Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures *

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 onto a generic regular Network on Chip (NoC) architecture and constructs a deadlock-free deterministic routing function such that the total communication energy is minimized. At the same time, the performance of the resulting communication system is guaranteed to satisfy the specified constraints through bandwidth reservation. As the main contribution, we first formulate the problem of energy/performance aware mapping, in a topological sense, and show how the routing flexibility can be exploited to expand the solution space and improve the solution quality. An efficient branch-and-bound algorithm is then described to solve this problem. Experimental results show that the proposed algorithm is very fast, and significant energy savings can be achieved. For instance, for a complex video/audio application, 51.7% energy savings have been observed, on average, compared to an ad-hoc implementation.

1. Introduction

Regular tile-based NoC architecture was recently proposed to mitigate complex on-chip communication problems [1][4][5]. As shown in the left of Fig. 1, such a chip consists of 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 on-chip wires, the inter-tile communication can be achieved by routing packets.

Given a target application described as a set of concurrent tasks which have been assigned and scheduled, to exploit such an architecture, the fundamental questions to answer are: *i*) *which tile* each IP should be mapped to, *ii*) *what routing algorithm* is suitable for directing the information among tiles, such that the metrics of interest are optimized. More precisely, in order to get the best power/performance trade-off, the designer needs to determine the *topological placement* of these IPs onto different tiles. Referring to Fig. 1, this means to determine, for instance, onto which tile (e.g. (3,1), (1,3) *etc*.) each IP (e.g. DSP2, DSP3 *etc*.) should be placed. Since there may exist multiple *minimal* routing paths, one also needs to select *one* qualified path for each communicating tile pair. For example, one has to determine which path (e.g. (3,1)→(2,1)→(2,2)→(1,2)→(1,3) or (3,1)→ (3,2)→(3,3)→(2,3)→(1,3) *etc*.) should the packets follow to send data from DSP2 to DSP3, if these two IPs are meant to be placed to tile (3,1) and (1,3), respectively.

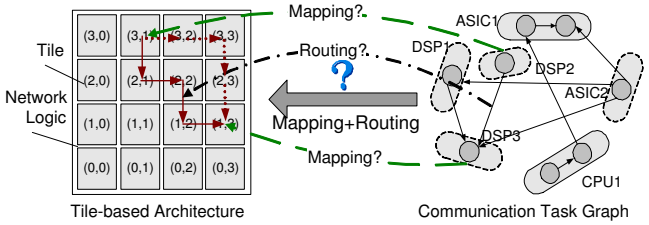

Figure 1. Tile-based arch and mapping/routing problems

While task assignment and scheduling problems have been addressed before [2], the *mapping* and *routing* problems described above represent a new challenge, especially in the context of the regular tile-based architecture, as this significantly impacts the energy and performance metrics of the system. In this paper, we address this very issue and propose an efficient algorithm to solve it. To this end, we first propose a suitable routing scheme (Sec. 3) and a new energy model (Sec. 4) for NoC. The problem of mapping and routing path allocation are then formulated in Sec. 5. Next, an efficient branch-and-bound algorithm is proposed to solve this problem under performance constraints (Sec. 6). Experimental results in Sec. 7 show that significant energy savings can be achieved, while guaranteeing the specified system performance. For instance, for a complex video/audio application, on average, 51.7% energy savings have been observed compared to an ad-hoc implementation.

2 Related Work

In [1][4][5], the on-chip interconnection networks are proposed to structure the top-level wires on a chip and fa-

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

cilitate modular design. While these papers discuss the overall advantages and challenges of regular NoC architecture, to the best of our knowledge, our work is the first to address the *mapping* and *routing path allocation* problems for tile-based architectures and provide an efficient way to solve them. Although routing (especially wormhole-based routing [3]) has been a hot research topic in the area of *direct* networks for parallel and distributed systems [6][7][8], the specifics of NoC force us to re-think standard network techniques and adapt them to the context of NoC architectures. In what follows, we address this issue by presenting a suitable routing technique for NoC together with an algorithm for automatic generation of the routing function.

3 The Proposed Routing Scheme for NoCs

The major differences between regular data-networks and NoCs are:

A. Buffering space: To minimize the implementation cost, it's more reasonable to use *registers* as buffers for on-chip routers instead of huge memories (SRAM or DRAM) as is the case of regular data-networks. This also helps in decreasing access latency, which is critical for typical SoC applications.

B. Use of deterministic routing: We believe that for NoC, deterministic routing is more suitable because:

• *Resource limitation and stringent latency requirements*

Compared to deterministic routers, implementing adaptive routers requires by far more resources. Moreover, since in adaptive routing the packets may arrive out of order, huge buffering space is needed to reorder them. This, together with its protocol overhead leads to prohibitive cost, extra delay and jitter.

• *Traffic predictability*

In contrast to typical regular data-networks, most NoCs need to support one application or, at most, a small class of applications. Consequently, the designer has a good understanding of the traffic characteristics and can use this information to avoid congestion by wisely mapping the IPs and allocating the routing paths. This further weakens the potential advantages of adaptive routing.

In summary, because of the aforementioned characteristics, together with consideration for deadlock problem, programmability, *etc*, we argue that the most appropriate routing technique for NoC should be *deterministic*, *deadlock-free*, *minimal*, *wormhole-based* [11].

4 Platform Description

In this section, we describe the regular tile-based architecture and the energy model for its communication network.

4.1 The Architecture

The system under consideration is composed of $n \times n$ tiles interconnected by a $2D$ mesh network[1]. Fig. 2 shows an abstract view of a tile in this architecture.

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

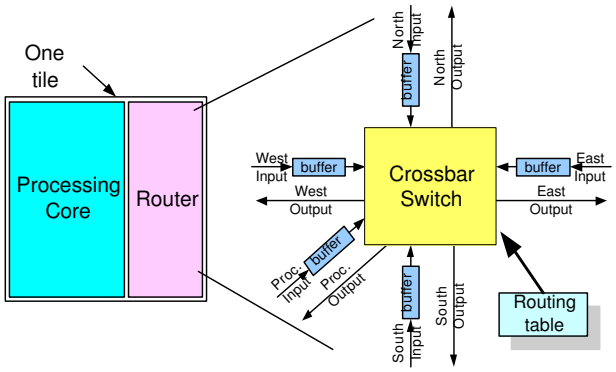

Figure 2. The typical structure of a tile

Each tile in Fig. 2 is composed of a *processing core* and a *router*. The router is connected to the four neighboring tiles and its local processing core via channels (each consisting of two *one-directional point-to-point* links).

Due to the limited resources, buffers are implemented using registers (typically in the size of one or two flits each). A 5×5 crossbar switch is used as the switching fabric in the router. Based on the source/destination address, the routing table decides which output link the packet should be delivered to.

4.2 The Energy Model

Ye *et al.* [9] proposed a model for power consumption of network routers. The *bit energy* (E_{bit}) metric is defined as the energy consumed when one bit of data is transported through the router:

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

where $E_{S_{bit}}$, $E_{B_{bit}}$ and $E_{W_{bit}}$ represent the energy consumed by the switch, buffering and interconnection wires, respectively. (The authors in [9] assume the buffers are implemented in SRAM or DRAM.) However, the above power model is targeted for network routers where the *entire* chip is occupied by just *one* router. For tile-based NoC architectures, we need a new energy model because:

• In Eq. (1), $E_{B_{bit}}$ becomes dominant during congestion, as accessing and refreshing the memory are power expensive. This is no longer true for NoC where buffers are implemented with registers.

• In Eq. (1), $E_{W_{bit}}$ is the energy consumed on the wires inside the switch fabric. For NoC, the energy consumed *on the links* between tiles ($E_{L_{bit}}$) should also be included. 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 length of a link is typically in the order of *mm*, the energy consumed by buffering ($E_{B_{bit}}$) and internal wires ($E_{W_{bit}}$) is negligible[2] compared to $E_{L_{bit}}$; Eq. (2) reduces to:

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

[2]We evaluated the energy consumption metrics using Spice for a $0.35\mu m$ technology. The results show that $E_{B_{bit}} = 0.073pJ$, which is indeed negligible compared to $E_{L_{bit}}$ (typically in the order of pJ).

Consequently, in our new model, the average energy consumption of sending one bit of data from tile t_i to tile t_j is:

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

where n_{hops} is the number of routers the bit passes.

For $2D$ mesh networks with *minimal* routing, Eq. (4) shows that the average energy consumption of sending one bit of data from t_i to t_j is determined by the *Manhattan* distance between them.

5 Problem Formulation

Simply stated, for a given application, we try to decide onto which tile should each IP be mapped and how should the packets be routed, such that the total communication energy consumption is minimized under performance constraints. To formulate this problem more formally, we define the following terms:

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, and each directed arc $a_{i,j}$ characterizes the communication from c_i to c_j. Each $a_{i,j}$ has the following 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 network in order to meet the performance constraints.

Definition 2 An *Architecture Characterization Graph* (ARCG) $\mathcal{G}' = G(T,R)$ is a *directed* graph, where each vertex t_i represents one tile in the architecture, and each directed arc $r_{i,j}$ represents the routing from t_i to t_j. Each $r_{i,j}$ has the following properties:

- $P_{i,j}$: a set of candidate *minimal* paths from tile t_i to tile t_j. $\forall p_{i,j} \in P_{i,j}$, $L(p_{i,j})$ gives the set of links used by $p_{i,j}$.
- $e(r_{i,j})$: arc cost. This represents the average energy consumption (*joule*) of sending one bit of data from t_i to t_j, *i. e.*, $E_{bit}^{t_i,t_j}$.

Definition 3 For an ARCG $\mathcal{G}' = G(T,R)$, a *routing function* $\mathcal{R}: R \rightarrow P$ maps $r_{i,j}$ to *one* routing path $p_{i,j}$, where $p_{i,j} \in P_{i,j}$.

Using these definitions, the energy/performance aware mapping and routing path allocation problem 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 and a *deadlock-free, minimal* routing function $\mathcal{R}()$ which:

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

such that:

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

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

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

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

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

In this formulation, 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 communication traffic (load) of any link will not exceed its bandwidth.

We show in detail in [11] that the impact of mapping on the energy consumption is significant. In order to generate the best solution, routing paths have to be wisely allocated. In the following, we propose an efficient algorithm which can find nearly optimal solutions in reasonable run times.

6 Energy/Performance Aware Mapping and Routing Path Allocation

6.1 The Data Structure

Our approach is based on a *branch-and-bound* algorithm which efficiently walks through the *searching tree* that represents the solution space. Fig. 3 shows a searching tree example for mapping a 4-IP application onto a 2×2-tile architecture.

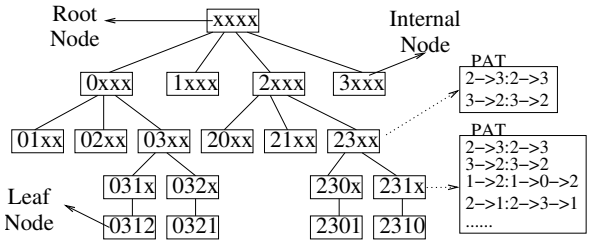

Figure 3. An example search tree

Each node in the tree is either a *root* node, an *internal* node, or a *leaf* node. The root node corresponds to the state where *no* IP has been mapped and *no* routing path has been allocated. Each internal node represents a *partial* mapping. For example, the node labeled "23*xx*" represents a partial mapping where IP_0 and IP_1 are mapped to tile t_2 and tile t_3 respectively, while IP_2 and IP_3 are still unmapped. Each leaf node represents a *complete* mapping of the IPs to the tiles. Each node also has a *path allocation table* (PAT) which stores the routing paths for the traffic among its *occupied* tiles. For instance, the PAT of node "231*x*" in Fig. 3 shows that the traffic from t_1 to t_2 takes the path $t_1 \rightarrow t_0 \rightarrow t_2$, *etc.* The PAT of any node is automatically generated by the algorithm. When a child node is generated, the PAT of its parent node is inherited. Next, the routing paths for the traffic involving the newly occupied tile are allocated and added to its PAT. Caution must be taken to ensure freedom of deadlock.

To explain the algorithm, we define the following terms:

Definition 4 The *cost* of a node is the energy consumed by the communication among those IPs that have already been mapped.

Definition 5 Let $\mathcal{M}$ be the set of vertices in the APCG that have already have been mapped. A node is called a *legal* node if and only if it satisfies the following tow conditions:

- The routing paths specified by the PAT are deadlock free.
- $\forall l_k, B(l_k) \geq \sum_{\forall a_{i,j}, c_i c_j \in \mathcal{M}} b(a_{i,j}) \times f(l_k, p_{map(c_i),map(c_j)})$ where $p_{map(c_i),map(c_j)}$ is the routing path from tile $map(c_i)$ to $map(c_j)$ specified by the PAT.

6.2 The Branch-and-Bound Algorithm

Given the above definitions, finding the optimal solution of Eq. (6) is equivalent to finding the *legal leaf* node which has the *minimal* cost. To achieve this, our algorithm searches the optimal solution by alternating the following two steps:

Branch: An unexpanded node is selected and its next *unmapped* IP is enumeratively assigned to the remaining *unoccupied* tiles to generate the corresponding new child nodes. The PAT of each child node is also generated by first copying its parent node's PAT and then allocating the routing paths for the traffics between the newly occupied tile and the other occupied tiles. The routing paths specified by the PAT have to be deadlock-free.

Bound: Each of the newly generated child nodes is 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 Lower Bound Cost (LBC) is higher than the *lowest* Upper Bound Cost (UBC) that has been found[3].

How the algorithm allocates the routing paths is critical to its performance. Better routing path allocation helps balancing the traffic which leads to a better solution, but needs more time to compute. Next, we describe our routing path allocation heuristic which can find a good routing path within reasonably short computational times.

Routing path allocation has the objective to find *deadlock-free*, *minimal* routing paths, and at the same time, to balance the traffic across the links.

To be deadlock free, the routing algorithm needs to prohibit *at least one turn* in each of the possible routing cycles. In addition, it should *not* prohibit more turns than necessary to preserve the adaptiveness. Based on this, several deadlock-free adaptive routing algorithms have been proposed [7], including *west-first*, *north-last* and *negative-first*. In [8], Chiu proposed the *odd-even* turn model which restricts the locations where some types of turns can take place such that the algorithm remains deadlock-free.

In our algorithm, we convert the adaptiveness offered by the above algorithms into the flexibility for the routing path allocation by constructing a *Legal Turn Set* (LTS). A LTS is composed of and *only* of those turns allowed in the corresponding algorithm. Any path to be allocated can only employ turns from the LTS. The advantage of using LTS is two folds: First, since the turns are restricted to LTS, deadlock-free routing is guaranteed. Second, since only minimum turns are prohibited, the routing path allocation is still highly flexible.

[3]The computation of UBC/LBC is important for the performance of our algorithm. Please refer to [11] for a detailed description of the method that we use to compute UBC and LBC.

Theoretically, the turns allowed in *any* of such deadlock-free adaptive routing algorithms can be used to construct a LTS. In this paper, we choose *west-first* and *odd-even* routing algorithm to build our LTS's[4]. Both of these two algorithms prohibit only $\frac{1}{4}$ of the total possible turns. However, the degree of the adaptiveness provided by *odd-even* routing is distributed more evenly than that provided by *west-first*.

Given an LTS, the following heuristic is used to allocate routing paths for a List of Communication Loads (LCL):

```
Sort LCL by the flexibility of each CL
for each CL in LCL {
   cur_tile = CL.source; dst_tile=CL.destination
   while(cur_tile ≠ dst_tile) {
     link = choose_link(cur_tile, dst_tile);
     cur_tile = link.next_tile();
     link.add_load(CL.bandwidth); }}
```

Figure 4. Routing Path Allocation

In Fig. 4, the Communication Load (CL) is first sorted by flexibility. The flexibility of a CL can be 1 (if there exists more than one *legal* path[5]) or 0 (otherwise). CLs with lower flexibility are given higher priorities for path allocation. If two CLs are tied in flexibility, the one with higher bandwidth requirement is given priority. Function *choose_link* returns the *least* loaded link allowed by LTS.

6.3 The Pseudo Code

```
Sort the IPs by communication demand
root_node = new node(NULL)
MUBC = +∞, 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
     allocate routing paths
     if(n_new's mirror node exists in the PQ)
       continue
     if(n_new.LBC>MUBC) 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<MUBC) MUBC = n_new.UBC
       PQ.insert(n_new) }}
}
```

Figure 5. The pseudo code of the algorithm

Fig. 5 gives the pseudo code of our algorithm. Two speedup techniques are employed to trim away more non-promising nodes, early in the search process:

• *IP ordering*: IPs are sorted by their communication demands ($\sum_{\forall j \neq i}\{v(a_{i,j}) + v(a_{j,i})\}$ for IP c_i) so that the IPs with higher demand will be mapped earlier. Since the positions

[4]*north-last* and *negative-first* routing are similar to *west-first* routing.

[5]A *legal* path has to be *minimal* and employ only those turns in the LTS.

of the IPs with higher demands have larger impact on the overall communication energy consumption, fixing their positions earlier helps exposing those non-promising nodes earlier in the searching process; this reduces the number of nodes to be expanded.

• *Priority queue (PQ)*: The PQ sorts the nodes to be branched based on their cost. The lower the cost of the node, the higher the priority it has for branching, as expanding a node with lower cost will more likely decrease the minimum UBC.

Obviously, as the system size scales up, the run time of the algorithm will also increase. Fortunately, we can trade-off the solution quality with run time by limiting the maximum length of PQ. When PQ's length reaches a threshold value, strict criteria are applied to select the child nodes for insertion into PQ.

7 Experimental Results

7.1 Evaluation on Random Applications

We first compare the run-time and the solution quality of our algorithm against a *simulated annealing* optimizer (SA). To make the comparison fair, SA was optimized by carefully selecting parameters such as number of moves per temperature, cooling schedule, *etc*. Since SA restricts itself to *XY* routing, for the first set of experiments, we also configure our algorithm to run without exploiting the routing flexibility by fixing it to just *XY* routing. In the following, we call this version EPAM-XY. (EPAM stands for Energy/Performance Aware Mapping.)

Four categories (I, II, III, IV) of random benchmarks were generated, each containing 10 applications with 9, 16, 25 and 36 IPs, respectively. EPAM-XY and SA were then applied to map these applications onto architectures with the same number of tiles. The results are shown in Fig. 6.

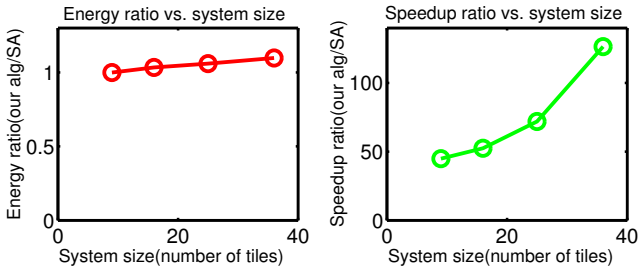

Figure 6. Comparison between SA and EPAM-XY

As shown in Fig. 6, for applications with 9 tiles, EPAM-XY runs **45** times faster than SA, on average. As the system size scales up, the speedup also increases dramatically. For applications with 36 tiles, the average speedup of EPAM-XY over SA increases to **127**. Meanwhile, the solutions produced by our algorithm remain very competitive. Indeed, the energy consumption of the solutions generated by EPAM-XY is only 3%, 6% and 10% larger than the SA solutions for category II, III and IV, respectively. For category I, EPAM-XY even finds better solutions because it can walk through the whole search tree due to the small problem size.

To evaluate the benefits of exploiting the routing flexibility, we also applied our algorithm on benchmark applications using different LTS configurations. The comparison between the algorithms with LTS's based on *XY* routing (EPAM-XY), *Odd-Even* routing (EPAM-OE) and *West-First* (EPAM-WF) is shown in Fig. 7.

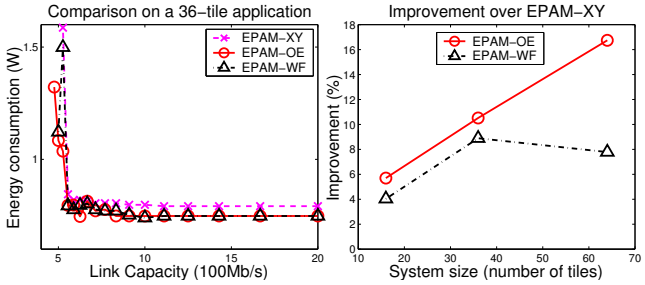

Figure 7. The effectiveness of routing flexibility exploitation

The left part of Fig. 7 shows how EPAM-XY, EPAM-OE and EPAM-WF perform for a typical 36-tile application as the link bandwidth constraints change. The results demonstrate two advantages of routing flexibility exploitation:

• First, it helps to find solutions for architectures with lower link bandwidth (which implies a lower implementation cost). For example, for this application, without exploiting the routing flexibility, the link bandwidth has to be $526Mb/s$ (or higher) in order to find a solution which meets the performance constraints. By exploiting the routing flexibility, the link bandwidth requirement decreases to $476Mb/s$ (EPAM-OE) and $500Mb/s$ (EPAM-WF), respectively.

• Second, given the same architecture, exploiting routing flexibility leads to solutions with less energy consumption. The benefit becomes more significant as the links' loads reach their capacity. For instance, using the architecture where each link can provide $526Mb/s$ bandwidth, the power consumption of the solution generated by EPAM-XY is $1.60W$. The power consumptions of the solutions generated by EPAM-OE and EPAM-WF are only $1.50W$ and $1.04W$ respectively. So significant power savings are achieved.

The right part of Fig. 7 shows how the effectiveness of routing flexibility exploitation changes as the problem size scales. Let BW^{min} be the minimum link bandwidth needed by the corresponding algorithm to be able to find a solution which meets the performance constraints. The improvement (Y-axis) is defined as $\frac{BW^{min}_{XY} - BW^{min}}{BW^{min}}$, which is a measure of how much the link bandwidth can be relaxed by exploiting routing flexibility compared to that using the EPAM-XY. As we can see, both EPAM-OE and EPAM-WF perform better than EPAM-XY. For applications with 16 tiles, EPAM-OE provides 5.69% improvement over EPAM-XY, on average. As the problem size scales up to 36 tiles and 64 tiles, the average improvement increases to 10.51% and 16.74%, respectively.

Overall, EPAM-OE performs *much better* than EPAM-WF, due to the fact that the *odd-even* routing provides more even adaptiveness than the *west-first* routing [8].

7.2 A Video/Audio Application

To evaluate the potential of our algorithm for real applications, we applied it to a generic MultiMedia System (MMS) [11]. 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 first partition MMS into 40 concurrent tasks and then assign/schedule these tasks onto 16 selected IPs available from industry [12]. These IPs range from DSP, generic processor, embedded DRAM to customized ASIC. We then use real video and audio clips as inputs to derive the communication patterns among these IPs.

Applying EPAM-OE to MMS, the solution is found in less than 0.5 sec. CPU time. An ad-hoc implementation was also developed to serve as reference. The results are shown in Table I.

Table 1. Comparison of ad-hoc vs. EPAM-OE

Movie clips	Ad-hoc(mW)	EPAM-OE(mW)	Savings
box/hand	171.2	82.8	51.6%
akiyo/cup	226.2	104.8	53.7%
man/phone	133.3	66.88	49.8%

In Table 1, 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 **51.7%** energy savings, on average, which demonstrates the effectiveness of our algorithm.

To compare the performance between EPAM-OE and SA, both of them are applied to MMS for an architecture whose link bandwidth is fixed to $333Mb/s$. The results are shown in Table 2.

Table 2. Comparison between SA and EPAM-OE

	SA	EPAM-OE	Improvement
Run Time (sec)	25.55	0.31	82.419
Power (mW)	119.36	105.12	11.9%

As shown in Table 2, our algorithm generates a better solution (about 12% less power) with significantly shorter run time compared to SA. We should point out that while the time of SA is affordable for this system (because it has only 4×4 tiles), the run time of SA increases dramatically as the system size scales up. For instance, for systems with 7×7 tiles, the average run time of SA increases to 2.2 hours. For systems with 10×10 tiles (which may be available in the near future [5]), our algorithm needs just a few minutes to complete while the run time of SA becomes prohibitive (in our experiments, SA did not finish in 40 hours of CPU time).

To show how much improvement our algorithm can achieve by exploiting routing flexibility, we applied EPAM-XY, EPAM-OE and EPAM-WF to MMS (Fig. 8). As shown in Fig. 8, EPAM-XY fails to find a solution when the link bandwidth decreases to $324Mb/s$ (point A in Fig. 8). In contrast, both EPAM-OE and EPAM-WF can still find solution even the link bandwidth decreases to $307Mb/s$ (point B in Fig. 8), thus suggesting a 5.5% improvement. Given the same architecture whose link bandwidth is fixed to $324Mb/s$, the power consumptions of the solutions found by EPAM-OE and EPAM-WF are both $109.2mW$, while the power consumption of the EPAM-XY solution is $150.9mW$; thus, **27.6%** of power savings are achieved. Again, this shows the advantage of exploiting the routing flexibility.

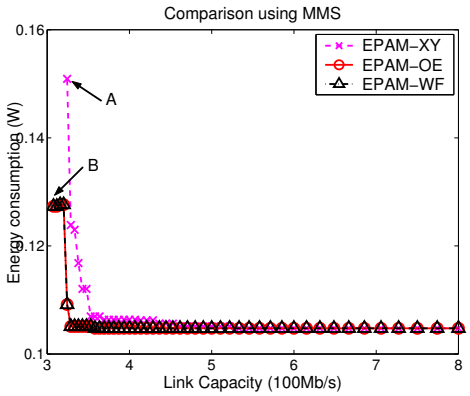

Figure 8. Comparison Using MMS Application

8 Conclusion and Future Work

In this paper, we proposed an efficient algorithm for solving the mapping and routing path allocation problems in regular tile-based NoC architectures. Although we focus on the architectures interconnected by $2D$ mesh networks, our algorithm can be adapted to other *regular* architectures with different network topologies. This remains to be done as future work.

References

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

[2] 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.

[3] W. J. Dally, C. L. Seitz, "The torus routing chip," *Journal of Distributed Computing*, vol. 1, no. 3, pp. 187–196, 1986.

[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] L. M. Ni, P. K. McKinley "A survey of wormhole routing techniques in direct networks," *Computer*, vol. 26, no. 2, Feb. 1993.

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

[8] G. Chiu, "The odd-even turn model for adaptive routing," *IEEE Tran. on Parallel and Distributed Systems*, vol. 11, no. 7, pp. 729–738, July 2000.

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

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

[11] J. Hu, R. Marculescu, "Exploiting the routing flexibility for energy/performance aware mapping of regular NoC architectures," *CSSI Technical Report*, Carnegie Mellon University, Sept. 2002. Available at http://www.ece.cmu.edu/∼sld/publications.html

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