# Smart Hill Climbing for Agile Dynamic Mapping in Many-Core Systems

Mohammad Fattah, Masoud Daneshtalab, Pasi Liljeberg, Juha Plosila
Department of Information Technology, University of Turku, Turku, Finland
{mofana, masdan, pakrli, juplos}@utu.fi

## ABSTRACT

Stochastic hill climbing algorithm is adapted to rapidly find the appropriate start node in the application mapping of network-based many-core systems. Due to highly dynamic and unpredictable workload of such systems, an agile run-time task allocation scheme is required. The scheme is desired to map the tasks of an incoming application at run-time onto an optimum contiguous area of the available nodes. Contiguous and un-fragmented area mapping is to settle the communicating tasks in close proximity. Hence, the power dissipation, the congestion between different applications, and the latency of the system will be significantly reduced. To find an optimum region, we first propose an approximate model that quickly estimates the available area around a given node. Then the stochastic hill climbing algorithm is used as a search heuristic to find a node that has the required number of available nodes around it. Presented agile climber takes the steps using an adapted version of hill climbing algorithm named Smart Hill Climbing, SHiC, which takes the run-time status of the system into account. Finally, the application mapping is performed starting from the selected first node. Experiments show significant gain in the mapping contiguousness which results in better network latency and power dissipation, compared to state-of-the-art works.

**Categories and Subject Descriptors:** D.4.7 [**Operating Systems**]: Organization and Design – *Real-time systems and embedded systems.*

**General Terms:** Algorithms, Management, Performance, Design.

**Keywords:** On-chip many-core systems, Application mapping, Task allocation, AI algorithms, Hill climbing.

## 1. INTRODUCTION

The Future Multi-Processor Systems-on-Chip (MPSoCs) are likely to have tens or hundreds of resources connected together. Networks on chip [1] (NoCs) have emerged as a promising solution for communication infrastructure of such systems. NoCs provide a regular platform for connecting the system resources and makes the communication architecture scalable and flexible compared to traditional bus or hierarchical bus architectures.

Many-core systems will feature an extremely dynamic workload where an unpredictable sequence of different applications enter and leave the system at run-time. In order to handle the featured dynamic nature, a run-time system manager is required to efficiently map an incoming application onto the system resources [2]–[4]. Applications are modeled as a set of communicating tasks, and the mapping function of the central manager (CM) of the system decides on the appropriate node for each task.

The system performance is significantly influenced by the utilized mapping approach. For example, assume a dispersed application mapping where tasks of an application are mapped onto a distant and fragmented set of nodes. This increases the average hop count within tasks of an application and places tasks of different applications between each other. Accordingly, the power dissipation and congestion probability (latency) of the network will dramatically increase. On the other hand, consider a convex and contiguous application mapping where tasks of an application are placed on relatively close nodes without fragmentation. This will reduce the average distance between mapped tasks and isolates the communication of different applications from each other which decreases the congestion probability between them.

To have an efficient mapping, it is desired to place an application on a near convex and contiguous set of nodes where the remaining set of available nodes is also kept contiguous. Finding a convex region of nodes is a polynomial, $O(n^3)$, problem [5]. However, considering more aspects such as near-convexity condition, remaining nodes contiguity, etc. turns it into clustering problems [6] with deterministic algorithms of NP-hard complexity. Indeed, these are not tolerable time complexities regarding the growing size and dynamic nature of the systems.

In this work, we propose an algorithm with significantly decreased complexity in order to find the appropriate area for a given application. The task allocation is then efficiently carried out through our *CoNA* method [4]. Briefly stated, *CoNA* starts from a *first node* and attempts to map the application tasks onto a set of contiguous nodes around it. Thus, the job of finding the appropriate area is to select a *first node* which has the required number of contiguous and near convex available nodes around it and leads to least fragmentation of remaining nodes. For this, we first propose an approximate model to estimate the number of the available (contiguous and near convex) nodes around any given node. Then, hill climbing search heuristic is adapted in order to find the optimum *first node* rapidly among all the available nodes. The proposed Smart Hill Climbing, *SHiC*, is equipped with a level of intelligence in which the steps are not taken fully stochastic. The proposed near optimal (non-deterministic) solution tackles the problem in $O(\sqrt{n})$, n is the given network size, in the best case and $O(n^2)$ in the worst case; which is an impractical case.

The rest of the paper is organized as follows: In Section 2 we present related works and motivate the impact of the optimum area (*first node*) selection. Section 3 formally defines the mapping problem and different metrics to evaluate the mapping results. Section 4 describes our *SHiC* heuristic in the *first node* selection. The simulation details along with the experimental results are presented and discussed in Section 5. Finally, Section 6 concludes the paper.

## 2. RELATED WORK AND MOTIVATION

There are several works dealing with dynamic management of workload in multi- and many-core systems. In this Section we explore different *first node* or generally the mapping area selection methods used by different works. We also motivate the importance of the selection method by presenting several examples.


*DAC '13*, May 29 - June 07 2013, Austin, TX, USA.

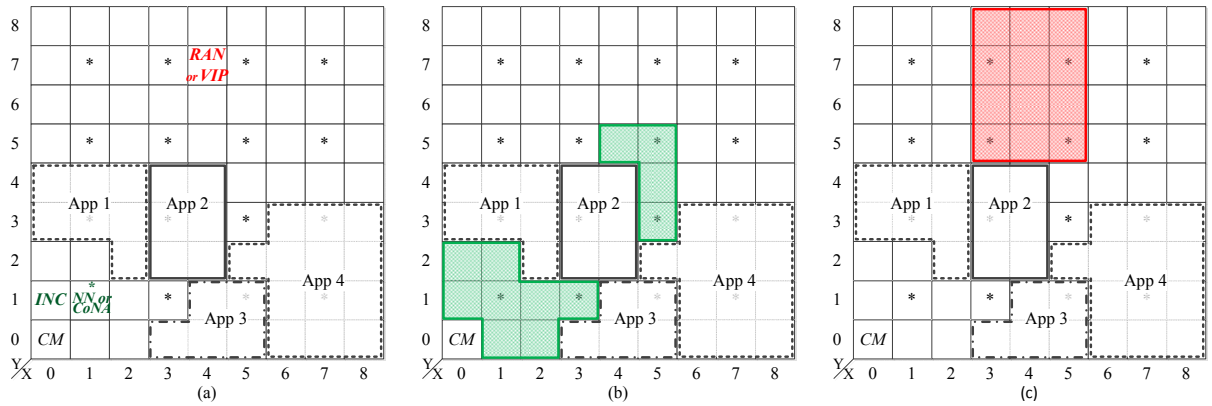


**Figure 1. Possible scenarios resulting in the area dispersion. (a) Initial configuration of the system and the *first node* for the next application in different methods. (b) Mapped area (green hatched area) dispersion in *NN*, *INC* and *CoNA* methods. (c) Contiguous mapping of *DistRM*, *STDM* and *VIP*-supported approaches, however they lead to fragmentation of remaining nodes.**

Carvalho et al. [2], [7] present different heuristics, such as Nearest Neighbor (*NN*), Best Neighbor (*BN*), etc. In all of their heuristics, a clustering mechanism for the *first node* selection is considered. A set of sparse nodes (cluster nodes) are assumed to select the *first node* of the mapping algorithm among them; i.e. a mapping solution can exist only when a free cluster node exists. This is to assure some amount of available nodes around the chosen *first node*.

Chou et al. [3], in their incremental (*INC*) approach, break down the mapping problem into two steps: the region selection, and the task allocation. In the region selection step, they start from the closest node to the CM and include it in the region. Then, they iteratively add nodes to the selected region trying to keep both the selected region and remaining nodes contiguous. Afterward, in the task allocation step, application tasks are mapped inside the selected region.

As an advanced approach, CoNA [4] selects the closest node to the CM with all its four neighbors available. Thus a minimum number of available nodes are assured. Then, task graph is traversed in breath-first order and tasks are mapped onto the neighboring of their parents in which a smaller square is formed.

There are lots of possible scenarios in which *NN*, *INC* and CoNA methods would easily result in area fragmentation. Figure 1 (a) and (b) depict one of those scenarios. Notice that the cluster nodes of the *NN* algorithm are indicated by asterisks. The current status of the system is shown in Figure 1 (a), where four applications are running on the system after entrance and exit of several applications. As indicated in the Figure 1 (a), the selected *first node* for the next application will be nodes (1, 1) or (0, 1) for *CoNA* and *NN* or *INC* algorithms, respectively. Now, an application with 12 tasks enters the system while there are only 8 contiguous nodes available regarding the selected *first node*. Accordingly, a fragmented area of nodes (the hatched area in Figure 1 (b)) will be chosen for the application. Consequently, as can be seen, the communication paths between dispersed nodes will be stretched and can be congested by communications of other existing applications (1 to 3) .

As a decentralized mapping algorithm for tree-structured applications, Weichslgartner et al. [8] suggest three different methods for the *first node* selection. Two of them need prior knowledge about all incoming applications, while the one which is more dynamic, the *farthest-away* algorithm, leads to inefficient results as it does not consider the size of applications.

Both the decentralized *DistRM* [9] and *STDM* [10] approaches, start mapping from a random node. The randomly chosen node in *DistRM* starts exploring its neighboring nodes as well as distant nodes in the system to find the node with enough surrounding resources to start allocation. To this end, *DistRM* utilizes a negotiation algorithm in cooperation with a proposed distributed directory service. On the other hand, the randomly chosen node in *STDM* tries to map the incoming application around it. In case of failure a new random node is chosen until several times. Both methods impose huge negotiation traffic on the network in order to compensate the applied randomness.

Asadinia et al. [11], in their VIP-supported approach, find all contiguous regions of free nodes, and select the smallest region with the size greater than or equal to the application size. Then, they start mapping from one of the selected region nodes with the maximum number of neighbors. This is to guarantee the contiguity of the mapped region and select a *first node* with maximum available nodes around it.

As another possible scenario, let us assume that in the system status of Figure 1 (a), the node (4, 7) is chosen by *VIP* as it is one of the nodes with maximum number of neighbors, *DistRM* or *STDM* after several random node generations. The selected *first node* has enough resources around it for the proposed application with 12 tasks and, as shown in Figure 1 (c), a contiguous set of nodes are allocated to the application tasks. However, the remaining set of available nodes is dispersed and the future applications may suffer from the area fragmentation of the available nodes.

Note that optimum *first node* selection is not only a function of the current state of the system but also the size of the application. For instance, *CoNA*, *NN* and *INC* approaches could have worked optimally, in the described scenario, if the size of the application had been less than or equal to 8. Moreover, the area selection method not only has to be agile but also must avoid random trial and error approaches. This is to minimize time and communication penalties imposed onto the system.

In this work, we propose an algorithm for optimum *first node* selection in order to achieve an agile and efficient mapping of applications. It uses the general knowledge of currently running applications and imposes no additional traffic on the network.

# 3. DEFINITIONS

In the following, we present formal definitions of an application, the NoC architecture, and the mapping problem. In order to reduce the problem size and simplify the analysis, we consider a homogenous mesh-based NoC in our definitions and experiments. Moreover, we define several evaluation metrics as assessment tools to compare different algorithms. The metrics weigh the resulted mapping of an application. Later in Section 5, we evaluate effect of our *first node* selection method, *SHiC*, on the network performance using these metrics along with extracted results of the network simulation.

## 3.1 Problem Definition

Mapping algorithms try to allocate system resources, connected together through an on-chip network, to tasks of a requested application in an optimal way.

Each application in the system is represented by a directed graph denoted as a task graph $A_p$ =*TG(T, E)*. Each vertex $t_i \in T$

represents one task of the application $A_p$, while the edge $e_{i,j} \in E$ stands for a communication between the source task $t_i$, and the destination task $t_j$. Task graph of an example application with 6 tasks is shown in Figure 2. The amount of data transferred from a task $t_i$ to $t_j$ of edge $e_{i,j}$ is indicated on the edge as $w_{i,j}$.

An architecture graph *AG(N, L)* describes the communication infrastructure, which is a simple $M \times M$ 2D-mesh NoC with the XY routing (Figure 3 (a)). The *AG* contains a set of nodes $n_{x,y} \in N$, connected together through communication links $l_k \in L$. Each node $n_{x,y}$ contains a 5-port router $r_{x,y}$ connected to the local processing element $pe_{x,y}$ by its local port.

Mapping of an application onto the NoC-based multi-core system is defined as a one-to-one mapping function from the set of application tasks *T*, to the set of NoC nodes *N*:

$$map: T \rightarrow N, s.t.\ map(t_i) = n_{x,y}; \forall t_i \in T, \exists n_{x,y} \in N \quad (1)$$

Based on the definition, a mapping function is started if and only if there are enough available nodes to map onto them. Figure 3 (a) illustrates a possible mapping of the application in Figure 2, onto the described NoC platform. For simplicity, we denote a node where a task $t_i$ is mapped onto as $nt_i$ and the packet corresponding to the edge $e_{i,j}$ as $pck_{i,j}$; i.e. the packet sent from $nt_i$ to $nt_j$. The set of running applications on the system is also denoted by *APPS* which changes at run-time due to the system's dynamic nature.

Note that the cardinality of a set shows the number of elements in that set, while the cardinality of a number means the absolute value of that number; e.g. $|APPS|$ means number of running applications, while $|-4|$ equals the number 4.

## 3.2 Evaluation Metrics

### 3.2.1 Average Weighted Manhattan Distance

The dissipated energy related to a packet delivery is a function of both the packet size, and the path the packet traverses [12]. As a metric to evaluate the power consumption of a mapped application, *Average Weighted Manhattan Distance (AWMD)* is the sum product of *MD* and corresponding weight of communicating nodes, averaged by the total communication volume of the application:

$$AWMD_{map(A_p)} = \frac{\sum_{\forall e_{i,j} \in E} w_{i,j} \times MD(nt_i, nt_j)}{\sum w_{i,j}} \quad (2)$$

In other words, the *AWMD* of a mapped application determines the number of hops that each bit of the application data traverses in the network. The *AWMD* value in Figure 3 (a), for instance, is 63/35 = 1.8, which means each bit will dissipate 1.8 times of energy unit despite the networking overheads.

### 3.2.2 Mapped Region Dispersion

The network performance is highly correlated to the network congestion [13], [14]. Congestion increases the network latency dramatically [15] and also increases the network dynamic power consumption considerably [16]. This is why there are plenty of works aiming to diminish the network congestion in different aspects, like routing [16], [17] .

Two types of congestion can be defined from the perspective of dynamic application mapping: external and internal congestions. External congestion occurs when a network channel is contented

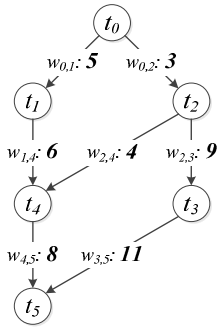


**Figure 3. An application with 6 tasks and 7 edges.**

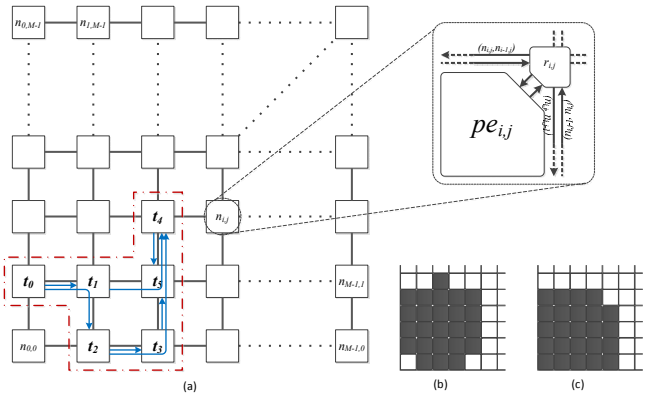


**Figure 2. (a) NoC-based platform, with an application mapped onto it (the highlighted region.). (b) An area of 24 nodes with minimum *MRD* value; and (c) the same number of nodes with the least *NMRD* value.**

by the packets of different applications; while the internal congestion is related to the packets of the same application. Hence, the internal congestion probability is related to the utilized mapping algorithm [4], and thus it is out of this paper scope.

As aforementioned, to decrease the external congestion probability, the mapped area of an application should be as convex as possible and minimally fragmented. Several works, e.g. [3], [4], considered the average pairwise MD of the allocated nodes as a metric to assess the *Mapped Region Dispersion*:

$$MRD_{map(A_p)} = \frac{\sum_{t_i,t_j \in T} MD(nt_i, nt_j)}{\binom{|T|}{2}} \quad (3)$$

Distant node allocation will increase the *MRD* for the obtained mapping; i.e. more external congestion probability. The increased congestion probability is originated from communication of tasks of different applications that are mapped among each other.

The area with the smallest *MRD* is almost circular [18]. Regarding the mesh topology of the network, however, a circular region will generate irregularity in remaining available nodes and more area fragmentation in long term. This is shown in Figure 3 (b) for allocation of 24 nodes with the least *MRD* value. On the other hand, a rectangular allocation forms regular regions, decreases applications overlap and thus isolates their communications. Thus, the best mapped area would be square (shown in Figure 3 (c)) as it is the rectangle with the smallest *MRD*. It can be shown that the *MRD* of a square with $|T|$ nodes will be:

矩形分配的原因

$$MRD_{SQ(|T|)} = \frac{2 \times \sqrt{|T|}}{3} \quad (4)$$

Thus the *Normalized MRD* metric is defined which assesses the squareness of the mapped region independent of the size of the application:

$$NMRD_{map(A_p)} = 1 + \frac{\left| MRD_{map(Ap)} - MRD_{SQ(|T|)} \right|}{MRD_{SQ(|T|)}} \quad (5)$$

The *NMRD* value of 1 means a squared area. *NMRD* increases as the mapped area is getting more fragmented and less similar to a square shape. For example, the *NMRD* of the dispersed mapped area in Figure 1(b) is 1.86 while it is 1.01 in the rectangle region of Figure 1(c). Moreover, as the *MRD* of the first area is almost twice that of the second one (4.3 vs. 2.3); the *AWMD* will get almost doubled using the same mapping strategy. Thus, dispersed allocation not only increases the congestion probability but also results in more power dissipation of the network. This shows the importance of this work in finding the optimum area for an application.

## 4. Smart First Node Selection

Based on the motivational examples and analysis presented in previous sections, the optimum area for an application mapping is

a (i) contiguous and (ii) square region of available nodes. CoNA [4] attempts to form a contiguous area around its *first node* while favoring square shapes. Given that, appropriate selection of the mapping *first node* is the important step towards the contiguity of the allocated area. Accordingly, the appropriate *first node* is the one with the required number of available nodes in a square shape around it while the minimum fragmentation of remaining nodes occurs after the mapping.

In this section, we first define our approximate model which estimates the number of available nodes in a square shape around a given node. Then the proposed model is utilized in the adapted hill climbing search hubristic to find the appropriate *first node* in an agile and smart manner.

## 4.1 Square Factor

The *square factor* of a given node, $SF(n_{i,j})$, is the estimated number of contiguous, almost square-shaped, available nodes around that node. Hence, the appropriate *first node* for mapping of an application would be the node with the *SF* equal to the application size.

Each running (already mapped) application in the system is modeled as a rectangle defined by its *left-down* and *right-up* corner nodes. Regarding the modeled rectangle of a running application, there might be (1) some nodes within the rectangle which do not belong to the application and/or (2) some nodes of application which are excluded from the rectangle. The rectangle of each application is modeled such that minimizing the number of these two types of nodes. The rectangle models of running applications 1 to 3 of Figure 1 are shown in Figure 4 (a). For instance, the rectangle of the application 1 excludes one node ($n_{2,2}$) of the application and the rectangle of the application 3 includes one node ($n_{3,1}$) which is not part of the application. However, they are the best fit rectangles according to the included and excluded nodes.

To calculate the $SF(n_{i,j})$, we first find the largest square centered on $n_{i,j}$, $SQ_{max} = (n_{i,j}, r_{max})$, where it fits within the mesh limits and has no conflict with other rectangles (running applications) of the system. This is shown in Figure 4 (b) for the node $n_{7,1}$ which is the *first node* of the application 4. In addition to the $SQ_{max}$ area, there might be also some more nodes beyond the square borders not belonging to system rectangles, as marked with asterisk in Figure 4(b). These nodes are counted in order to prevent available nodes from being isolated while keeping the mapped area close to the square shape. The square factor of a given node $n_{i,j}$ is calculated by summing up the area of the $SQ_{max}$ with the available nodes beyond the square borders. For instance, the $SF(n_{7,1})$ will be the square area, 9, summed up with marked nodes, 5, which is 14.

During the *SF* calculation of a node, the algorithm also calculates a direction, called open direction (*openDir*), which indicates one of the eight neighbors of the node estimated to have a larger *SF*. The *openDir* is towards that side of $SQ_{max}$ where there is the maximum number of nodes beyond it. In the example shown in Figure 4 (b), there are 3 asterisked nodes beyond the up side of the $SQ_{max}$ versus 1 in the left side (corners are not counted in). Thus the *openDir* would be upward, meaning the upper neighbor is probably holding a larger *square factor*. When there is more than one side with the maximum number of nodes beyond them, the *openDir* is then towards their vector addition. An openDir of value zero means no specific direction is predicted to result in a larger *square factor*. The climber function of the hill climbing heuristic uses the computed open direction to provide smartness in taking the steps.

Note that there can be available nodes inside an application rectangle; e.g. $n_{3,1}$ is inside the rectangle of the application 3. In such cases the *square factor* is evaluated by counting the available neighbors of the node, and the open direction is toward the side exiting from the rectangle. For instance, $SF(n_{3,1}) = 3$ because two of the node's neighbors are available ($n_{2,0}$ and $n_{2,1}$); the open direction is leftward to exit from the rectangle of the application. Moreover, the system *CM* is always assumed as a rectangle of one node.

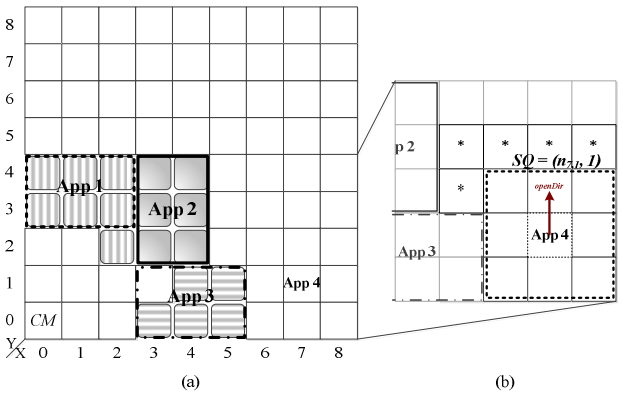


**Figure 4. (a) Applications are modeled as rectangles in our approach. (b) *Square factor* calculation of the node $n_{7,1}$ before application 4 being mapped.**

As the algorithm for the *SF* calculation explores the distance between the given node and all rectangles (running applications, *APPS*) of the system to find the $SQ_{max}$, the *SF* calculation has a linear time complexity of O (|*APPS*|).

## 4.2 SHiC: Smart Hill Climbing

To find the appropriate *first node*, one can calculate the *SF*, $O(|APPS|)$, for all $M \times M$ nodes of the system, and select the one with the best *SF* regarding the application size (exhaustive search). However, this will take $O(M^2|APPS|)$ time which is not a tolerable complexity for the future many-core systems with hundreds of nodes. Instead, *SHiC* starts from a randomly selected node and walks smartly through the network nodes by means of the defined open direction to reach the optimum node. This significantly reduces the amount of traversed nodes, resulting in an agile mapping algorithm. The pseudo code of our proposed algorithm is shown in Figure 5.

The original hill climbing heuristic starts from a node and moves in the direction of the increasing value (of the *SF*) to the uphill (max. *SF*) [19]. However, the appropriate *first node* is not the one on the uphill but the one in a specific height; i.e. the node with the *SF* equal to the application size. Hence, *SHiC* moves in the direction of the optimum *SF* instead of the maximum one. *SHiC* looks for the node with the optimum *SF* according to a preference

**Inputs:** Size (|T|) of the requested application $A_p$, current set of running applications APPS.

**Output:** chosen first node: $n_{fn}$.

```
(1)  repeat 2+√|APPS| times
(2)    n_cur ← choose a random available node;
(3)    iter ← 0 ;
(4)    yawAngle ← 0 ;
(5)    while (iter < (M/2) AND SF(n_cur) ≠ |T|)
(6)      if openDir(n_cur) = (0,0) then
(7)        moveDir ← a random direction;
(8)      else
(9)        moveDir ← (SF(n_cur) < |T| ) ? openDir (n_cur): -openDir(n_cur);
(10)       divert moveDir by (yawAngle);
(11)     end if
(12)     n_next ← move in moveDir;
(13)     if n_next is available AND preferred to n_cur then
(14)       n_cur ← n_next ;
(15)       yawAngle ← 0 ;
(16)     else
(17)       yawAngle ← a random in opposite side ;
(18)     end if
(19)   end while
(20)   if n_cur is preferred to n_fn then
(21)     n_fn ← n_cur ;
(22)   end if
(23) end repeat
```

**Figure 5. Smart Hill Climbing, *SHiC*, pseudocode.**

function. Standing on a node $n_{cur}$, a step to the next node $n_{next}$ is called to be a preferred step when:

$$(SF(n_{cur}) < |T| \text{ AND } SF(n_{next}) > SF(n_{cur})) \text{ OR} \quad (6)$$
$$(SF(n_{next}) \geq |T| \text{ AND } SF(n_{next}) < SF(n_{cur}))$$

By means of the defined preference function, *SHiC* first looks for the node with the smallest *SF* value which is larger than or equal to the application size. Otherwise, the node with the largest *SF* value is preferred. Note that, when there are two nodes with equal *SF*, the one closer to mesh corner is preferred to decrease the incurred defragmentation of remaining nodes.

The original hill climbing heuristic examines all possible neighbors to find the best choice, which will increase the execution time significantly. As an alternative, stochastic (first-choice) hill climbing generates random moves until one is generated that is preferred to the current one [19]. Moreover, *SHiC* uses the calculated *openDir* to impart smartness to the steps it takes (lines 6 to 11). Standing on the current node, the climber first moves according to the *openDir*. If the taken step is not preferred (line 17), the climber yaws and chooses a random direction on the opposite side of the yaw angle for the next move. The climber takes M/2 steps (line 5) as it is the required number of steps to start from a node in the mesh corner and reach the center node of the mesh.

On the other hand, the hill climbing heuristic might get stuck on local optimums. For example, the climber might stuck to the node $n_{1,1}$ of the Figure 1, while looking for a node with the *SF* equal to 12. As a solution, we use the random restart approach in which the proposed stochastic hill climbing approach is executed several times starting from different randomly chosen nodes [19]. The algorithm is repeated ($2+\sqrt{|APPS|}$) times (line 1) as the number of the fragmented regions in the system is related to the number of running applications.

Finally, the best found node is passed to the mapping algorithm as the *first node*. The *SHiC* outer loop is executed $O(\sqrt{|APPS|})$ times, the *while* loop is executed O($M$) times and $SF(n_{next})$ is calculated in each iteration in $O(|APPS|)$. Thus, the time complexity of the *SHiC* will be $O(M\times|APPS|^{3/2})$. This is significantly faster than the exhaustive search, and gets equal when $|APPS|$ is equal to the mesh size; i.e. one application per node which is an impractical case.

# 5. RESULTS AND ANALYSIS

In this section, we assess the impact of the *SHiC* method on improving the mapping results. Several set of applications with 4 to 35 tasks are generated using TGG [20] where the communication volumes ($w_{i,j}$) are randomly distributed between 2 to 16 flits of data. Experiments are performed on our in-house cycle-accurate SystemC many-core platform which utilizes a pruned version of Noxim [21], as its communication architecture. Different mapping and *first node* selection methods are evaluated over the network size varying from 8×8 to 20×20 nodes. These sizes are according to the current industry trends [22], [23] as well as the future many-core systems.

A random sequence of applications is entered into the scheduler FIFO according to the desired rate, λ. The sequence is kept fixed in all experiments for the sake of fair comparison. Applications are scheduled based on First Come First Serve (FCFS) policy and the maximum possible scheduling rate is called $\lambda_{full}$. An allocation request for the scheduled application is sent to the *CM* of the platform residing in the node $n_{0,0}$. The *first node* is selected using our *SHiC* method as well as other approaches. Then the application is mapped according to the desired mapping algorithm and tasks are allocated to the system nodes regarding the mapping result. Nodes emulate the behavior of their allocated task and inform the *CM* upon the task termination. Hence, the *CM* is kept updated about the status of the nodes at run-time without using any monitoring facilities. In order to have a holistic view of the results and enable real case comparisons, each set of experiments are performed over 10 million cycles where hundreds of applications enter and leave the system.

**Table 1. Extracted Results for Different Methods**

| *Mapping / first node* | $L_{avg}$ | *Ext. Cong.* | *NMRD* | *AWMD* |
|---|---|---|---|---|
| *CoNA / INC* | 42 | 7.71 | 1.26 | 1.29 |
| *CoNA / NN* | 43 | 7.65 | 1.20 | 1.23 |
| *CoNA / CoNA* | 41 | 5.63 | 1.09 | 1.06 |
| *CoNA / SHiC* | 40 | 3.96 | 1.00 | 1.00 |
| *CoNA / Exh. SF* | 39 | 3.55 | 0.97 | 0.96 |
| *INC / INC* | 52 | 11.51 | 1.27 | 1.49 |
| *INC / SHiC* | 42 | 4.02 | 1.02 | 1.04 |

## 5.1 Square Factor Accuracy and SHiC Success

In our first study, we assess the accuracy of the *square factor* and *SHiC* success in finding the optimum node. We run several experiments while different combinations of *first node* selection approaches and mapping algorithms are set. In order to assess the *square factor* accuracy, in one of the experiments we perform exhaustive search to select the node with the optimum *SF* as the *first node*. The network size is set to 16×16 and applications enter the system with $0.8\lambda_{full}$ rate. Table I shows the average latency of the network ($L_{avg}$), the percentage of packets delivered by external congestion (*Ext. Cong.*), and normalized values of *NMRD* and *AWMD* metrics for different experiments.

As can be seen, the best results belong to exhaustive search on the *SF* values. This shows the accuracy of the proposed model. Results demonstrate that *SHiC* has been successful in finding the optimum node, as the dispersion (*NMRD*) and power (*AWMD*) metrics show only 4% of increase. We remind the significant effect of area dispersion on the external congestion and network latency. This highlights the importance of the contiguous and convex mapping towards performance improvement of the system. Moreover, *SHiC* approach significantly enhances the performance of the system under the same utilized mapping algorithm; e.g. 50% reduction in the power consumption of *INC* mapping. This proves the effectiveness of the *first node* selection method despite the utilized mapping algorithm. Note that, the extracted power values follow the same trend as *AWMD*; thus omitted for brevity.

When the λ increases towards fully utilized state ($\lambda_{full}$), it is less possible to find a well-shaped region with required number of nodes. As the second evaluation, we study the λ effect on the system performance. Figure 6 shows *AWMD* and *NMRD* values of *CoNA*/*SHiC* case as well as their ratio to the *CoNA*/*NN* case, indicated by R(*AWMD*) and R(*NMRD*). Other approaches follow almost the same trend. As can be seen both the dispersion and power dissipation of applications increase versus increase of λ, while *SHiC* keeps its preeminence over other approaches and outperforms by 10 to 30 percent.

## 5.2 Scalability Evaluation

The *SHiC* performance in different network sizes is evaluated as

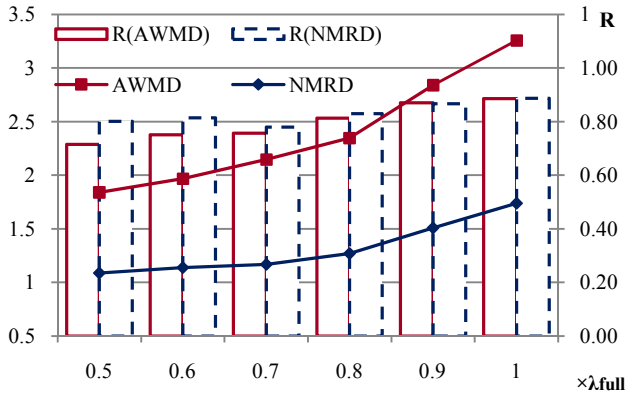


**Figure 6. *AWMD* and *NMRD* values for *SHiC* approach in different λ values, along with their ratio to *NN*, valued in the right axis.**

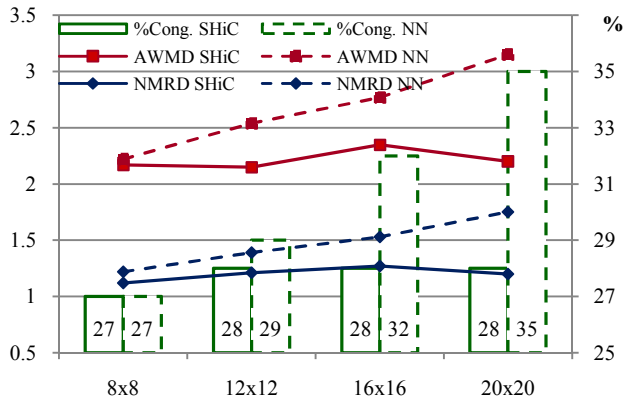


**Figure 7. *AWMD*, *NMRD* and % of congested packets in *CoNA/SHiC* and *NN/NN* cases over different network sizes. The congestion values are according to the right axis.**

our third study. Different experiments are done over various network sizes while the λ is kept $0.8\lambda_{full}$. Figure 7 shows *AWMD*, *NMRD* and percentage of congestion in different configurations for both the *CoNA*/*SHiC* and the *NN*/*NN* cases. As can be seen *SHiC* scales well as the network size increases and keeps the system performance at the same level. While the system performance in other approaches drops. Thus, the *SHiC* approach is suitable for both the current network sizes and future many-core systems.

# 6. CONCLUSION

In this paper, we proposed an agile scheme, *SHiC*, to find the optimum *first node* for run-time application mapping of many-core systems. The adapted stochastic hill climbing algorithm finds a node that has the required number of available nodes around it. This algorithm utilized an approximate model which quickly estimates the available area around a given node. The provided open direction aided the climbing algorithm to reach the optimum node faster by taking smart steps.

Simulation results over different network sizes and system utilizations showed significant improvement influenced by *SHiC* in different network parameters such as congestion and power dissipation. More precisely, results emphasized the significant impact of convex mapping on congestion reduction of the network.

As future work a distributed version of the proposed work collaborating with the utilized mapping algorithm is planned. Moreover, a more precise model of mapped applications is desired in order to increase the achieved contiguity of the mapped applications.

# 7. REFERENCES


[1] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Design Automation Conference, 2001. Proceedings*, 2001, pp. 684– 689.

[2] E. Carvalho, N. Calazans, and F. Moraes, "Heuristics for Dynamic Task Mapping in NoC-based Heterogeneous MPSoCs," in *18th IEEE/IFIP International Workshop on Rapid System Prototyping, 2007. RSP 2007*, 2007, pp. 34–40.

[3] C.-L. Chou, U. Y. Ogras, and R. Marculescu, "Energy- and Performance-Aware Incremental Mapping for Networks on Chip With Multiple Voltage Levels," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 10, pp. 1866–1879, Oct. 2008.

[4] M. Fattah, M. Ramirez, M. Daneshtalab, P. Liljeberg, and J. Plosila, "CoNA: Dynamic application mapping for congestion reduction in many-core systems," in *2012 IEEE 30th International Conference on Computer Design (ICCD)*, 2012, pp. 364 –370.

[5] D. P. Dobkin, H. Edelsbrunner, and M. H. Overmars, "Searching for empty convex polygons," in *Proceedings of the fourth annual symposium on Computational geometry*, New York, NY, USA, 1988, pp. 224–228.

[6] S. Fortunato, "Community detection in graphs," *arXiv:0906.0612*, Jun. 2009.

[7] E. L. d. S. Carvalho, N. L. . Calazans, and F. G. Moraes, "Dynamic Task Mapping for MPSoCs," *IEEE Design & Test of Computers*, vol. 27, no. 5, pp. 26–35, Oct. 2010.

[8] A. Weichslgartner, S. Wildermann, and J. Teich, "Dynamic decentralized mapping of tree-structured applications on NoC architectures," in *2011 Fifth IEEE/ACM International Symposium on Networks on Chip (NoCS)*, 2011, pp. 201–208.

[9] S. Kobbe, et al., "DistRM: distributed resource management for on-chip many-core systems," in *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, New York, NY, USA, 2011, pp. 119–128.

[10] M. Hosseinabady and J. L. Nunez-Yanez, "Run-time stochastic task mapping on a large scale network-on-chip with dynamically reconfigurable tiles," *IET Computers Digital Techniques*, vol. 6, no. 1, pp. 1 –11, Jan. 2012.

[11] M. Asadinia, M. Modarressi, A. Tavakkol, and H. Sarbazi-Azad, "Supporting non-contiguous processor allocation in mesh-based CMPs using virtual point-to-point links," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, 2011, pp. 1–6.

[12] T. T. Ye, G. D. Micheli, and L. Benini, "Analysis of power consumption on switch fabrics in network routers," in *Proceedings of the 39th annual Design Automation Conference*, New York, NY, USA, 2002, pp. 524–529.

[13] C.-L. Chou and R. Marculescu, "Contention-aware application mapping for Network-on-Chip communication architectures," in *IEEE International Conference on Computer Design, 2008. ICCD 2008*, 2008, pp. 164–169.

[14] C.-Q. Yang and A. V. Reddy, "A taxonomy for congestion control algorithms in packet switching networks," *IEEE Network*, vol. 9, no. 4, pp. 34–45, Aug. 1995.

[15] J. W. Brand, C. Ciordas, K. Goossens, and T. Basten, "Congestion-Controlled Best-Effort Communication for Networks-on-Chip," in *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07*, 2007, pp. 1–6.

[16] S. Ma, N. Enright Jerger, and Z. Wang, "DBAR: an efficient routing algorithm to support multiple concurrent applications in networks-on-chip," in *Proceedings of the 38th annual international symposium on Computer architecture*, New York, NY, USA, 2011, pp. 413–424.

[17] M. Ebrahimi, et al., "HARAQ: Congestion-Aware Learning Model for Highly Adaptive Routing Algorithm in On-Chip Networks," in *2012 Sixth IEEE/ACM International Symposium on Networks on Chip (NoCS)*, 2012, pp. 19 –26.

[18] C. M. Bender, M. A. Bender, E. D. Demaine, and S. P. Fekete, "What is the optimal shape of a city?," *J. Phys. A: Math. Gen.*, vol. 37, no. 1, p. 147, Jan. 2004.

[19] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2009.

[20] *Task graph generator (TGG)*. [Online]. Available at: http://sourceforge.net/projects/taskgraphgen/.

[21] *Noxim: the NoC Simulator*. [Online]. Available at: http://noxim.sourceforge.net/.

[22] J. Howard, et al., "A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, 2010, pp. 108 –109.

[23] Tilera Corporation, "Tile-GX Processor Family," 2011.