MapPro: Proactive Runtime Mapping for Dynamic Workloads by Quantifying Ripple Effect of Applications on Networks-on-Chip

Mohammad-Hashem Haghbayan[1], Anil Kanduri[1], Amir-Mohammad Rahmani[1,2], Pasi Liljeberg[1], Axel Jantsch[3], and Hannu Tenhunen[1,2]
[1]Department of Information Technology, University of Turku, Finland
[2]Department of Industrial and Medical Electronics, KTH Royal Institute of Technology, Sweden
[3]TU Wien, Austria
{mohhag, spakan, amirah, pakrli}@utu.fi, hannu@kth.se, axel.jantsch@tuwien.ac.at

ABSTRACT

Increasing dynamic workloads running on NoC-based many-core systems necessitates efficient runtime mapping strategies. With an unpredictable nature of application profiles, selecting a rational region to map an incoming application is an NP-hard problem in view of minimizing congestion and maximizing performance. In this paper, we propose a proactive region selection strategy which prioritizes nodes that offer lower congestion and dispersion. Our proposed strategy, MapPro, quantitatively represents the propagated impact of spatial availability and dispersion on the network with every new mapped application. This allows us to identify a suitable region to accommodate an incoming application that results in minimal congestion and dispersion. We cluster the network into squares of different radii to suit applications of different sizes and proactively select a suitable square for a new application, eliminating the overhead caused with typical reactive mapping approaches. We evaluated our proposed strategy over different traffic patterns and observed gains of up to 41% in energy efficiency, 28% in congestion and 21% dispersion when compared to the state-of-the-art region selection methods.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design − Real-time systems and embedded systems

General Terms

Algorithms, Performance, Management, Design

Keywords

On-chip many-core systems, Application mapping, Task allocation, Proactive Runtime Mapping

1. INTRODUCTION

Networks-on-chip (NoC) based many-core systems have accelerated the performance of computationally intensive applications by providing a better communication infrastructure [8]. Applications running on a system are modelled as task graphs where tasks are individual computational blocks that communicate with each other [17]. Application mapping is the phase where a tile for a task is chosen in order to maximize the cores' and network's performance while minimizing latency and power consumption. Given the un-predictable nature and sequence of incoming applications [4] [5], mapping has to be performed dynamically rather than at design time [7] [2] [10]. With a wide range of applications entering and leaving such a system, runtime application mapping policies become crucial factor in determining the chip's performance, power consumption and reliability [6]. We consider run-time mapping as one of the first steps in servicing an incoming application as opposed to reactive steps like task migration. Mapping an entire application consisting of several communicating tasks, satisfying power and performance constraints is a complex process. Assuming that there can be other applications running in parallel on the chip, mapping a new application adds to the complexity and consumes more execution time, degrading the expected performance from a parallel system. Finding a preferable region to map an incoming application with least possible overhead is thus important to ensure high performance of the chip.

Runtime mapping is split into two phases viz., finding a suitable region to map an application [7], followed by mapping tasks of the application in the selected region [11]. Optimal region for mapping an application is the one that results in lower congestion, dispersion, power consumption and higher performance. An optimal region for an application can be found starting with an optimal node in the network, referred to as the $first\ node$, around which an application can be mapped. First node should have unoccupied nodes around it in a contiguous manner, forming a near convex shape which will be ideal for lower congestion and dispersion. First node selection effects internal and external congestion, dispersion and fragmentation of NoC-based systems, creating a significant impact on performance [12]. For a network of size $n \times n$, finding the optimal first node and a region in the mesh for an incoming application is of polynomial time complexity, of the order O(n^3) [9]. Adding the above mentioned constraints on congestion and dispersion to the mapping issue makes it an NP-hard problem [15]. Although first node selection is an important phase that determines performance of the system, it is not desirable to exhaustively search the network for an optimal first node. A near exhaustive search might offer an optimal first node, however the time spent in finding the first node nullifies any performance gains that come with it. Keeping in view the growing computational demands and scalability of networks, it is of critical importance to design fast and adaptive first node finding strategies. Significant

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
NOCS '15 Vancouver, BC Canada
Copyright 2015 ACM 978-1-4503-3396-2/15/09$15.00 ...$15.00.
http://dx.doi.org/10.1145/2786572.2786589.

work has been done so far on runtime mapping [11] [13] [7], yet not many existing techniques have optimized the first node selection. Existing first node selection strategies either do not consider the combination of spatial availability, congestion, dispersion [6] [7] [16] or involve expensive computations to search the network and find an optimal region [12].

In this paper, we propose a proactive first node selection strategy that provides a near convex (square shaped) region with minimal congestion and dispersion. We cluster the network into square shaped regions of different radii to fit applications of different sizes, and quantify spatial availability and probability for internal congestion of each square. We minimize the search to only those squares that can fit the size of a newly arrived application. We consider the impact of currently mapped applications on remaining un-occupied nodes by updating their availability and congestion metrics. This helps our proactive strategy of finding a preferable region for the next incoming application of any size with a single look-up for the square that can accommodate it. Our idea is to map the application in a suitable region up on its arrival itself, limiting the need for an expensive task migration later. The contributions from our work are:

- Quantification of spatial availability, internal congestion and dispersion into a unified metric
- Modeling the of ripple effect of a newly mapped application on remaining un-occupied nodes.
- Proactive first node selection for a generic NoC mesh running dynamic workloads.

The rest of the paper is organized as follows. We formulate the problem of first node selection and motivate its importance in Section 2. Section 3 formally introduces the preliminaries of our first node selection method and later details the proposed algorithm. The evaluation platform and results in comparison to state-of-the-art first node selection methods are presented in Section 4. Finally, Section 5 concludes the paper and discusses possible future works.

2. MOTIVATION

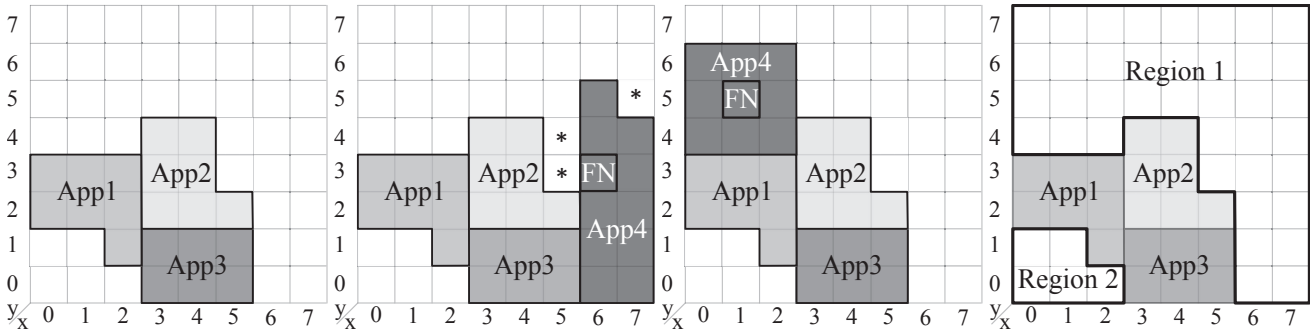
First node selection has a significant effect on internal and external congestion, and dispersion of applications running, which subsequently reflects in the chip's performance [12]. It is necessary to find an optimal first node that satisfies congestion and dispersion metrics, with minimal execution time. First node selection not only effects the current application, but also on future incoming applications. The effect of first node selection on current and future incoming applications is explained through a motivational example. Figure 1(a) shows the status of a network with three applications App1, App2 and App3 of sizes 6, 8 and 10 running on it. With the arrival of a new application App4 with 9 tasks, a suitable region has to be found such that all tasks of App4 are mapped with minimal congestion and dispersion. A naive first node selection allocates a random node located at (6,3) to App4 and subsequent mapping is shown in Figure 1(b). All the tasks are spread around the first node, which does not lead to a convex shape. Although it is near convex, it leaves the nodes located at (5,3), (5,4) and (7,5) fragmented. These three nodes do not form a convex shape are less likely to be part of future incoming applications. They either stay un-occupied resulting in poor resource utilization, or become part of same application resulting in a higher communication penalty. In addition, tasks of any application mapped onto these nodes in the future would contend with packets of App4 leading to external congestion. In contrast, a congestion, dispersion and spatial availability aware first node selection is shown in Figure 1(c). The node located at (1,4) is selected as first node and App4 is mapped around it in a perforate convex shape. This mapping of App4 ensures that all of its tasks are at minimal communication distance, and do not conflict with any other running applications. It causes no fragmentation leaving convex shapes for future incoming applications which would also minimize dispersion.

2.1 Problem Formulation

The foremost requirement of a first node is the availability of enough number of free nodes in its proximity on to which incoming application's tasks can be mapped. The next requirement is to have lower internal congestion - the communication penalty that arises from communicating tasks of an application that are mapped far apart. Contiguous nodes are the nodes that are adjacent to each other which ensure a minimal hop distance among communicating tasks of an application. Mapping an application onto contiguous nodes can minimize internal congestion. The other metric to be considered during first node selection is dispersion - a case where un-occupied nodes are far apart and spread out across the network into concave and irregular geometric shapes, leading to poor resource utilization. This will also result in external congestion - contention between packets belonging to different applications. A square shaped region has a minimal radius among regular geometric shaped regions, making it a compact fit for an application. Application mapped onto a square shaped (or almost square shaped) regions is less likely to interfere with other applications, minimizing both the probability for dispersion and external congestion. In view of these metrics, an optimal first node for a new application thus has to be:

- Spatially available - enough number of free nodes around it to map the application.
- Contiguous - free nodes that are contiguous to minimize internal congestion.
- Near Convex - free nodes forming a near convex geometrical shape to minimize dispersion and external congestion.

Finding the appropriate first node is the key factor to find the optimal region. However, all the aforementioned metrics have to be quantified to prioritize one node over the other. In addition, exhaustively searching for an optimal node has to be avoided. A state-of-the-art first node selection through a hill climbing approach is proposed in [12]. Three applications App1, App2 and App3 are currently running and App4 is about to enter the system, as shown in Figure 1(a). A hill climbing approach for first node selection proposed in [12] defines an *open direction*, the direction in which probability of finding free nodes is higher. Upon arrival of a new application, it randomly chooses a free node, identifies the *open direction* for the chosen node and expands the search space towards it iteratively. The search stops when the number of free nodes around a selected node is equal to the new application's size. It tries to walk through the mesh with the intention to find the smallest possible, contiguous region that can fit the new application. On an average, complexity of this method is of the order O($\sqrt{n}$), while on a worst case, it is O(n^2), where $n \times n$ is the network size. Although this approach results in lower dispersion and higher contiguity, its random nature and scenario of other applications running on the mesh affects its time complexity. While certain combination of occupied nodes on network favor hill climbing, some other combination might result in a worst case time complexity. Most importantly, with availability of contiguous free nodes being the only criteria, this approach limits itself to finding only one suitable



(a) 3 Applications Mapped (b) Naive region selection for App4 (c) Optimal region selection for App4 (d) Limitation of Hill Climbing

Figure 1: Impact of first node selection

first node, while there might exist a much better alternative. Figure 1(d) shows the possible open direction traversal of hill climbing approach in two Region-1 and Region-2. Randomly choosing a node in one region will ignore other possible optimal nodes. In case the random nature results in choosing a node from Region-2, hill climbing gets confined or stuck at this region and is less likely to find an optimal node. As mentioned in previous section, we cluster the network into squares of different radii and assign a combinatorial value of availability and congestion to each square. Figure 1(c) shows the mapping of App4 in square centered at (1,5), although there are other possible squares, we choose the best possible region for the new application so as to ensure contiguity with other applications. We have the bird-eye view of entire network, as opposed to hill climbing approach that allows us to make rational decisions for incoming applications. In the next section, we present our approach on quantifying these parameters and our proactive approach that eliminates first node finding overhead.

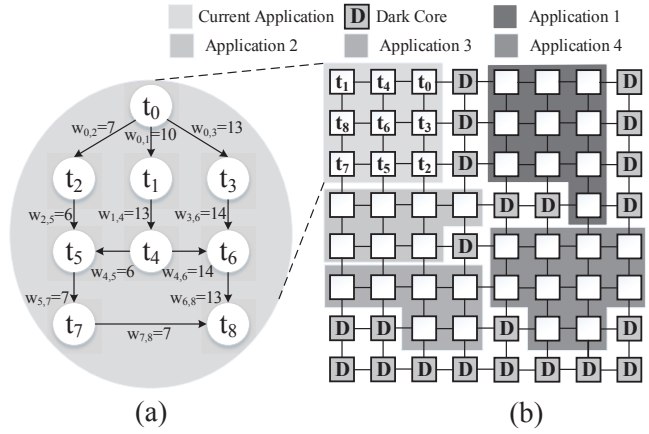


Figure 2: Mesh-Based platform with an application mapped onto it (the highlighted region.)

3. PROACTIVE FIRST NODE SELECTION

3.1 Preliminaries

Each application in the system is represented by a directed graph denoted as a task graph $Ap = TG(T, E)$. Each vertex $t_i \in T$ represents one task of the application, 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 a Gaussian Elimination application [3] which is extracted using TGG [1] is shown in Figure 2(a). The $w_{i,j}$ values, i.e. the amount of data transferred from t_i to t_j of edge $e_{i,j}$, is indicated on each edge. An architecture graph $AG(N, L)$ describes the communication infrastructure of the processing elements. Given its simplicity and popularity, we considered a 2D-mesh NoC (Figure 2(b)) with XY deterministic wormhole routing. Other network topologies can also use our first node selection strategy, however it might lead to inconsistent square regions towards the edges, as opposed to the center. The AG contains a set of nodes $n_{w,h} \in N$, connected together through unidirectional links $l_k \in L$. Each node is the combination of the Processing Element (PE) connected to the router. Mapping of an application onto the system is defined as a one-to-one function from the set of application tasks to the set of nodes:

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

Figure 2(b) illustrates a possible mapping of the application in Figure 2(a), onto the described platform.To attain a near convex shape, for simplicity and ease of application assignment, we cluster the network into squares of different radii, intuitively to fit applications of different sizes. A square region and its features are mentioned below.

$Square(S^r_{ij})$: A square S^r_{ij} is a square shaped collection of nodes, centered at node n_{ij} with r being its radius i.e., the distance from the center of the square to any of its edges. A square region can have both free and occupied nodes. The number of nodes in each square shape region in the network ranges in 1, 9, 25, ... , $(2i+1)^2$, for radii 0, 1, 2, 3.... $(\sqrt{M}-1)/2$. There can be different squares of same size, but with different centers. Figure 3 shows a squares centered at two different nodes, $n_{3,7}$ and $n_{6,3}$, with radius 1 and 2. The node $n_{3,7}$ is a center for the squares $S^1_{3,7}$ and $S^2_{3,7}$ and the $noden_{6,3}$ is a center for the squares $S^1_{6,3}$ and $S^2_{6,3}$. Note that each node is a center for squares with radius 1, 2, 3, ... up to $M/2$. All the squares of same size are grouped together into a square group, SG^r, where r is the radius of squares in the group. On the whole, total number of square groups in the mesh is expressed as:

$$groups = \sum_{i=1}^{(\sqrt{M}-1)/2} (2i+1)^2 = \frac{k(2k+3)^2 + 2k}{3} \quad (2)$$

Where $k = (\sqrt{M}-1)/2$. Searching for a square for an incoming application among $groups$ results in a time complexity of order

$O(M^{3/2})$, where $M \times M$ is the size of the network. For applications of every size within the mesh limits, there exists a square that has enough number of nodes to accommodate the application. For instance, an incoming application of size 7 can fit in a square of size 9 (radius 1). In general, an application of size $size$ can be fit in square with radius r such that $size \in ((r-2)^2, r^2]$. We find a perfect square even for applications that do not perfectly fit in a square, leaving few cores un-occupied. In such cases, we consider these un-occupied cores as available for mapping, but belonging to squares with a different first node than the current square and also with a different radius. With the arrival of a new application, a square suitable for the application size is chosen. However, there might be more than more than one square that can accommodate the new application. We resolve this by quantifying spatial availability and congestion of a square by assigning weighted parameter called *Vicinity Counter* to the center of a square.

Definition: *Vicinity counter, VC^r_{ij}, for a node located at (i,j) is the weighted sum of number of free nodes in the square centered at (i,j) with radius r. We assign the weights to a node $n_{i,j}$ as:*

$$Wn_{i,j} = \begin{cases} 1 & \text{if } n_{i,j} \text{ is unoccupied} \\ -1 & \text{if } n_{i,j} \text{ is occupied} \end{cases}$$

A free node has a weight 1 while an occupied node has a weight -1. Further, we consider the impact of occupied nodes in the square on congestion. A node that is occupied in the inner most proximity, close to the first node has more affect on internal congestion than the ones that are occupied, far from the first node. We quantify this by pegging the weight of an occupied node with its distance from the central node by assigning a higher penalty to occupied nodes closer to central node and relatively lower penalty to the ones that are far. The vicinity count for a node (i,j) is expressed as:

$$VC^r_{i,j} = \sum Wn_{i,j} \times (r - d + 1) \tag{3}$$

Where r is radius of square and d is the distance from occupied node to the center. The VC of a node (i,j) qualitatively represents the dispersion, internal congestion, and contiguity of the node (i,j) and thus of the selected square centered at (i,j). This makes it easier to choose a first node and thus the region for mapping an incoming application minimizing congestion.

$DistanceFactor(D_{ij})$: Distance Factor of a node located at (i,j) is the sum of distance of the node from all the other occupied nodes of the network. Distance Factor is expressed as:

$$D_{ij} = \sum_{x,y}^{X,Y} \mid (i-x) \mid + \mid (j-y) \mid \tag{4}$$

where D_{ij} is the Distance Factor of the node located at (i,j), (x,y)$\in$ (X,Y) are occupied nodes of the network. We use the Distance Factor to represent the contiguity of the chosen first node located (i,j) with respect to all the current occupied nodes. A higher D indicates that the chosen node is far from existing applications and is likely to violate contiguity metric. In contrast, a node with lower D value is nearer to existing applications and is more likely to be contiguous with them. For different first nodes with same VC value, we use the Distance Factor to prioritize nodes that result in a contiguous mapping.

Figure 4 shows the vicinity count values for two different scenarios of occupied nodes. In case of square centred at tile t_{37}, number of occupied nodes in the square are 2, with both the nodes being in the inner most square close to the first node. The VC for t_{37} = 25-(1*2 + 1*2) = 21. The other scenario presented is the square

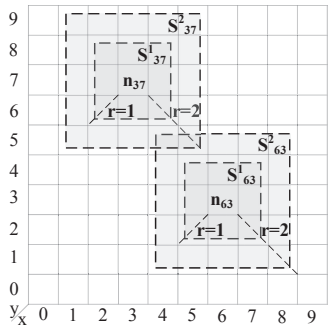


Figure 3: Example of the square regions

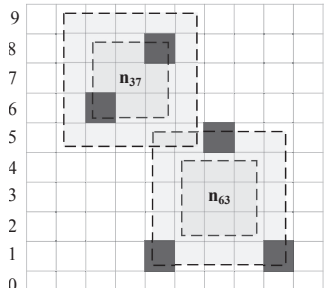


Figure 4: Quantifying probability of congestion

centred at node t_{63}, with 3 nodes occupied, all being in outer most square far from the central node. The VC for this square = 25-(1*1 + 1*1 + 1*1) = 22. Square centred at t_{63} has a better VC value and would be the preferred despite having more number of occupied nodes than that of square centred at t_{37}. This can be attributed to the fact that t_{37} has occupied nodes closer which would result in higher congestion and dispersion. The fact that tasks with higher communication volume will be mapped onto the central node and its neighbours, makes the impact of occupied nodes in inner squares more significant.

3.2 Ripple Effect of Mapping

When a new application has arrived and gets mapped onto the system, it effects its neighboring un-occupied nodes in terms of probability of congestion and dispersion. This propagates similar to the ripple effect of stone in water such that the impact decreases with distance from point of impact. We model this effect by updating the VC value for each node (thus squares) in the network once a new application is mapped. When a node gets occupied, other nodes surrounding it gets assigned a lower VC (as in Equation 3), subject to the distance from occupied node. When the system is initialized, all the nodes are assumed to be un-occupied. Figure 5 shows the Vicinity Counter values for square groups 1 to 4(SG-1 - SG-4) with radii 1 to 4 under initial conditions. Nodes on the edges have lesser neighbors reflecting in lower VC values and hence lower the chance of being prioritized. Nodes towards the center of the network tend to have higher VC values due to abundance of neighbors. Although for squares with lower radius VC value is almost even throughout the network, except for the edges. With increase in radius, resulting in squares that can fit applications of higher size, VC value increases towards the center of the network. Figure 5 shows the top view of the network which distinguishes be-

Algorithm 1 First Node Selection

Inputs: $appSize$: Size of the entering application, $newMapped$: Incoming application; $newReleased$: Released application;
Outputs: C_{fn}: The selected first node for the mapping;
Constants: M: Size of the network;
$groupLength$: Number of square groups equal to $[(\sqrt{M}-1)/2]$;
Global Variables: $VC[M][groupLength]$: Vicinity counter array; $maxVC[groupLength]$: Maximum vicinity counter in each group;

Body:

1: $C_{fn} \leftarrow maxVC_{[(\sqrt{appSize}-1)/2]}$
2: **if** $newMapped$ **then**
3: **for** each $C_{xy} \in newMapped$ **do**
4: $addCore(C_{xy})$;
5: **end for**
6: **end if**
7: **if** $newReleased$ **then**
8: **for** $eachC_{xy} \in newReleased$ **do**
9: $releaseCore(C_{xy})$;
10: **end for**
11: **end if**

Algorithm 2 Updating VC Values

Inputs: n_{xy}: Input core located in Row x and Column y;
Variables: VC: Vicinity counter for each node in a square; $maxVC$: Node with the maximum vicinity counter in square; D: Distance variable;
Constants: $maxRadius$: The maximum radius length of the square $((\sqrt{M}-1)/2)$; //M is the network size;
Outputs: Updating the global variables defined in Algorithm 1;

Body:

1: **for** each core n_{ij} located in Row i and Column j **do**
2: r' = $maximum(|i-x|, |j-y|)$;
3: $D_{ij} += r'$;
4: **for** r = 1 to $maxRadius$ **do**
5: **if** $r - r' \geq 0$ **then**
6: VC^r_{ij} -= r - r';
7: **if** $VC^r_{ij} > maxVC_r$ **then**
8: $maxVC_r \leftarrow VC^r_{ij}$;
9: **else**
10: **if** $VC^r_{ij} = maxVC_r$ and $D_{ij} < D_{maxVC_r}$ **then**
11: $maxVC_r \leftarrow VC^r_{ij}$;
12: **end if**
13: **end if**
14: **end if**
15: **end for**
16: **end for**

tween optimal and non-optimal regions for incoming application of any size. This approach makes it simpler to choose preferable first node by prioritizing nodes with higher VC. With every new application mapped onto the system, the VC values for all corresponding un-occupied nodes gets updated accordingly. Figure 6 shows the updated VC values after 3 applications App1, App2 and App3 being mapped for square groups of radius 1 to 4 (SG-1 - SG-4). Nodes in the proximity of mapped regions gets assigned to a lower VC value , while this impact lessens as we move away from occupied regions. Since SG-3 and SG-4 can fit applications of higher sizes compared to that of SG-1 and SG-2, the degradation of VC is much higher for SG-3 and SG-4. The network's status gets updated and once again is ready to choose preferable first nodes for next incoming applications of any size based on updated VC values. Figure 7 shows the updated VC values for square groups 1 to 4, after another application App4 enters the system. The updated values show a further deterioration in VC, making it easier to classify preferable regions for future incoming applications. The procedure for handling requests of incoming and exiting applications is shown in Algorithm 1. When an application of size $appSize$ enters the system, the first node that can fit the new application has to be selected. The node which centers the square that can fit $appSize$ with best possible VC value, $maxVC_{[(\sqrt{appSize}-1)/2]}$ is chosen (line 1). Application is then mapped onto the system around the selected first node and all the nodes onto which application is being mapped are marked as occupied (lines 2-5). If an application finishes its execution and leaves the system, all the nodes on which the exiting application ran are now marked as un-occupied. Every time an application enters or leaves the system, status of nodes gets changes. This reflects in changing VC values of all the remaining nodes, which is updated as shown in Algorithm 2. For every node, the effective distance from the node to the center of square(s) it belongs to is calculated (line 2). These values accumulated as a sum to compute distance factor for each node (line 3). The VC value for the node is updated based on the node status (Wn_{ij}) and its distance from center ($r - r\prime$) (line 5). When an application has entered the system, the VC value is computed as a subtraction of node status, while it is computed as addition in case an application has exited (line 5). The node with highest VC value in the group of squares with radius r and with lowest Distance Factor is chosen as the $maxVC_r$, which in turn will be the first node for any incoming application that fits in square with radius r (line 7).

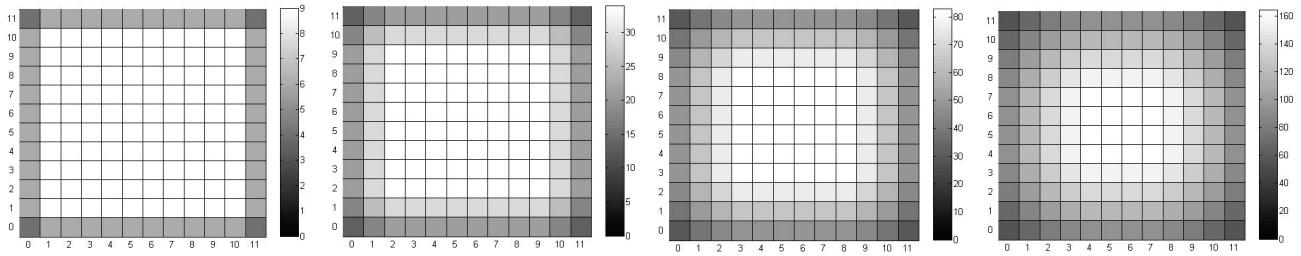
VC values get updated progressively node by node starting from a recently occupied first node. So, the complexity of the algorithm would be $O(appSize \times M)$. This latency consumed in updating VC values may affect a newly arriving applicationâĂŹs throughput, if it arrives while the VC values are still being updated. In case a new application arrives later, the first node can be calculated immediately as the VC values are already updated, proactively. In practice, the algorithm is much faster than the worst case and terminates in a constant time, subject to arrival of new applications. Therefore, the speedup of the algorithm depends on the probability of application arrival in time. If the application queue is full, calculating first node is of $O(appSize \times N)$ complexity, as VC values should be updated with exit of an application and arrival of a new application. When the application queue is empty (not full) and also in case of time gap between the arrival of two consecutive applications, the first node for the next application can be proactively calculated. For example if App1 arrives at t=0 and the next application App2 arrives at t=1ms, there would be 1ms time for the algorithm to calculate the first node for App2, and this time consumed will not reflect in App2's throughput.

4. EXPERIMENTAL RESULTS AND EVALUATION

To evaluate our proposed first node selection approach (MapPro), we compare it against state-of-the-art first node selection by hill climbing (SHiC) proposed in [12], along with other relevant first node selection strategies of Nearest neighbor (NN) [6] and Incremental selection (INC) [7]. In order to gain insight on impact of first node selection strategies, we employed the same contiguous mapping approach proposed in [11], CoNA, for all cases. The evaluation is consolidated to four NoC-based systems that use MapPro, SHiC, INC and NN as first node selection methods followed by CoNA for mapping around selected first node.

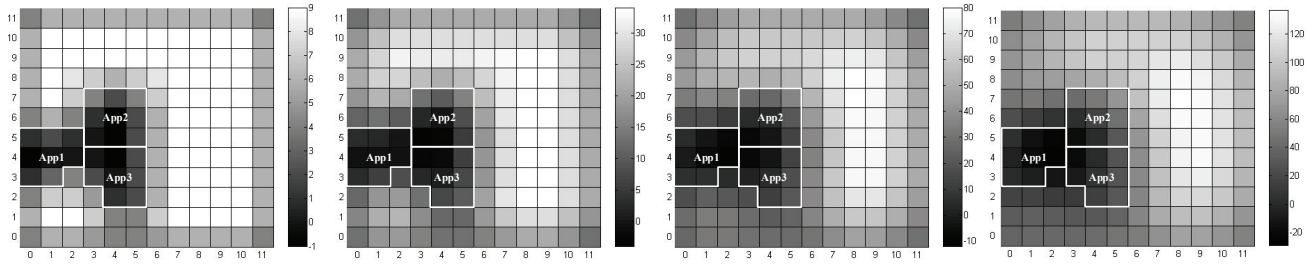
4.1 Evaluation Metrics

We evaluate all the first node selection strategies over different metrics that indicate network efficiency in terms of performance, energy efficiency, dispersion, internal and external congestion. The



(a) SG-1 (b) SG-2 (c) SG-3 (d) SG-4

Figure 5: Initial VC values for different Square Groups



(a) SG-1 (b) SG-2 (c) SG-3 (d) SG-4

Figure 6: VC values for different Square Groups with 3 applications mapped

Table 1: AMD, MRD and APL of first node selection strategies over 12×12 NoC

Method	AMD	MRD	Avg. Latency
MapPro	2.38	2.81	22.96
SHiC	3.08	3.39	23.33
INC	3.76	3.14	23.51
NN	3.71	3.96	23.59

Table 2: AMD, MRD and APL of first node selection strategies over 14×14 NoC

Method	AMD	MRD	Avg. Latency
MapPro	2.23	2.66	21.40
SHiC	2.31	2.76	21.48
INC	3.16	3.50	23.99
NN	2.46	2.81	21.73

Table 3: AMD, MRD and APL of first node selection strategies over 16×16 NoC

Method	AMD	MRD	Avg. Latency
MapPro	2.30	2.65	21.25
SHiC	2.74	3.13	22.50
INC	3.74	4.09	25.44
NN	2.98	3.41	23.26

Table 4: AMD, MRD and APL of first node selection strategies over 18×18 NoC

Method	AMD	MRD	Avg. Latency
MapPro	2.38	2.71	21.34
SHiC	2.53	2.81	22.26
INC	2.95	3.41	22.95
NN	2.52	2.79	21.81

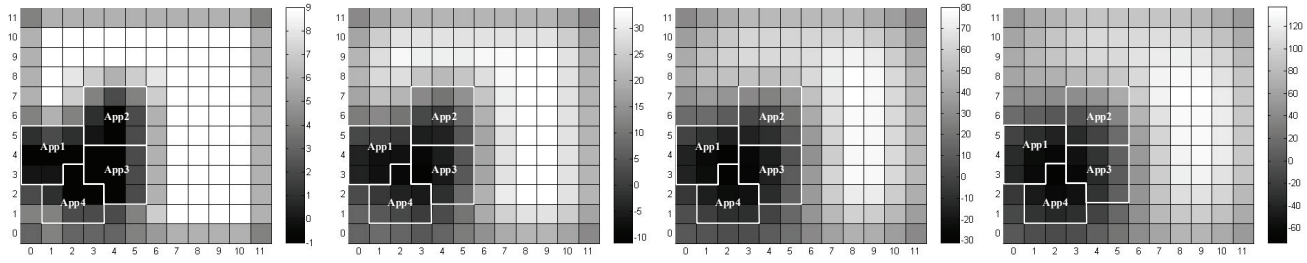
evaluation metrics used are listed below.

AMD and ***AWMD***: Manhattan distance (MD) is the number of hops to be traversed by a packet from its source node to destination node. Average Manhattan Distance (AMD) is the average of hops traversed (MD) by all the communicating nodes in a mapped application. A lower AMD indicates fewer number of hops traversed by packets indicting lower energy consumption. Average Weighted Manhattan Distance (AWMD) represents the volume of packets that traverse each hop as opposed to hop count alone. AWMD represents energy consumption of the network, as it is directly related to number of packets and number of hops each packet traverses.

APL: The time taken by a packet injected into the network from a source node till it is received at the destination node is traced as packet latency. Average of latencies of all the plackets in the network is reported as Average Packet Latency (APL). APL represents networks's performance in terms of total time consumed to route all packets.

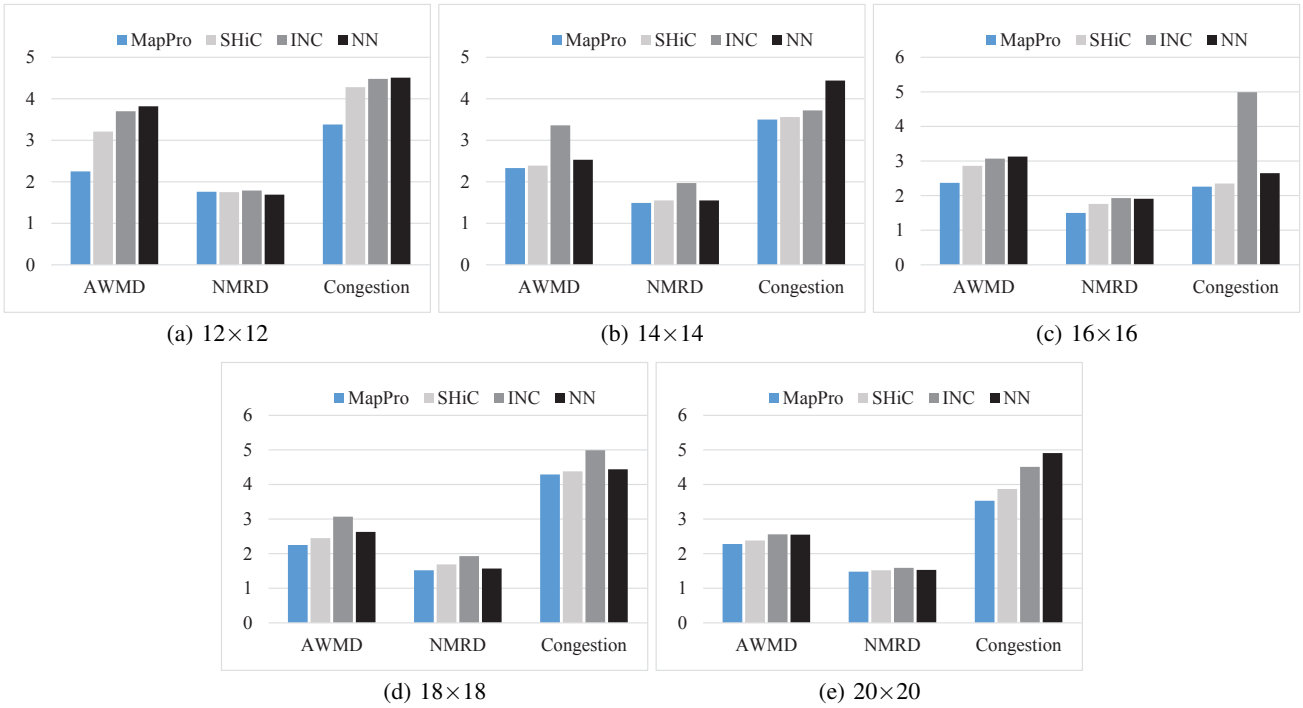
External Congestion: Number of contended packets belonging to different applications is the external congestion. We trace the packets congested from different applications as a metric representing the interference of one application's traffic on another.

MRD and ***NMRD***: Mapped region dispersion (MRD) is the average of pairwise Manhattan distances of all communicating nodes of a mapped application. Higher MRD indicates longer distance



(a) SG-1 (b) SG-2 (c) SG-3 (d) SG-4

Figure 7: VC values for different Square Groups with 4 applications mapped



(a) 12×12 (b) 14×14 (c) 16×16

(d) 18×18 (e) 20×20

Figure 8: Performance, Congestion and Dispersion of first node selection strategies over different network sizes

Table 5: AMD, MRD and APL of first node selection strategies over 20×20 NoC

Method	AMD	MRD	Avg. Latency
MapPro	2.24	2.60	21.37
SHiC	2.31	2.68	21.14
INC	2.49	2.82	21.86
NN	2.42	2.70	21.65

among communicating nodes and hence more probability of interfering with mapped regions of other applications. The Normalized MRD (NMRD) is a metric that evaluates the squareness of the mapped region, independent of the size of the application such that NMRD for a perfect square region is 1 [12]. A dispersed mapping results in NMRD higher than 1, indicating more fragmented regions.

4.2 Experiment Setup

We evaluate our proposed approach against other relevant peer works on a cycle-accurate many-core platform implemented in SystemC. We used a pruned version of Noxim [14] to provide the communication architecture between Processing Element modules. Dynamic application mapping is handled through a central manager (CM) assigned to the node $n_{0,0}$, which is responsible for servicing incoming application requests by finding a suitable first node. It behaves as a master core that implements software routines for fist node selection, task allocation and updating *VC* values of other nodes. Since it holds information regarding mapping decisions and task allocation, it automatically updates *VC* values of nodes with arrival of a new application without any communication overhead. Applications of different sizes in the range of 4 tasks up to 20 tasks with synthetic traffic patterns are generated from [1] for evaluation. The communication volume of each application is distributed as gaussian variation. In addition to the synthetic traffic, we also used soft real-time applications of MPEG4 and VOPD in our eval-

uation. Applications arriving the system enter the schedule's FIFO like buffer and are serviced in first-come-first-serve basis. The CM services incoming application's request by finding a suitable first node, subject to spatial availability of the network. Applications that finish execution are released from the network, making space for new applications to enter.

4.3 Results

We simulated first node selection methods of MapPro, SHiC, INC and NN over a same sequence of 200 applications that entered, executed and released from the system. We collected AWMD, external congested packets and NMRD metrics to evaluate performance, congestion and dispersion of the approaches respectively over network sizes ranging from $12\times$ 12 up to $20\times$ 20. Figure 8 shows these evaluation metrics over different network sizes. MapPro has better AWMD, NMRD and external congestion in comparison with all the other first node selection strategies. It performs particularly well in minimizing external congestion and dispersion due to allocation of square shaped regions and selection of contiguous regions for every new application. The *turn around time* of an application is the time elapsed between its arrival at the scheduler until it exits the system, while execution time is the time elapsed from the task allocation till the end of execution. MapPro provides a turn around time that is same as execution time unlike the other strategists which spend considerable amount of time in calculating an optimal region. The other metrics of AMD, MRD and Average packet latency (APL) for different network sizes are presented in Table 1 - 5. MapPro has a better MRD attributed to its favoring of square shaped regions. A square has the least possible radius among convex regions and thus the maximum hop distance in a square is limited (as opposed to a rectangle). This also reflects on AMD, although AMD and APL can be influenced by the chosen mapping algorithm CoNA to a large extent.

5. CONCLUSIONS

A proactive region selection strategy (MapPro) for mapping applications onto NoC-based many-core systems at runtime was proposed. Our strategy proactively calculates the propagated impact of spatial availability and dispersion on the network with every new mapped application. We exploited the idle time between two consecutive mapping requests to perform quantitative analysis on the network and find rational candidate regions to accommodate an incoming application that results in minimal congestion and dispersion. Our approach eliminates the overhead caused with typical reactive mapping approaches. The performance, congestion and dispersion metrics are used to compare MapPro against other recently proposed first node selection methods. Simulations of different traffic patterns over different sizes of networks showed considerable improvement in terms of higher network performance, lower congestion and dispersion, at a minimal execution time. First node selection based on power constraints and thermal issues is planned for future work.

Acknowledgment

The authors acknowledge the financial support by the Academy of Finland project entitled "MANAGE: Data Management of 3D Systems for the Dark Silicon Age", University of Turku graduate school (UTUGS), EU COST Actions IC1103: Manufacturable and Dependable Multicore Architectures at Nanoscale (MEDIAN) and IC1202: Timing Analysis on Code-Level (TACLe). The authors want to thank Mr. Mohammad Fattah for setting up NoC based system simulator with dynamic mapping feature.

6. REFERENCES

[1] TGG: Task Graph Generator. *URL: http://sourceforge.net/projects/taskgraphgen/*, 2010.

[2] M. Al-Faruque et al. Run-time adaptive on-chip communication scheme. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 26–31. IEEE, 2007.

[3] A. Amoura et al. Scheduling algorithms for parallel Gaussian elimination with communication costs. *IEEE Transactions on Parallel and Distributed Systems*, 9(7):679–686, 1998.

[4] P. Bogdan et al. Quale: A quantum-leap inspired model for non-stationary analysis of noc traffic in chip multi-processors. In *Proceedings of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, pages 241–248. IEEE Computer Society, 2010.

[5] P. Bogdan and R. Marculescu. Non-stationary traffic analysis and its implications on multicore platform design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(4):508–519, 2011.

[6] E. Carvalho et al. Heuristics for Dynamic Task Mapping in NoC-based Heterogeneous MPSoCs. In *Proc. of the International Workshop on Rapid System Prototyping*, pages 34–40, 2007.

[7] C. Chen-Ling et al. 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*, 27(10):1866–1879, 2008.

[8] W. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Proc. of Design Automation Conference*, pages 684–689, 2001.

[9] D. Dobkin et al. Searching for empty convex polygons. *Algorithmica*, 5(1-4):561–571, 1990.

[10] A. Faruque et al. Adam: run-time agent-based distributed application mapping for on-chip communication. In *Proceedings of the 45th annual Design Automation Conference*, pages 760–765. ACM, 2008.

[11] M. Fattah et al. CoNA: Dynamic application mapping for congestion reduction in many-core systems. In *Proc. of the International Conference on Computer Design*, pages 364–370, 2012.

[12] M. Fattah et al. Smart hill climbing for agile dynamic mapping in many-core systems. In *Proc. of the Design Automation Conference*, pages 1–6, 2013.

[13] M. Fattah et al. Adjustable Contiguity of Run-Time Task Allocation in Networked Many-Core Systems. In *Proc. of the Asia and South Pacific Design Automation Conference*, pages 349–354, 2014.

[14] F. Fazzino et al. Noxim: Network-on-chip simulator. *URL: http://sourceforge.net/projects/noxim*, 2008.

[15] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

[16] S. Kobbe et al. DistRM: distributed resource management for on-chip many-core systems. In *Proc. of the international conference on Hardware/software codesign and system synthesis*, pages 119–128, 2011.

[17] S. Murali et al. Bandwidth-constrained mapping of cores onto NoC architectures. In *Proc. of Design, Automation & Test in Europe Conference & Exhibition*, pages 896–901, 2004.