# Supporting Non-contiguous Processor Allocation in Mesh-based CMPs Using Virtual Point-to-point Links

Marjan Asadinia[1], Mehdi Modarressi[2], Arash Tavakkol[2,3], Hamid Sarbazi-Azad[2,3]
[1]Department of Computer Engineering, Sharif University of Technology, International Branch, Kish, Iran
[2]Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
[3]School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran
Asadinia@kish.sharif.edu, Modarressi@ce.sharif.edu, Arasht@ipm.ir, Azad@sharif.edu

***Abstract*— In this paper, we propose a processor allocation mechanism for run-time assignment of a set of communicating tasks of input applications onto the processing nodes of a Chip Multiprocessor (CMP), when the arrival order and execution life-time of the input applications are not known a priori. This mechanism targets the on-chip communication and aims to reduce the power and latency of the NoC employed as the communication infrastructure. In this work, we benefit from the advantages of non-contiguous processor allocation mechanisms, by allowing the tasks of the input application mapped onto disjoint regions (sub-meshes) and then virtually connecting them by bypassing the router pipeline stages of the inter-region routers. The experimental results show considerable improvement over one of the best existing allocation mechanisms.**

***Keywords-chip multiprocessors; network-on-Chip; processor allocation; contiguous allocation; non-contiguous allocation; power consumption; performance.***

## I. INTRODUCTION

In networks-on-chip (NoCs), the problem of task to node mapping, which determines on which node (or respective processing core) each task should be placed at, dramatically affects network performance characteristics, such as average inter-core distance and communication flow distributions [1]. These characteristics, in turn, determine power consumption and average network latency of the system. In [2], it has been shown that selectively mapping the tasks to NoC nodes results in a considerable power reduction and performance improvement, compared to an NoC with randomly mapped tasks.

In addition to the design-time mapping schemes, incremental mapping or processor allocation is another problem in large CMPs. Processor allocation deals with the problem of mapping the tasks of a newly arrived application to unallocated cores at run-time without interfering the execution of existing applications in the system. In this scenario, the exact sequence in which different applications arrive in the system and their lifetimes are unknown a priori.

The existing techniques of processor allocation in traditional parallel machines are generally divided into two groups: *contiguous* and *non-contiguous* [3]. In contiguous allocation techniques, the tasks of an application are mapped onto a set of adjacent network nodes which result in low communication overhead [4-5]. However, the main problem of this approach is its low system utilization, since an application has to wait for a properly sized and shaped contiguous group of processors, while there may be sufficient number of free processors in non-contiguous regions [3][6]. Non-contiguous allocation tries to mitigate this problem by mapping the tasks on multiple disjoint smaller sub-networks rather than always waiting until a single sub-network of the requested size being available.

Lifting the contiguity condition in non-contiguous allocation is expected to increase processor utilization. It may, however, lead to more traffic load and higher communication delay/power since it is very likely that the tasks of an application are mapped onto distant regions of the network and their packets have to travel a longer distance through different intermediate sub-networks.

Most existing processor allocation methods are proposed for the traditional parallel machines. Recently, few researchers have focused on processor allocation in NoC-based CMPs [7-8]. Among them, the run-time incremental mapping presented by Chou and Marculescu [8] is more related to our work. They split the mapping process into two steps: region selection and application to region mapping. The first step tries to select a near convex region in such a way that the contiguity of the remaining free nodes is maximized and subsequent applications can still select a near convex region. During region selection, a node that has most of its neighbors utilized is selected first, as it is very likely to be later isolated. Moreover, the nodes with less distance to the boundaries of the current region are more likely to be included. After region selection procedure, every task-graph vertex (application task) is assigned to some node in the region. The algorithm, in this step, aims to minimize the inter-processor communication. It also tries to meet task deadlines by assigning tasks to a node with appropriate voltage level.

In this paper, we propose a processor allocation scheme which aims to benefit from the good resource utilization of non-contiguous allocation schemes while behaving close to contiguous allocation schemes. Our method relies on forming virtual contiguous regions which may be composed of several non-physical contiguous groups of processors. The NoC then provides low-power and low-latency communication for the packets when traveling among different parts of a virtual region.

Our proposed approach adopts a hybrid NoC supporting a second low-power and low-latency switching method in addition to packet switching. In such NoCs, we direct communication among the nodes inside a region (or sub-network) through packet-switching, as this switching method benefits from flexibility, scalability and high resource utilization. The inter-region traffic, however, is handled by the second switching method to hide the latency of communication flows between the nodes in disjoint parts.

Investigating several hybrid NoCs with efficient switching methods, such as EVCs [9], VIPs [10], and hybrid packet/circuit-switched NoCs [11], we employ the idea presented in [10] for underlying NoC structure. In [10], virtual point-to-point (VIP) connections are integrated into a conventional packet-switched NoC to support high volume communication flows which can work better for communications between distant nodes. VIPs can be constructed in demand between any two NoC nodes at run-time and involve negligible area overhead.

VIPs give a power and latency close to physical dedicated point-to-point links [10]. They provide low-power and low-latency virtual dedicated paths between any two nodes by bypassing the pipeline of the intermediate routers. In this design, one virtual channel (VC) of each physical channel is designated to bypass the router pipeline stages. VIPs provide connection-oriented communication and, once created, their paths are stored in the routers along the path. In the VC assigned to VIPs, the buffer is replaced by a register (1-flit buffer) which holds the flits arriving on the VC. In addition to the reduced power and latency, VIPs isolate the inter-region from the intra-region traffic which allows applying better traffic management (such as bandwidth allocation and priority assignment) schemes.

In this paper, we introduce a VIP-enabled NoC architecture and show how it can be exploited in order to form virtual regions and allocate them to the new coming applications, in such a way that the total on-chip power and average packet latency is minimized. We use a centralized scheme to perform job mapping [12-13]. Since, most current MPSoCs and CMPs have a central root or configuration process (or processor) that initializes, configures, and manages the system [14], we consider the task of processor allocation is realized by this processor. Generally, a centralized managing scheme may suffer from scalability problems, but this scheme can efficiently manage processor allocation, since the manager is activated only when a new application arrives at the system; in a realistic scenario, arrival of new applications and completion of running applications do not occur frequently.

The rest of the paper is organized as follows. Section 2 introduces the NoC architecture used in this work. We present the proposed run-time processor allocation algorithm in Section 3. In Section 4, the proposed algorithm is compared to a state-of-the-art allocation algorithm in terms of power consumption and network performance. Finally, Section 5 concludes the paper.

## II. NoC Architecture

The routers in our design are baseline wormhole-switched routers which are slightly modified in order to support VIP connections. In this section, first, we briefly describe the VIPs. We refer interested readers to [10] for more details on VIPs. Although the NoC we consider in our work is not restricted to a specific switching and routing scheme, the NoC router, in this study, employs a wormhole switching mechanism with speculative 4-stage pipelined routers [15]. A state-of-the-art router speculatively performs virtual-channel allocation (VA) and switch allocation (SA) in parallel resulting in 4-stage routers, including buffer write (BW), route computation (RC), switch and virtual-channel allocation (SA+VA), and switch traversal (ST) stages. In this scheme, the routers speculate that a waiting packet will succeed in output VC allocation stage. Non-speculative requests (flits which have already been allocated a VC) are prioritized over speculative ones, so performance degradation may be caused by unsuccessful speculations.

VIPs can be considered a kind of circuit-switching scheme. However, they do not suffer from resource utilization and performance degradation (due to the circuit setup time) issues of the conventional circuit-switching. They can be added to a packet-switched NoC in order to improve its performance.
Here, in one virtual channel (virtual channel 0) of each physical channel, the buffer is replaced by a register (1-flit buffer). These virtual channels are devoted for establishing VIP connections between any two given nodes. The switch allocator unit is also slightly modified and each output port contains a VIP allocator that, for each output port belonging to a VIP, determines (using a 2-bit register) the input port connected to this output port along the VIP. There is no need for arbitration among VIPs, as VIP connections are not allowed to share the same links (i.e. each router port can be used by at most one VIP connection).

A VIP for a communication flow is established by appropriately setting the VIP allocators in each router in such a way that the VIP registers in each router are connected to proper output ports and then to the registers in the next router along the VIP. As a result, at each hop, the VIP data are forwarded in two stages: crossbar traversal and link traversal, with VIP registers acting as staging registers. Since the VIP flits bypass the other router pipeline stages, the power consumption and delay related to buffer read and write, route calculation, and arbitrations are removed. Unlike dedicated point-to-point links, which are physically established between the communicating nodes of a multi-core chip (based on the communication pattern of the target application) and are fixed during the system life-time, the VIP connections are dynamically reconfigurable and can be established based on the traffic pattern exhibited by the running application. It is a critical feature for processor allocation algorithms, as the VIP connections can be built and torn down when the applications start and stop execution.

Supporting VIPs in a baseline packet-switched router involves minor manipulating the baseline router architecture and impose a completely negligible area overhead [10]. VIPs do not suffer from the resource utilization problems of TDMA-based circuit-switched NoCs, since NoC cycles (time slots) are not reserved for the two NoC parts (packet-switched part and VIPs). Instead, each part uses the links when it has incoming flits to forward and the flow control and bandwidth allocation mechanisms developed for them guarantees that each part is provided with the required bandwidth.

## III. VIP-SUPPORTED PROCESSOR ALLOCATION

### A. Overview of the Approach

In this section, we describe how to exploit the capabilities of the NoC described in Section 2 in order to find an appropriate solution for the processor allocation problem. The solution should minimize the on-chip communication power and latency and increase resource utilization by considering all unallocated processor for placing a new task.
The NoC is assumed to be an $m \times n$ mesh with the router architecture described above. Each input application is described as a *Communication Task Graph* (CTG). The CTG is a directed graph $G\ (V,E)$, where each $v_i \epsilon V$ represents a task, and a directed edge $e_{i,j} \epsilon E$ characterizes the communication from $v_i$ to $v_j$. The communication volume (bits per second) corresponding to edge $e_{i,j}$ is denoted by $t(e_{i,j})$.

Simply stated, for a new input application, our objective is to map the tasks of the application into different nodes of a mesh-connected NoC in such a way that the total on-chip power and latency is minimized. To this end, the algorithm tries to map the tasks onto a contiguous sub-network. However, if it is not possible to find a contiguous region with sufficient number of processors, the tasks are mapped onto some non-contiguous sub-networks which use VIPs to communicate with each other.

As mentioned before, the allocation process is conducted by a root processor and in a centralized manner. The root process keeps the NoC state, i.e. the location of all existing applications, the traffic assigned to each NoC link, and the path of all existing VIP connections. It also keeps the state of all NoC nodes and recognizes which nodes are free or allocated. The root process is activated when a new application arrives at the system or one of the running applications completes its execution. It conducts allocating and releasing resources and then updates the NoC state. Since the root process performs the allocation and de-allocation of processors, it knows the characteristics of the on-chip traffic (VIP paths and link loads) and availability of nodes. Thus, it has all the required NoC state information and do not need to collect them from the NoC.

### B. The Proposed Algorithm

The processor allocation is accomplished in some steps as follows.

*1) Partition the CTG vertices into two sets:* Mapped (containing the already mapped CTG vertices and is initially empty) and Unmapped (which includes the CTG vertices not yet mapped and initially contains all CTG vertices). Once a vertex is mapped, it is moved to the Mapped set.

*2) Similarly, partition the NoC nodes into two sets:* Allocated (including the nodes to which a CTG node has been already assigned) and Unallocated (which includes the free nodes). Unallocated nodes may be located in some non-contiguous regions, or free contiguous regions.

*3) Find a free region with equal (or the nearest larger) size to the size (number of vertices) of the application. If there does not exist a contiguous group of free processors with the size equal to or larger than the CTG size, find a region with nearest smaller size. Map the task with the maximum communication demand onto one of the region nodes with maximum number of neighbors. The communication demand of a vertex is defined as the cumulative weight of the incoming and outgoing edges connected to it.*

*4) Repeat the following steps until the Unmapped set becomes empty, i.e. all CTG vertices are mapped:*

*a) Select the core $v_i$ that communicates most with the Mapped members.*

*b) Consider the smallest convex region that cover all Mapped members $v_j$ which communicate with $v_i$ ($e_{i,j} \epsilon CTG$). We refer to this convex region as search region (Fig. 2). Examine all unallocated nodes inside search region for placing $v_i$. For each selected node x:*

- Select the CTG edges between the selected core $v_i$ and each *Mapped* member $v_j$, which is mapped onto NoC node *y*, in order of their communication volumes.
- The communication of each node with its adjacent nodes (neighboring nodes at distance 1) is handled by the packet-switched network. However, each node requests for a VIP to communicate with farther nodes, if the destination task is not on an adjacent node. Consequently, if the distance between *x* and y is 1, consider $e_{i,j}$ for using only packet-switched network, otherwise in addition to packet-switching, examine the possibility of constructing a VIP for the edge (i.e. the existence of sufficient unused resources) along one of the shortest paths between *x* and *y*. However, if a VIP for such edges could not be found, they should be handled by the packet-switched network. Map the core onto the node which minimizes

$$\sum_{\forall v_j \in V \text{ connected to } v_i} t(e_{i,j}) \times dist(x, M(v_j)) \times (VIP(e_{i,j}) \times W_{vip} + (1 - VIP(e_{i,j})) \times W_{ps}) \quad (1)$$

where *dist(a,b)* is the Manhattan distance between NoC nodes *a* and *b, $M(v_j)$* is the NoC node onto which the CTG vertex $v_j$ is mapped, and $t(e_{i,j})$ is the communication volume of edge $e_{i,j}$. $VIP(e_{ij})$ is defined as

$$VIP(e) = \begin{cases} 1, & if\ \ a\ \ VIP\ \ is\ \ reserved\ \ for\ \ e \\ 0, & otherwise \end{cases} \quad (2)$$

$W_{ps}$ and $W_{vip}$ are two weights that reflect the cost of traversing one hop over the VIP and packet-switched network, respectively. Based on evaluating the power consumption of VIPs and packet-switching using Orion power library [16], we roughly set $W_{ps}$ and $W_{vip}$ to 5 and 3, respectively.

If all nodes inside *search region* are already allocated or cannot be used due to bandwidth limitation of links, examine the nodes within the fixed local neighborhood of *d* hops of the region boundary nodes (Fig. 1). For each value of *d,* all possible nodes at that neighborhood are examined within this step. The value of *d* starts from 1 and the nodes within each neighborhood are handled in a separate round of this step. *d* increases by 1 until a free node is found.

The path should not violate the bandwidth of the network links (to avoid congestion). More formally, the bandwidth constraint of each NoC link $l_k$ must be satisfied as

$$\forall l_k, BW(l_k) \geq \sum_{\forall e_{i,j} \in E} X^k(i,j) \qquad (3)$$

where $BW(l_k)$ is the bandwidth of link $l_k$ and $X^K(i,j)$ and is obtained by

$$X^k(i,j) = \begin{cases} t(e_{i,j}), & if \quad l_k \in \quad path(e_{i,j}) \\ 0, & otherwise \end{cases} \qquad (4)$$

where $path(e_{i,j})$ represents the set of links onto which CTG edge $e_{i,j}$ (with volume $t(e_{i,j})$) is mapped either on a VIP or on a packet-switched part.

After finding a route for all CTG edges, VIP connections are constructed by setting the VIP allocators, while packet-switched routes are set up by appropriately setting the routing table of the routers.

Although the objective of this algorithm is defined to optimize the power consumption, it is obvious that by mapping the tasks on nearby nodes and also not allowing the bandwidth of each link to be violated, the algorithm improves the average packet latency as well.

The algorithm proposed in this paper is simple and compared to other related work [7][8], involves moderate computation to perform mapping. Consequently, according to the run-time and power analysis in [8], the run-time of our algorithm is in the order of microseconds and hence can conduct the processor allocation procedure at real-time.

## IV. Evaluation

### A. Simulation Environment

In this section, we evaluate the proposed processor allocation method. We compare our processor allocation algorithm against the allocation scheme presented in [8] referred as *incremental mapping*. This allocation scheme is one of the most recent processor allocation approaches shown to have good performance with respect to other existing methods [8]. It works on NoCs with several voltage levels, but we set the voltage of all nodes to the same value. It works on a

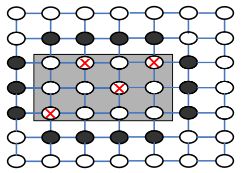


Figure 1. The shaded area represents *search region* for a new task which communicates with the tasks placed on the nodes marked by X. The black nodes are the nodes within the fixed local neighborhood of 1 hop of the region boundaries.

mesh NoC with 4-stage speculative pipelined routers. The packet-switched part of our NoCs uses the same router architecture. Both allocation schemes are implemented in XMulator [17], an event driven flit level NoC simulator coded in C++. The simulator is augmented by Orion power library in order to evaluate the power consumption of the NoCs under the considered allocation strategies. The power results reported by Orion are based on a 64-bit NoC implemented in 65nm technology and the working frequency of the NoC is set to 2 GHz.

We use SPLASH-2 [18] traffic traces to evaluate our method. The traces are collected for different application sizes. The size varies from 3×3 to 7×7. The NoC size for this experiment is 10×10. We construct the communication task-graph of each application by profiling the inter-node communications in advance.

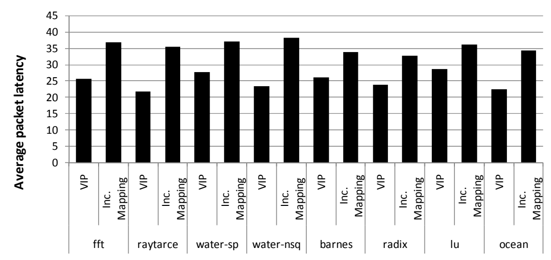
### B. Experimental Results

In the first scenario, we assume that all the applications run on the system are the same, but with different sizes. There are two packet sizes: 3 and 5 flits corresponding to the request and reply packets and the NoC buffers are 8-flit deep. The NoC with the incremental mapping uses 2 virtual channels per physical channel. Our NoC replaces one of the virtual channels with 1-flit VIP buffers and uses one virtual channel and one VIP latch per port.

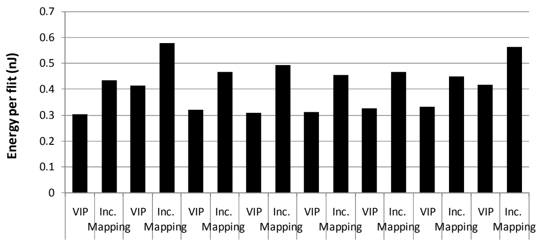
In these experiments, we generate a random application size between 3×3 and 7×7 tasks. The application lifetime, i.e. the number of cycles it is running on the system is also selected randomly between 150000 and 200000 cycles. If there is sufficient number of free nodes, the application is mapped into the NoC and this procedure continues with a new application. Otherwise, the application waits until a running application is finished and the NoC has enough nodes to host it. We refer to the rate of application acceptance obtained by this method as $\lambda_{full}$.

Fig. 2.a displays the average packet latencies given by the VIP-based and the incremental mapping algorithms for different SPLASH programs where the applications with random sizes enter the system with rate $\lambda_{full}$ during 10 million cycles of simulation. Both networks work under the same input scenario. As Fig. 2.a shows, the proposed method outperforms the other considered method by up to 30% (26% on average) across all of the benchmarks.

The energy per flit consumed by the networks is depicted in Fig. 2.b and shows that following the same trend as the latency, our method gives up to 33% (29% on average) reduction in energy consumptions, compared to the other



(a)



(b)

Figure 2. The average network latency (a) and energy per flit (b) of the proposed processor allocation and the incremental mapping.

considered allocation algorithm. The obtained improvements are mainly due to the fact that the VIPs forward a significant portion of the inter sub-network traffic over short-cut paths, thereby the power-hungry buffering and time-consuming allocation operations are removed.

We then consider another scenario in which the NoC accepts a combination of the SPLASH applications. In this scenario, in addition to the application size and life-time, the application type is also selected based on a uniform distribution.

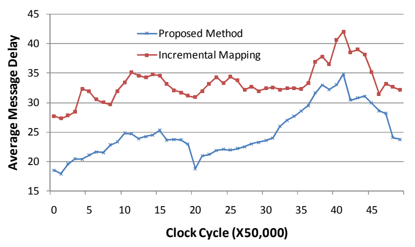
Fig. 3 shows the latency and energy consumption of the two NoCs along the time. In these figures, each data point stands for the average energy per flit and latency of the network in a 2.5 million cycle time intervals.

As the Fig. 3 shows, our method always outperforms the other considered processor allocation method and decreases the energy consumption. When some old applications leave the system, the free processors form scattered and non-contiguous regions. In this case, the capability of the VIPs in virtually connecting these regions is used to optimize the power and performance of communication.

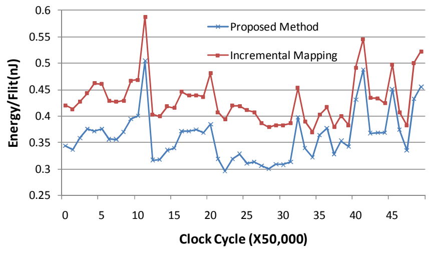
In these experiments, when larger applications arrive at the system, the improvement obtained by our proposed methodology increases. This is due to the fact that large applications are more likely to be mapped onto several non-contiguous regions and benefit more from the VIPs.

Table 1 shows the minimum, maximum, and average VIP length and the minimum, maximum and average percentage of the traffic handled by VIPs during the system life-time for the above scenario (reported in Fig. 3) across all data points. As mentioned before, VIPs absorb a significant portion of the on-chip traffic and handle it with lower power consumption and latency.

To study the impact of the buffer depth on the obtained power and performance improvement, we repeat the experiments with the previous parameters, but with a buffer depth varying from 2 to 40 flits. Fig. 4 displays the results under a random combination of SPLASH applications with the arrival rate of $\lambda_{full}$.



(a)



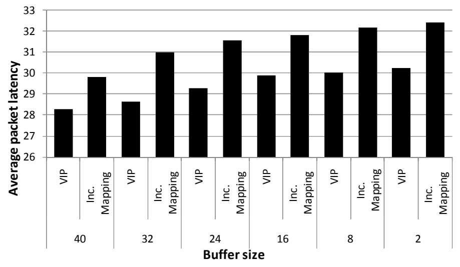
(b)

Figure 3. The average network latency and energy per flit of the proposed processor allocation and the incremental mapping.

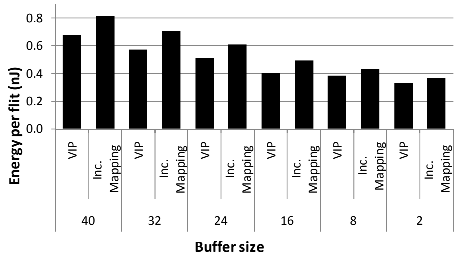
Table I. The traffic of the network handled by VIPs and the length of the formed VIPs

| | Max. | Min. | Average |
|---|---|---|---|
| **VIP Traffic Coverage (%)** | 49% | 22% | 34% |
| **VIP Length (Hop)** | 8 | 3 | 4.84 |

As the fig. 4 shows, our NoC provides lower latency and energy values for all buffer sizes. However, as the buffer size is increased, the obtained energy improvement increases. When the buffer depth is increased, the performance is improved as a result of the increased NoC capacity to accept more traffic. However, it also increases energy consumption, as larger buffer arrays consume more power. In our NoC, a portion of the traffic is handled by VIPs and does not pass through buffers, so the negative effect of the increased buffer size on the energy consumption is moderated.

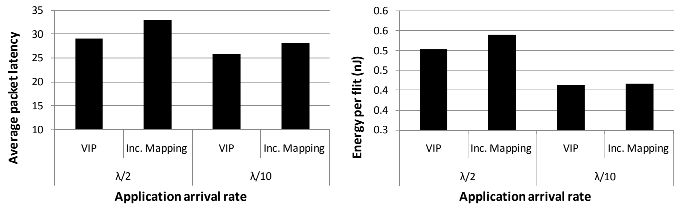


(a)



(b)

Figure 4. The average network latency (a) and energy per flit (b) of the proposed processor allocation and the incremental mapping for different buffer sizes.



(a) (b)

Figure 5. The average network latency (a) and energy per flit (b) of the proposed processor allocation and the incremental mapping for two different arrival rates.

We have also studied the effect of the average arrival rate of the application on the obtained performance improvement. For this purpose, we consider the random combination of SPLASH programs and calculate $\lambda_{full}$. Then, we set the arrival rate of the applications to $\lambda_{full}/10$ and $\lambda_{full}/2$ and repeat the experiments.

That is the next application to be run, its size, and lifetime are generated like the previous experiments, but the arrival rate is determined based on a Poisson distribution with the mentioned rates. Fig. 5 outlined the performance and power improvements given by our algorithm over the incremental mapping algorithm for these scenarios. The figure indicates that for low arrival rates, the energy consumption and latency difference of the two networks is not big.

At lower arrival rates, the network has more number of free processors on average, and it is more likely for both algorithms to find a contiguous group of free processors to allocate to a new application and thus the advantages of VIPs in forming virtual contiguous regions is not fully exploited.

## V. Conclusion

This paper targeted the well-known processor allocation problem in NoC-based CMPs. Processor allocation or incremental application mapping deals with run-time assignment of a set of communicating tasks of an input application onto unallocated nodes of a CMP when the arrival order and execution lifetime of the input applications are not known a priori. Focusing on on-chip communication, we used a hybrid switched NoC in which packet-switching and virtual point-to-point connections are integrated into the same NoC. The tasks of a single application are mapped onto a virtual region which may consist of several non-contiguous groups of processors. The virtual point-to-point links guarantee low-latency and low-power communication among different parts of a virtual region, while packet switching part handles the traffic inside each contiguous region. This mechanism benefits from the advantages of both traditional contiguous and non-contiguous processor allocations mechanisms. The experimental results showed considerable improvement over one of the best existing allocation mechanisms.

Integrating task migration into our algorithm in order to achieve better mappings can be considered as a future work. Designing a light-weight parallel architecture for implementing this algorithm in hardware is another work in this line. The designed logic can be integrated onto the CMP and boost the performance.

## Acknowledgment

The research of H. Sarbazi-Azad and M. Modarressi was supported in part by Iran Telecommunication Research Center (grant no. t/۵۰۰/۱۱۳۱۴).

## References

[1] A. Jantsch, and H. Tenhunen, Networks on Chip, Kluwer Academic Publishers, 2003.

[2] J. Hu, and R. Marculescu, “Energy- and Performance-aware Mapping for Regular NoC Architectures”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 24, No. 1, 2005, pp. 551-562.

[3] S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh, and L. M. Mackenzie, “Non-contiguous Processor Allocation Strategy for 2D Mesh Connected Multicomputers based on Sub-meshes Available for Allocation”, in *Proc. of International Conference on Parallel and Distributed Systems (ICPADS'06)*, 2006.

[4] K. Li and K.-H. Cheng, “A Two-Dimensional Buddy System for Dynamic Resource Allocation in a Partitionable Mesh Connected System”, *Journal of Parallel and Distributed Computing,* Vol. 12, No. 1, pp. 79-83, 1991.

[5] B. S. Yoo, C. R. Das, “A Fast and Efficient Processor Allocation Scheme for Mesh-Connected Multicomputers”, *IEEE Transactions on Parallel & Distributed Systems,* Vol. 51, No. 1, pp. 46-60, 2002.

[6] I. Ismail and J. Davis, “Program-based static allocation policies for highly parallel computers”, in *Proc.s of the IPCCC 95,* pp. 61-68, 1995.

[7] M. Faruque, R. Krist, and J. Henkel, “ADAM: Run-time Agent-based Distributed Application Mapping for on-chip Communication”, in *Proc. of DAC,* 2008, pp. 760-765.

[8] C. Chou, and R. Marculescu, “Incremental run-time application mapping for homogeneous NoCs with multiple voltage levels”, *in Proc. of the int. Conf. on Hardware/software Codesign and system synthesis,* pp. 161-166, 2007.

[9] A. Kumar, L. S. Peh, P. Kundu, and N. K. Jha, “Express virtual channels: towards the ideal interconnection fabric”, in *Proc. of the International Symposium on Computer Architecture (ISCA),* 2007, pp.150-161.

[10] M. Modarressi, A. Tavakkol, H. Sarbazi-Azad, " Virtual Point-to-Point Connections for NoCs", in *IEEE Transactions on Computer-Aided Design for Integrated Circuits and Systems (IEEE TCAD),* Vol. 29, No. 6, June 2010.

[11] M. Modarressi, H. Sarbazi-Azad, and M. Arjomand, “An SDM-Based Hybrid Packet-Circuit-Switched On-Chip Network”, in *Proc. of Design, Automation and Testing in Europe Conference (DATE),* 2009.

[12] L. Smit, G. Smit, J. Hurink, H. Broersma, D. Paulusma, and P. Wolkotte, “Run-time mapping of applications to a heterogeneous reconfigurable tiled system on chip architecture”, in *Proc. of the IEEE int. Conf. on Field-Programmable Technology,* pp. 421-424, 2004.

[13] E. Carvalho, N. Calazans, and F. Moraes, “Heuristics for dynamic task mapping in NoC-based heterogeneous MPSoCs”, in *Proc. of the 18th IEEE int. workshop on Rapid System Prototyping,* pp. 34-40, 2007.

[14] K. Goossens, J. Dielissen, and A. Radulescu, “The Æthereal Network on Chip: Concepts, Architectures, and Implementations”, in *IEEE Design and Test of Computers,* Vol. 22, No. 5, Sept-Oct 2005, pp. 414-421.

[15] W. J. Dally, and B. Towles, Principles and practices of interconnection networks, Morgan Kaufmann Publishers, 2004.

[16] A. Kahng, B. Li, L. Peh and K. Samadi, “ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration“, In *Proceedings of Design Automation and Test in Europe (DATE),* France, 2009.

[17] Xmulator NoC Simulator, from http:// www.xmulator.org, 2009.

[18] SPLASH-2, http://www.flash.stanford.edu/apps/SPLASH/.