Incremental Run-time Application Mapping for Homogeneous NoCs with Multiple Voltage Levels

Chen-Ling Chou, Radu Marculescu
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213-3890, USA
{chenlinc,radum}@andrew.cmu.edu

ABSTRACT

In this paper, we propose an efficient technique for run-time application mapping onto Network-on-Chip (NoC) platforms with multiple voltage levels. Our technique consists of a region selection algorithm and a heuristic for run-time application mapping which minimizes the communication energy consumption, while still providing the required performance guarantees. The proposed technique allows for new applications to be easily added to the system platform with minimal inter-processor communication overhead. Moreover, our approach scales very well for large designs. Finally, the experimental results show as much as 50% communication energy savings compared to arbitrary mapping solutions.

Categories and Subject Descriptors

J.6 [**Computer Applications**]: Computer-Aided Design – *computer-aided design (CAD)*.

General Terms

Algorithms, Performance, Design

Keywords

Dynamic application mapping, Networks-on-Chip, Low-power

1. INTRODUCTION

Due to increasing systems complexity, the design of future Systems-on-Chip (SoCs) faces major challenges. One such challenge is the design of the communication infrastructure. Indeed, the traditional bus-based or more complex hierarchical bus structures (*e.g.* AMBA, STBus) cannot satisfy the performance, scalability, flexibility, and energy efficiency requirements of future systems. Instead, the Network-on-Chip (NoC) architectures, consisting of various processor and storage elements interconnected via a packet switched network, emerged as a promising design paradigm [1].

With SoC design moving towards a communication-centric paradigm, communication energy minimization becomes a major concern. This is the motivation behind our approach to develop a run-time application mapping algorithm for minimizing the inter-processor communication. The target NoC architecture consists of several homogenous processing elements (*PE*s) that reside on the same chip. The *PE*s can operate at multiple voltage levels, under different energy-performance trade-offs. A global manager (GM) handles the real-time deadlines and inter-processor communication. When a new application arrives, the incoming tasks need to be allocated to the appropriate resources of the current configuration. The GM makes a run-time decision by executing our proposed algorithm in order to ensure communication energy savings. At the same time, the pre-existing applications are still run on the initial resources they have been allocated to. Recent work shows that task migration mechanisms with check points can also provide energy savings [18]. Our run-time mapping approach can be also used in conjunction with task migration to provide further energy savings.

CODES+ISSS'07, September 30–October 5, 2007, Salzburg, Austria.

1.1 Motivational example

A system architecture with two voltage levels is shown in Figure 1. The gray squares represent the *PE*s operating at higher voltage levels, while the black dots show the tasks of a pre-existing application, already running on system resources, which cannot be reallocated. Assume that applications App 1 and App 2 shown in Figure 1(a) and Figure 1(b), respectively, need to be mapped on the initial system configuration depicted in Figure 1(c). Each edge in App 1 represents communication between two nodes, where the weights of the edges give the corresponding communication volume. In this simple example, we assume all edges carry the same weight equal to 1 unit and *XY* routing is used to move data around. For instance, the communication cost for nodes 5 and 6 in Figure 1(f) is 2. Suppose that nodes 4 and 6 are the critical nodes for App 1 and node 2 is the critical node for App 2; this means that they *must* be allocated to the PEs with highest voltage level in order to meet the application deadlines. For this example, the mapping can go in two different ways: Either using a greedy approach (*i.e.,* minimizing the inter-processor communication cost at each time step without considering any incoming application), or using our proposed solution which *does* consider applications that may come in the near future. As shown in Figure 1(d), even though the first approach minimizes the communication cost for the current configuration, the newly generated region formed by the remaining (available) PEs is quite dispersed. Hence, mapping additional applications (*e.g.,* App 2) on this configuration becomes less effective as seen in Figure 1(e). Contrary to this, our proposed solution offers a more effective mapping in the presence of dynamically incoming applications. As seen in Figure 1(g), when App 2 is incrementally mapped after App 1, the total inter-processor communication cost becomes smaller than the cost obtained using the greedy approach. Intuitively, since the pre-existing applications cannot be reallocated, the performance of the greedy solution becomes worse compared to the solution derived from our proposed approach.

In general, there is no true optimal solution for incremental mapping, since it is not possible to know *in advance* the exact sequence in which different applications may come into the system. The key point in our approach for selecting a near

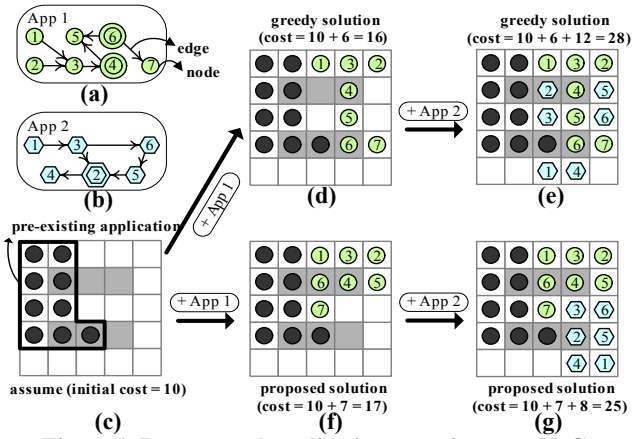

Figure 1. Incremental application mapping onto NoCs.

convex region is to minimize the non-contiguous regions which may cause a higher communication cost when mapping additional applications. Near convex region stands for a region whose area is *close* to the area of its convex hull [15] and the average pairwise distance is as small as possible [17].

1.2 Novel contribution

The novel contribution of this paper consists of an efficient run-time mapping algorithm that allocates the incoming applications based on a pre-existing system configuration. This mapping algorithm achieves several goals:

- Meets application deadlines
- Minimizes inter-processor communication energy consumption
- Incoming applications can easily be added to the resulting system with minimal inter-processor communication overhead.

Furthermore, with shorter communication distances among nodes, the network contention can be also mitigated and the dynamic link scheduling or network flow control improved.

This paper is organized as follows: Section 2 reviews related work. Platform description and the run-time mapping methodology are given in Section 3. Section 4 presents the problem formulation and a practical solution for incremental run-time mapping. Finally, the experimental results appear in Section 5, while Section 6 summarizes our main contribution.

2. RELATED WORK

The authors in [2] propose a branch and bound algorithm to map IP cores onto a tile-based NoC architecture, while satisfying the bandwidth constraints and minimizing the total communication energy consumption. The work in [3] considers the mapping problem for minimizing the communication delay with split routing. These off-line algorithms are *not* applicable to the run-time mapping problem which needs to complete within microseconds. Broersma *et al.* [5] propose the MinWeight algorithm for solving the minimum weight processor assignment problem; this can be done in polynomial time, but only for task graphs with maximum degree at most 2. Smit *et al.* in [4] extend the algorithm in [5] by solving the problem of run-time task assignment on heterogeneous processors with task graphs restricted to a small number of vertices or a large number of vertices with degree no more than two. The authors of [14] present a mapping and scheduling strategy for hard real-time embedded systems which communicate over a *shared medium* (*i.e.* bus) aiming at minimizing the system modification cost. Operating system (OS) issues to make NoCs more efficient in a dynamic manner are addressed in [13].

3. PRELIMINARIES

3.1 NoC architecture

Our NoC architecture consists of identical processing elements (*PE*s) interconnected by a 2-D $n \times n$ mesh network with bi-directional links. Each *PE* consists of a processing unit, a cache, local memory, and control unit, as shown in Figure 2. The *PEs* operate at a fixed voltage and frequency levels which are selected from a finite set (V_i, f_i). When the voltage level of a *PE* is different from that of the network, mixed-clock FIFOs are utilized. Since the focus of this paper is *not* on determining the voltage island partitioning, we assume that this is already determined using approaches similar to the voltage/frequency island partitioning and voltage assignment algorithms in [16].

Data network is responsible for delivering data packets among *PE*s. The *control* network (*i.e.,* the routers and links represented by smaller squares and dotted lines in Figure 2) is used to move around the control messages generated by the OS. Data and control networks are separated to ensure that data transmission in the data network does *not* interfere with the OS-generated control messages in the control network.

Finally, a global manager (GM) is included in this architecture. GM runs the proposed run-time mapping algorithm in order to allocate the incoming tasks to the appropriate resources. We note that a single central GM may not scale well, so for larger networks an hierarchical control mechanism should be applied.

3.2 Overview of the methodology

The proposed methodology is shown in Figure 3. The incoming applications are described by the application communication graph (ACG) which is generated using an off-line technique like in [11][12]. Each ACG = $G(V, E)$ (Figure 3) is a directed graph and contains:

- *Nodes*: Each node $v_k \in V$ represents a set of tasks obtained from the off-line task partitioning process. The tasks belonging to the same node are allocated onto the same idle PE. Each task has a given execution time. In case of data-dependent tasks, the worst case execution time is used.
- *Edges*: Each edge $e_{i,j} \in E$ represents the inter-node communication from v_i to v_j; node v_i is any neighbor of node v_j. Weights $w(e_{i,j})$ characterize the communication volume (bits) between nodes i and j, while weights $bw(e_{i,j})$ represent the minimum bandwidth requirements (bits per sec) from v_i to v_j;
- $S(v_k)$: Each node has its *minimum voltage* at which should operate in order to meet the application deadlines. Since the maximum operating frequency for a given voltage level is selected, higher voltage levels result in faster execution assuming that the task execution times (in clock cycles) remain the same. For instance, the system in Figure 4(b) has only two different voltage levels ('*H*' and '*L*'). The critical nodes in the

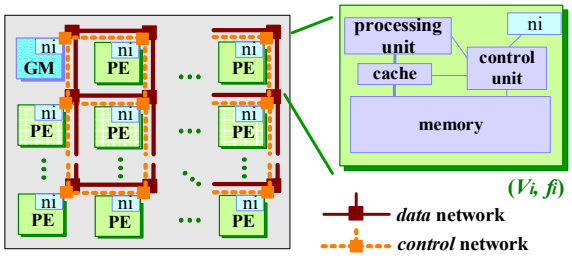

Figure 2. Homogeneous processing elements interconnected via a packet-switched network and an OS-controlled mesh.

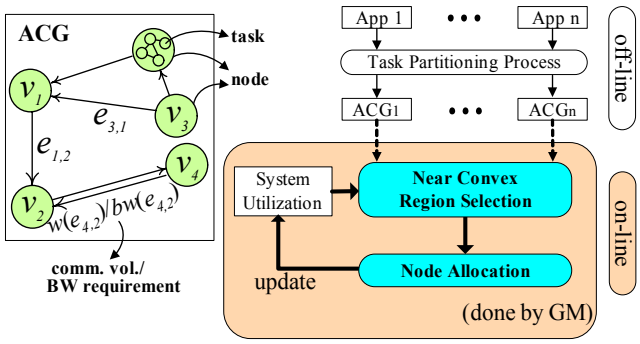

Figure 3. Overview of the proposed methodology.

ACG shown in Figure 4(a) (marked with "V_{min}=*H*") must be mapped to the *PE*s belonging to the *'H'* voltage level (*i.e.* the gray squares in Figure 4(b)).

During normal operation, the behavior of *PE*s (either active or idle) is monitored and conveyed to the GM through the control network. The GM recognizes the location of all pre-existing applications and performs the run-time task mapping for all subsequent applications, while minimizing the inter-processor communication and meeting the application deadlines. The run-time mapping process is activated *only* when new applications arrive in the system. As shown in Figure 3, our run-time mapping process involves two essential steps:

1. Near convex region selection: This consists of selecting a near convex region (contiguous, if possible) containing a number of *PE*s equal to the number of nodes in the ACG of the incoming application. If there are more than one feasible near convex regions, then the one closest to the off-chip memory containing the application code is selected (to minimize the code transfer energy consumption). The motivation for the near convex region selection is to allow for a more effective mapping for any subsequent applications. *PE*s belonging to different voltage levels are provided to help meeting the application deadlines. As shown in Figure 4(c), both regions shown with solid lines (*i.e.,* R1 and R2) satisfy the near convex region criterion with at least two *PE*s at the high voltage level; we assume R1 is selected in this example. The region shown with dotted line (R3) is too dispersed; if selected, it will incur much higher communication costs.

2. Node allocation: This step consists of assigning nodes to *PE*s within the selected region (with critical nodes mapped onto *PE*s with higher voltage levels), while minimizing the inter-node communication (see Figure 4(d)).

After the node allocation process is completed, the GM updates the system behavior and gets ready for continuing the mapping process. Last but not least, our run-time incremental mapping technique is *not* limited to mesh topologies; these two steps can be extended to any arbitrary topology with deterministic routing.

3.3 Energy model

We model the energy consumption of the network using the bit energy metric proposed in [7]. The network employs wormhole switching and *XY* routing. Suppose that the source (PE_s) and destination (PE_d) operate at static voltage/frequency levels (V_i, f_i), and (V_j, f_j), respectively, while the entire network uses a (constant) voltage/frequency level (V_n, f_n). The average energy consumption for sending one bit of data from PE_s to PE_d is:

$$E_{bit}(s,d) = \underbrace{E_{R_{bit}(V_i \rightarrow V_n)}}_{(a)} + \underbrace{E_{Network}(s,d)}_{(b)} + \underbrace{E_{R_{bit}(V_n \rightarrow V_j)}}_{(c)} \quad (1)$$

where the $E_{R_{bit}(V_i \rightarrow V_n)}$ and $E_{R_{bit}(V_n \rightarrow V_j)}$ terms (denoted by (a) and (c)) represent the energy consumed while transferring data between the network and the processors operating at different voltage levels. For instance, when PE_s operates at V_i and the network operates at V_n, the energy consumption of sending one bit of data from the processor to the network is $E_{R_{bit}(V_i \rightarrow V_n)}$. On the other hand, the $E_{Network}(s,d)$ (term (b)) is defined as:

$$E_{Network}(s,d) = E_{R_{bit}(V_n \rightarrow V_n)} \times (hops(s,d)-2) + E_{Link}(s,d) \times (hops(s,d)) - 1 \quad (2)$$

where *hops*(*s*,*d*) is the number of hops between nodes *s* and *d*. The routers attached to nodes *s* and *d* are not included in this equation since they are already accounted in Equation 1. The total communication energy consumption is defined as:

$$E_{comm} = \sum_{\forall e_{i,j} = (s_i, d_i) \in E} w(e_{i,j}) \times E_{bit}(s_i, d_j) \quad (3)$$

4. INCREMENTAL RUN-TIME MAPPING

4.1 Problem formulation

Given the ACG of an incoming application, our objective is to first select a near convex region and then decide on which PE within this region should each node in ACG be mapped to, such that the communication energy consumption is minimized under given timing constraints. To formulate this problem, we need a few notations as follows:

- PE_{ij}: the *PE* located at the intersection of the i^{th} row and j^{th} column of the network. PE_{11} is the global manager (GM);
- $V(PE_{ij})$: the voltage level that processor PE_{ij} belongs to;
- $MD(PE_{ij}, PE_{i'j'})$: Manhattan distance between PE_{ij} and $PE_{i'j'}$.

Using these notations, the problem of incremental mapping for NoCs can be formulated as follows:

Given the current system behavior and the ACG of the incoming application

Find a near convex region *R* and a node mapping function *map*( ) that satisfies:

Objective $\forall v_k \in V, map(v_k) \rightarrow PE_{ij}$ in *R*

$$min\left\{Energy = \sum_{\forall e_{i,j}} w(e_{i,j}) \times MD(map(v_i), map(v_j))\right\} \quad (4)$$

such that:

$$\forall v_k \in V, V(PE_{ij}) \geq S(v_k) \quad (5)$$

for all $e_{i,j}$ satisfying the bandwidth constraints at each link

4.2 Solution to region selection problem

We need two new terms to define the region selection problem:

Dispersion factor (*D*): The *PE* dispersion factor, *D*(*PE*), is defined as the number of idling neighbors of that *PE*. Smaller *D*(*PE*) values indicate a high likelihood for including the *PE* into the current region. Indeed, all *PE*s except the boundary ones have four neighbors. A *PE* that has most of its neighbors utilized (*i.e.*, it has a small *D*(*PE*) value), it is very likely to be later isolated; then selecting this *PE* for the current region helps reducing its dispersion probability.

Centrifugal factor (*C*): The *PE* centrifugal factor, *C*(*PE*), is defined as the Manhattan distance between any *PE* and the border of the current region. Smaller *C*(*PE*) values indicate a high likelihood for including the *PE* into the current region. Indeed, since every *PE* in a near convex region should be close

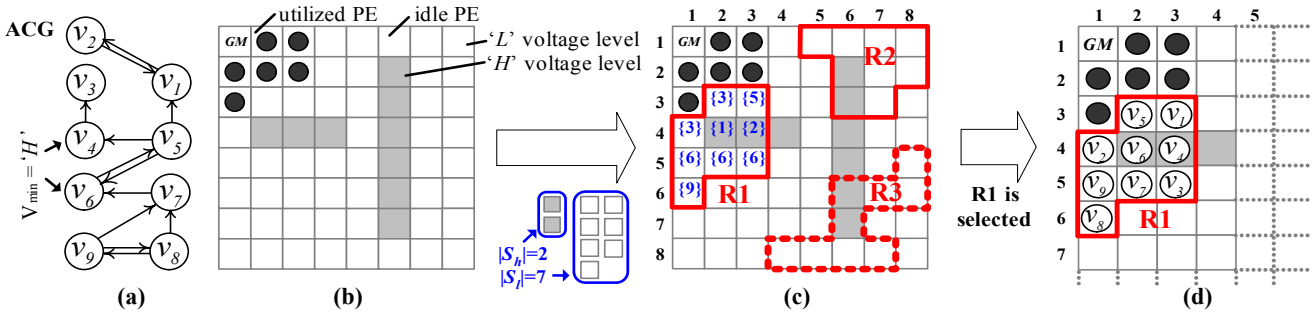

Figure 4. Incremental run-time mapping process. (a) ACG of the incoming application (b) System behavior (c) Near convex region selection process (d) Node allocation process.

to any border of that region, the *PE* with the smaller $C(PE)$ is better suited for selection to form a near convex region.

The steps of region selection similar to maze routing [19] are shown in Figure 5 (assuming k voltage levels in the system).

Step 1): For each node, assign it to a voltage level which satisfies the deadline constraint at the minimum voltage. Then, get the size of ***PE*** *sets for each voltage level,* $|\boldsymbol{S_c}|$*, and sort them out in non-decreasing order,* $S_1 \le S_2 \le S_3 \le \ldots \le S_k$.

Step 2): Start with $\boldsymbol{S_1}$*, select a* $\boldsymbol{PE_{ij}}$ *with minimum code transfer energy consumption and include it into the region.*

Step 3): Update the $\boldsymbol{D(PE)}$ *and* $\boldsymbol{C(PE)}$ *for unselected and idle PEs of that set. Select* $\boldsymbol{PE_{ij}}$ *with lowest* $\boldsymbol{D(PE_{ij})+C(PE_{ij})}$ *in the region. Continue with Step 3 until the number of PEs in the selected region matches the size of this set.*

Step 4): Repeat Step 3 for the remaining sets.

Figure 5. Near convex region selection.

We consider now a simple example and describe our approach step by step. The ACG of the incoming application and system behavior are given in Figure 4 ((a) and (b)), where the black dots are showing the pre-existing applications in the system. The number marked on the *PE* (*e.g.*, {3} on PE_{32}) in Figure 4(c) represents the *selection order* in getting a near convex region. *PE*s with the same number show that they are selected into the region at the same time.

From Figure 4(a), $|S_h| = 2$ and $|S_l| = 7$ (*Step 1*). Start with S_h (*Step 2*), and assume PE_{42} is selected first into the region (Figure 4(c)). Then, PE_{43} is the second node selected for the region (*Step 3*) since it has the lowest $D(PE_{43}) + C(PE_{43}) = 3+1$ ($D(PE_{43}) = 3$ because PE_{33}, PE_{44}, and PE_{53} are idle). *Step 3* is terminated since the *PE*s in S_h are all selected into the region. Now, we deal with the selection of S_l (*Step 4*). Going back to *Step 3*, PE_{32} and PE_{41} are selected since they all have the lowest $D(PE) + C(PE) = 1+1$. With the same rule, PE_{33}, PE_{51}, PE_{52}, PE_{53} are subsequently selected. Finally, PE_{61} is randomly selected among PE_{61}, PE_{62}, PE_{63}, and PE_{54}, with all of them getting the same value of $D(PE) + C(PE)$.

The time complexity for region selection algorithm is $O(VlogV)$. In *Step 1*, calculating the size of *PE* sets takes $O(V)$, while the runtime for sorting them is $O(VlogV)$ if using QUICKSORT. For implementing *Steps 3* and *4* with HEAP, we can record the frontier of the region and store the information, $D(PE) + C(PE)$, of this wavefront with runtime $O(VlogV)$.

4.3 Solution to node allocation problem

After the near convex region is selected, we continue with allocating nodes of the incoming application to the *PE*s with specific voltage levels in the selected region, while minimizing the inter-processor communication. To keep track of node allocation process, our heuristic colors each node white, gray, or black. All nodes start out white and may later either become gray and then black, or become directly black. A gray node indicates that it has some tentative *PE* locations but its precise location will be decided later. On the contrary, a black node indicates that it has been already mapped onto some *PE* and this mapping will not change anymore. A *PE* is set to be unavailable after a black node is mapped onto it.

We define two actions for nodes during the heuristic:

DISCOVER: This consists of: 1) Select available *PE*s with a specific voltage level for node t and 2) Color node t gray; then, node t is considered as "discovered".

FINISH: This consists of: 1) Select a specific *PE* for node t such that the distance between node t and its gray or black neighboring nodes is minimized and bandwidth requirements for additional links are satisfied (Note that if more than one *PE* gets the minimum distance, we select the PE_{ij} with its $D(PE_{ij})$ closest to the number of nonblack neighbors of node t) and 2) Color node t black; then, node t is considered as "finished".

In short, the nodes are colored during the process to indicate their state. Each node is initially white, it is grayed when it is discovered and is blackened when it is finished. We first sort nodes into an ordered set using the non-increasing order of their total communication volume; that is, the higher communication volume a node has, the earlier it is discovered or finished. The node allocation heuristic is summarized in Figure 6.

Step 1): Color all nodes white. Then, start with the first white node t in the smallest ***PE*** *sets based on the ordered set.*

Step 2): ***IF*** *neighbors of node t are neither gray nor black, then do* ***DISCOVER*** *for node* ***t****.*

Step 3): ***IF*** *neighbors of node t are either gray or black, then do* ***FINISH*** *for node* ***t****.*

Step 4): Go back to the first node of the ordered set, do Steps 2 and 3 for each nonblack node ***t*** *until the color of any nonblack node changes. Then go to Step 5.*

Step 5): Repeat Step 4 if there exists any nonblack node in the ordered set; otherwise, stop the heuristic.

Figure 6. Node allocation steps.

Let us follow now the same example in Figure 4. The ACG in Figure 4(a) is going to be mapped onto the region R1 (shown in Figure 4(d)) which has been selected in Section 4.2. Remember that for this ACG, the smallest ***PE*** set is $S_h = \{4, 6\}$.

Assume the node ordered set is (9, 6, 7, 8, 5, 4, 1, 3, 2) based on the total communication volume; also, all nodes are initially colored white. We start with node 6, which is the first white node in S_h (*Step 1*). Since at this time its neighbors, nodes 5

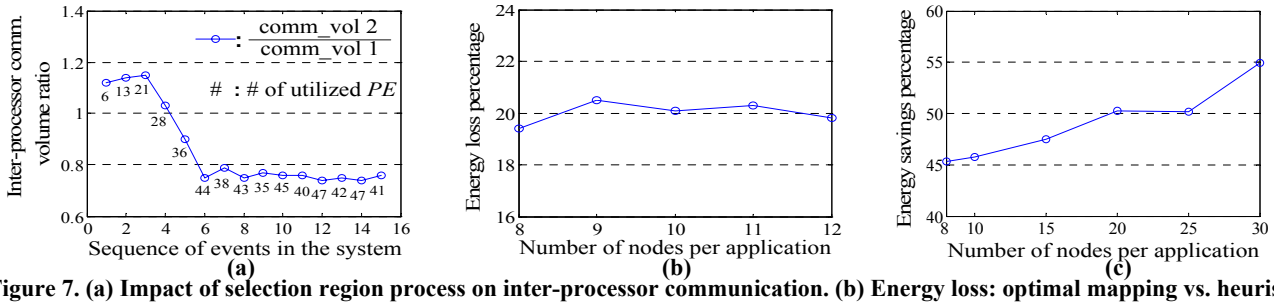

Figure 7. (a) Impact of selection region process on inter-processor communication. (b) Energy loss: optimal mapping vs. heuristic solution given a selected region. (c) Energy savings: arbitrary mapping vs. heuristic solution.

and 7, (see in Figure 4(a)) are white, we **DISCOVER** this node (*i.e.* color it gray) and select PE_{42} and PE_{43} (because these are the only *PE* locations at '*H*' voltage level in region R1) to map it (*Step 2*); namely, node 6 will be allocated onto PE_{42} **or** PE_{43} later. Then continuing with *Step 4*, we go back to the first node in the ordered set. At this moment, the color of node 9 is changed from white to gray since all neighbors of node 9 are white and we select PE_{32}, PE_{33}, PE_{41}, PE_{51}, PE_{52}, PE_{53}, and PE_{61} for it. With the following repeat of *Step 4*, the color of nodes 9 and 6 is unchanged. Then, we consider node 7; its color changes from white to black directly since nodes 6 and 9 are gray. We allocate node 7 onto PE_{52}. Then, repeat *Step 4*; node 9 becomes black since it neighbor, node 7, is colored black and we allocation node 9 onto the precise location, PE_{51}. We continue this process until all nodes are colored black and each node is allocated onto a precise *PE* location. Figure 4(d) shows the final result of node allocation process.

The total run time of our heuristic has complexity of $O(V^2+E)$. This is because the body of the loop (*Steps 4-5*) executes $|V|$ times, while reaching at most $|V|$ vertices each time. In addition, since each node will be reached at most two times (*i.e.*, DISCOVER and FINISH), and the adjacency list of each node is scanned only when the node is reached, the total time of scanning the adjacency lists is $O(E)$.

5. EXPERIMENTAL RESULTS

We first evaluate the impact of near convex region selection and node allocation heuristics using synthetic benchmarks generated with TGFF [8]. Experiments are performed on a Intel® Pentium 4 CPU (2.60GHz with 768MB memory). Our on-line algorithm is compared against the optimal and arbitrary mapping solutions. Then, in Section 5.3, the energy overhead of running our on-line algorithm is evaluated using a MicroBlaze soft processor, a 32-bit Harvard RISC architecture optimized for Xilinx Vertex-II Pro XC2VP30 FPGA.

5.1 Evaluation of region selection

To show that the choice of a near convex region heavily impacts the communication cost of the incremental mapping process, we consider the following experiment. Several sets of applications are generated using the TGFF package [8]. The node number and the communication volume are randomly generated according to some specified distributions. Then applications are randomly selected for mapping onto the system or being removed from it. For mapping onto the resulting system with pre-existing application staying fixed, two different strategies are implemented: 1) A greedy approach minimizing the inter-processor communication cost of the current configuration (but without considering the newly incoming applications) and 2) A near convex region is first selected using the proposed approach and then the application is optimally mapped onto this region using exhaustive search.

The number of nodes per application ranges between 5 and 10. The system consists of 7×7 PEs with PE_{11} being used as a global manager. The variance of the communication volume per edge in one application is set arbitrarily between 0 and 10^6. Initially, there is no application in the system. The sequence of events in the system is incremented whenever an application comes into or departs from system. If the number of idle *PE*s in the system is smaller than the number of nodes of the incoming application, then the incoming application is not accepted.

Figure 7(a) shows the inter-processor communication ratio between the mapping using strategy 1 (*i.e.,* without selecting a region) and that in strategy 2 (*with* selecting a near convex region). Here, the inter-processor communication contains all communications (*i.e.,* pre-existing and the incoming applications) in the system. The numbers below the data points stand for the number of utilized *PE*s in the system (except the GM) in that particular system configuration.

As shown in Figure 7(a), there is a slight increase in the communication ratio at the beginning because the greedy approach performs well when the number of utilized *PE*s in the system is small. Once the number of utilized PEs increases due to the incoming applications, the benefit of our proposed algorithm becomes obvious. Finally, the ratio becomes stable since for strategy 1, when the application leaves the system, there is always a scattered region left for additional mapping. This example demonstrates that near convex region selection definitely helps the incremental mapping process.

5.2 Evaluation of node allocation heuristic

We first compare the runtime and solution of our heuristic against exhaustive approach by considering the same regions. The runtime for finding the optimal mapping within selected region increases exponentially with the number of nodes in each application: For 8, 9, 10, 11, and 12 nodes in one application, it takes 0.2*sec*, 1.5*sec*, 4*min*, 10*min*, and 2*hrs*, respectively, to obtain the optimal mapping. On the other hand, the runtime of our heuristic stays *within* 40μ*sec,* when the number of nodes varies between 8 and 20. Since finding the optimal mapping for a region with 13 nodes takes more than 26*hrs*, we vary the number of nodes per application from 8 to 12 (see points on X-axis in Figure 7(b)). More specifically, there are 5 categories, (|V|=8-12), each category containing 40 applications generated with TGFF.

We denote the energy consumption of our mapping heuristic by E_h, and the energy consumption of the optimal mapping with the same region by E_e. Thus $(E_h\text{-}E_e)/E_e \times 100\%$ is the percentage of reported energy loss compared to the optimal solution. As shown in Figure 7(b), the energy loss for each category is always less than 21%. Therefore, our node allocation heuristic provides good results for large designs.

Now, we compare the arbitrary mapping against our heuristic in Figure 7(c). Since the runtime of arbitrary mapping is very fast, we can consider ACGs with a large number of nodes (*i.e.* 30 in Figure 7(c)) and see if our algorithms scale well. The number of nodes per application used in this experiment ranges between 8 and 30 (*i.e.*, 6 categories, |V|=8, 10, 15, 20, 25, and 30). We generate 20 different regions with *PE* locations corresponding to the number of nodes in each category and then run 20 different applications on each selected region. The variance of communication volume per edge in one application is still set between 0 and 10^6.

We denote the energy consumption of our heuristic mapping by E_h, and the energy consumption of the arbitrary mapping solution by E_a; then $(E_a\text{-}E_h)/E_a \times 100\%$ is the percentage of reported energy savings compared to the arbitrary mapping solutions which were averaged from 500 random results. As shown in Figure 7(c), at least 45% savings can be achieved in all categories; of note, the savings increase as the node size scales up.

5.3 Energy overhead for on-line algorithms when running real applications

The energy overhead of on-line processes contains the message transmission into the control network and our on-line algorithms (*i.e.,* near convex region selection and node allocation). The incremental mapping process is activated only when a new application arrives, and so the *PE*s need to send their status to GM. The communication volume for all control messages is [a bits (for showing *PE* address, which depends on network size) + 1 bit (*PE* status)] × *MD* (Manhattan distance of *all PE*s to GM). For the 6 × 6 network, $a = 6$ (2^6>36), *MD* = 180; therefore, all control bits for one incoming application are within 1 Kilobit. Compared to the communication volume in real applications (which is in the Megabits range), the energy overhead for transmitting the control messages is negligible.

Next, we evaluate the extra energy overhead of our on-line algorithms. Our system contains 6 × 6 *PE*s of AMD ElanSC520 (133 MHz), AMD K6-2E (500Mhz), and one MicroBlaze core (160MHz) for the global manager running our on-line algorithms. To evaluate the potential of our on-line algorithms for real applications, we apply it to embedded system synthesis benchmark suite, E3S [9]. We first do the off-line partitioning process for each benchmark. The communication energy consumption is measured by a C++ simulator using the Orion library [10]. We start with a given system configuration running a set of pre-existing applications. We denote the power consumption of our heuristic mapping as P_h, the power consumption of the arbitrary mapping solution as P_a, and the power consumption of running the on-line algorithms as $P_{on\text{-}line}$. Thus, $[(P_a \times T_{et}) - (P_h \times T_{et} + P_{on\text{-}line} \times T_{on\text{-}line})] / [(P_a \times T_{et})] \times 100\%$ is the percentage of energy savings compared to the arbitrary mapping solution, where T_{et} is the execution time of that application and $T_{on\text{-}line}$ is the execution time of running our on-line algorithms.

We observe that about 48.6% communication energy savings can be achieved, on average, compared to an arbitrary implementation when the execution time of applications is over 0.5 *sec*. The energy overhead of run-time mapping process on MicroBlaze for *telecom* and *consumer* takes 3.6*msec* and 3.1*msec*, respectively and consumes 450μJ and 384μJ.

6. CONCLUSION AND FUTURE WORK

We have proposed a run-time approach to incrementally map a number of applications onto a homogeneous tile-based NoC with multiple voltage levels. Using the near convex region selection technique, the mapping results of our heuristic can be obtained very efficiently and they are not far from the optimal ones. Moreover, additional incoming applications can be added into system more easily and large designs with arbitrary topologies can be analyzed.

We plan to extend the work in several directions. First, for SoC applications, we intend to consider more heterogeneity (*i.e.,* different cores with different functionality), while including computation energy consumption as well. Another direction is to address the run-time incremental scheduling issue and the processor sharing problem.

7. ACKNOWLEDGEMENTS

Authors acknowledge the support of the Gigascale Systems Research Focus Center, one of the five research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program. We also thank U. Y. Ogras of CMU for suggestions that improved this paper.

8. REFERENCES

[1] W. J. Dally, B Towles, "Route packets, not wires: on-chip interconnection networks," Proc. DAC, Las Vegas, NV, June 2001.

[2] J. Hu, R. Marculescu, " Energy- and performance-aware mapping for regular NoC architectures," *IEEE Trans. on CAD*, 24(4), Apr. 2005.

[3] S. Murali, G. De Micheli, "Bandwidth-constrained mapping of cores onto NoC architectures," Proc. DATE, Paris , France, Feb. 2004.

[4] L. T. Smit, *et. al*, "Run-time assignment of tasks to multiple heterogeneous processors," Progress 2004 Embedded Systems Symp., the Netherlands, Oct. 2004.

[5] H. J. Broersma, *et. al*, "The computational complexity of the minimum weight processor assignment problem," Proc. Intl. Workshop on Graph-theoretic Concepts in Computer Science, Bonn, Germany, June 2004.

[6] H. Wu, M. D. F. Wong, I-Min Liu, "Timing-constrained and voltage-island-aware voltage assignment," Proc. DAC, San Francisco, CA, July 2006.

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

[8] Task graphs for free (TGFF v3.0) Keith Vallerio, 2003. http://ziyang.ece.northwestern.edu/tgff/.

[9] R. Dick, "Embedded system synthesis benchmarks suites (E3S)," http://www.ece.northwestern.edu/~dickrp/e3s.

[10] H. Wang, X. Zhu, L. Peh, S. Malik, "Orion: a power-performance simulator for interconnection networks," Proc. Intl. Symp. on Microarchitecture (MICRO), Istanbul, Turkey, Nov. 2002.

[11] A. M. Pastrnak, P. H. N. de With, S. Stuijk, J. van Meerbergen, "Parallel implementation of arbitrary-shaped MPEG-4 decoder for multiprocessor Systems," in Proc. Visual Comm. and Image Processing, San Jose, CA, Jan. 2006.

[12] J.-M. Chang, M. Pedram, "Codex-dp: co-design of communicating systems using dynamic programming," *IEEE Trans. on CAD*, 19(7), July 2000.

[13] V. Nollet, T. Marescaux, D. Verkest, "Operating-system controlled network on chip," Proc. DAC, San Diego, CA, June 2004.

[14] P. Pop, P. Eles, T. Pop, Peng Zebo, "An approach to incremental design of distributed embedded systems," Proc. DAC, Las Vegas, NV, June 2001.

[15] J. Kao, F. B. Prinz, "Optimal motion planning for deposition in layered manufacturing," Proc. Design Engineering Technical Conf., Atlanta, GA, Sept. 1998.

[16] U. Y. Ogras, R. Marculescu, P. Choudhary, D. Marculescu, "Voltage-frequency island partitioning for GALS-based networks-on-chip," Proc. DAC, San Diego, CA, June 2007.

[17] M. A. Bender, *et. al*, "Communication-aware processor allocation for supercomputers," Proc. Workshop on Algorithm and Data Structure, Waterloo, Canada, Aug. 2005.

[18] S. Bertozzi, *et. al*, "Supporting task migration in multi-processor systems-on-chip: a feasibility study," Proc. DATE, Munich, Germany, March 2006.

[19] C. Y. Lee, "An algorithm for path connection and its applications," *IRE Trans. Electron Comput.*, vol. EC-10, pp. 346-365, Sept. 1961.