/scheduling into our framework such that the computation and communication energy consumption can be optimized at the same time.

ACKNOWLEDGMENTS

The authors would like to thank Dr. John Darringer and Dr. Youngsoo Shin of IBM T.J. Watson Research Center for insightful comments on this work.

REFERENCES

[1] P. Guerrier, A. Greiner, "A generic architecture for on-chip packet-switched interconnections," *Proc. DATE*, pp. 250–256, March 2000.

[2] W. J. Dally, B. Towles, "Route packets, not wires: on-chip interconnection networks," *Proc. DAC*, pp. 684–689, June 2001.

[3] J. Chang, M. Pedram, "Codex-dp: co-design of communicating systems using dynamic programming," *IEEE Tran. on CAD of Integrated Circuits and Systems*, vol. 19, No. 7, July 2002.

[4] A. Hemani, *et al*, "Network on a chip: an architecture for billion transistor era," *Proc. of the IEEE NorChip Conf.*, Nov. 2000.

[5] S. Kumar, *et al*, "A network on chip architecture and design methodology," *Proc. Symposium on VLSI*, pp. 117–124, April 2002.

[6] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," *Proc. ISCA*, May 1992.

[7] T. T. Ye, L. Benini, G. De Micheli, "Analysis of power consumption on switch fabrics in network routers," *Proc. DAC*, June 2002.

[8] R. P. Dick, D. L. Rhodes, W. Wolf, "TGFF: task graphs for free," *Proc. Intl. Workshop on Hardware/Software Codesign*, March 1998.

[9] M. R. Garey, D. S. Johnson, "Computers and intractability: a guide to the theory of NP-completeness," *Freeman*, 1979.

[10] http://www.mentor.com/inventra/cores/catalog/index.html