Adjustable Contiguity of Run-Time Task Allocation in Networked Many-Core Systems

Mohammad Fattah, Pasi Liljeberg, Juha Plosila, Hannu Tenhunen
Department of Information Technology, University of Turku, Turku, Finland
Email: {mofana, pakrli, juplos, hanten}@utu.fi

***Abstract*—In this paper, we propose a run-time mapping algorithm, CASqA, for networked many-core systems. In this algorithm, the level of contiguousness of the allocated processors (α) can be adjusted in a fine-grained fashion. A strictly contiguous allocation ($\alpha = 0$) decreases the latency and power dissipation of the network and improves the applications execution time. However, it limits the achievable throughput and increases the turnaround time of the applications. As a result, recent works consider non-contiguous allocation ($\alpha = 1$) to improve the throughput traded off against applications execution time and network metrics. In contradiction, our experiments show that a higher throughput (by 3%) with improved network performance can be achieved when using intermediate α values. More precisely, up to 35% drop in the network costs can be gained by adjusting the level of contiguity compared to non-contiguous cases, while the achieved throughput is kept constant. Moreover, CASqA provides at least 32% energy saving in the network compared to other works.**

***Keywords*—Processor allocation; Application Mapping; Dynamic Many-Core Systems; Contiguous Task Mapping;**

I. Introduction

According to the International Technology Roadmap for Semiconductors [1], by 2020 Multi-Processor Systems-on-Chip (MPSoCs) will integrate hundreds of processing elements (PEs) connected by a Network on chip [2] (NoC) based communication infrastructure. NoCs provide a regular platform for connecting the system resources and make the communication architecture scalable and flexible compared to traditional bus or hierarchical bus architectures.

Such many-core systems will feature an extremely dynamic workload where an unpredictable sequence of different applications enter and leave the system at run-time. Applications are made up of a set of concurrent tasks which communicate to synchronize and exchange data among each other. This featured dynamic nature is handled through the mapping function of the system manager. This is to allocate available system resources to the tasks of applications at run-time.

Communication pattern of underlying NoC is directly influenced by the way communicating tasks are allocated to the PEs. The impact of mapping strategy on the performance of system is emphasized in many-core systems [3] as well as in recent supercomputers [4]. Placing communicating tasks in distant proximity will increase the power dissipation [5] and the congestion probability of the the network. In the presence of congestion, where different messages contend for the same network resources, the path length becomes an issue especially for large packet sizes [6]. Studies show up to eight fold increase in the message latency [4] and a doubled execution time of applications [7], in the absence of sophisticated allocation methods.

Accordingly, a contiguous allocation is preferred in which tasks of an application are settled on close proximity. Regarding the dynamic nature of many-core systems, however, a contiguous solution might not be always possible for an incoming application. Limiting to only contiguous solution, hence, keeps the applications waiting unnecessarily until the required contiguous area becomes available. This will degrade the achievable throughput (the total number of applications that complete their execution per time unit) and the turnaround time (total time between submission of an application and its completion) of each application. As a result, state-of-the-art dynamic mapping algorithms [8]–[11] allow a non-contiguous processor allocation with no limit on the level of dispersion.

In this paper we propose an allocation algorithm, CASqA, in which the level of desired contiguity (denoted by α) can be adjusted in fine-grain between strictly contiguous ($\alpha = 0$) to unlimited non-contiguous ($\alpha = 1$) solutions. When a contiguous solution is not possible, the application is mapped only if it can be mapped within the desired contiguity. This establishes a balance between the increased waiting time of applications (contiguous mapping) and the crippled performance (non-contiguous solutions). Interestingly, results show that the maximum throughput is not achieved in non-contiguous allocation ($\alpha = 1$), but when using intermediate α values ($0 < \alpha < 1$). Accordingly, the same throughput as non-contiguous mappings can be achieved with up to 30% improvement in the network costs. Note that in this paper we equivalently use the terms "the level of allowed dispersion" and "the level of desired contiguity" to express α.

In addition to the level of contiguity, the order in which tasks of an application are placed, within an area, affects the network costs and thus the application performance. Consequently, CASqA considers the pattern of the communication within an application tasks while allocating the processors. To this end, CASqA is equipped with a single evaluation metric representing both the power dissipation and congestion probability of the network incurred by the candidate mapping solutions.

The rest of the paper is organized as follows: In Section II the mapping problem as well as the contiguousness and power-congestion metrics are formally defined, while Section III discusses related work. Section IV describes our novel contiguity adjustable square allocation (CASqA) method. The simulation setups along with the effect of adjusting the level of desired contiguity (α) on the system performance is presented and discussed in Section V. Moreover, this Section illustrates a comparison with other state-of-the-art methods. Finally, Section VI concludes the paper and discusses some potential future works.

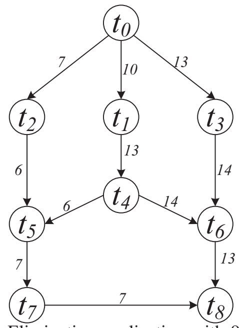


Fig. 1: A Gaussian Elimination application with 9 tasks and 11 edges.

II. Definitions

A. Problem Definition

Mapping algorithms try to optimally allocate system resources to the concurrent tasks of a requested application. 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 [12] which is extracted using TGG [13] is shown in Fig. 1. Note that $w_{i,j}$ of an edge $e_{i,j}$, i.e. the amount of data transferred from task t_i to t_j, is indicated on each edge.

An architecture graph $AG(N, L)$ describes the communication infrastructure of the processing elements. We consider a simple $W \times H$ 2D-mesh NoC with the XY wormhole routing (Fig. 2(a)). 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 a 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:

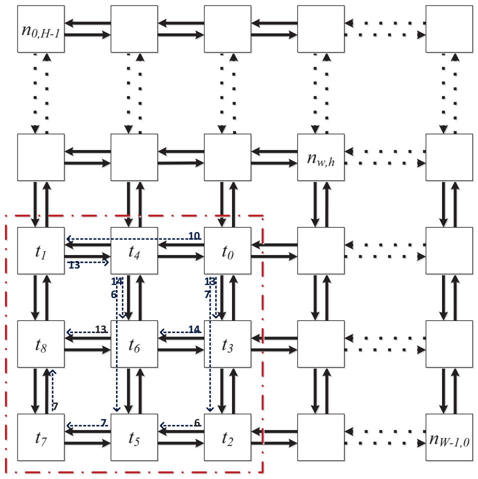
$$map : T \rightarrow N, s.t.map(t_i) = n_{w,h}; \forall t_i \in T, \exists n_{w,h} \in N \quad (1)$$

Fig. 2(a) illustrates a possible mapping of the application in Fig. 1, onto the described platform. For simplicity, we denote a node where a task, t_i, is mapped onto as nt_i and the routing path used by communication edge $e_{i,j}$ as $pck_{i,j}$; i.e. the traversed path by the packet sent from nt_i to nt_j. Note that due to XY routing assumption, the path can be determined once the application is mapped.

B. Evaluation Metrics

As mentioned before, traffic congestion and power dissipation are two main concerns of on-chip domain. Congestion increases the network latency dramatically [6] and cripples the application performance. The power dissipation and cooling issues are also known as the limiting walls [14], [15] of future silicon technologies.

The energy dissipation of the network related to an obtained mapping is a function of both the volume ($w_{i,j}$) and the traversed path ($pck_{i,j}$) of the communications [5]; while the congestion is related to the packets sharing the same path along their delivery. Contiguous mapping kills two birds with one stone. (i) It diminishes the network power consumption by decreasing the $pck_{i,j}$ lengths of the mapped application. (ii) It isolates the data communication of each application and hence the congestion among them (external congestion). As mentioned, however, limiting the mapping algorithm to only contiguous solutions will degrade the system throughput and applications turnaround time. Accordingly, recent works allow non-contiguous mapping.



$ICE_{0,1} = 10 + 10$
$ICE_{0,2} = 20 + 7$
$ICE_{0,3} = 20$
$ICE_{1,4} = 13$
$ICE_{2,5} = 6$
$ICE_{3,6} = 14$
$ICE_{4,5} = 20 + 6$
$ICE_{4,6} = 20$
$ICE_{5,7} = 7$
$ICE_{6,8} = 13$
$ICE_{7,8} = 7$

$\Rightarrow \sum ICE_{i,j} = 173$

$\sum w_{i,j} = 110, \Rightarrow$

$ICEB_{Ap} = 1.57$

(b)

Fig. 2: (a) Mesh-Based platform with an application mapped onto it (the highlighted region.); and (b) $ICEB$ calculation of the mapped application

Several works (e.g. [9], [10], [16]–[18] etc.) considered the average pairwise Manhattan distance (MD, also known as L1 distance) of allocated nodes as a metric (called MRD in [19]) to assess the contiguity of a mapped applications. Bender et al. [20] showed that the most contiguous region, i.e. with the least MRD value, would be almost circular. Regarding the mesh topology of the network, however, a circular processor allocation will generate irregularity in remaining available nodes and more area fragmentation in long term.

As an alternative to circular; rectangular allocation forms regular regions, and isolates data communications of (and thus decreases the congestion probability between) different applications. Targeting both the contiguity and regularity of allocation method leads to square shape which is a rectangle with the smallest MRD. It is proved in [17] that a square allocation is a $7/4$-approximation in the worst case in contrast to MRD minimization which is 2-approximation.

The preference of square allocation over circular is also observed in [19]. The $NormalizedMRD$ is provided as a metric to asses both the squareness and contiguousness of a mapped application, independent of the application size ($|T|$):

$$NMRD_{map(Ap)} = 1 + \frac{|MRD_{map(Ap)} - MRD_{SQ(|T|)}|}{MRD_{SQ(|T|)}} \quad (2)$$

where $MRD_{SQ(|T|)}$ is the MRD value for a square with $|T|$ nodes.The $NMRD$ value of 1 means a convex and squared area. $NMRD$ increases as the mapped area is getting more fragmented and/or less similar to a square shape.

As a result, CASqA collects required number of available PEs within a square shape as long as the allowed level of dispersion is respected. In addition to the level of allowed dispersion, the order in which tasks of an application are mapped onto a set of nodes will affect the energy dissipation and congestion probability of the network. This is referred to as internal congestion probability which stands for the network links shared among the data flows of the same application.

We define the *Internal Congestion and Energy* (ICE) of an edge, $e_{i,j} \in E$, in a mapped application as:

$$ICE_{i,j} = \sum_{\forall l_k \in pck_{i,j}} w_{l_k} \quad (3)$$

where, w_{l_k} is the total value of $w_{i,j}$ of edges which their $pck_{i,j}$ includes the link l_k. Let us follow the ICE calculation of $e_{0,2}$ for the mapped application in Fig. 2(a). The path between nt_0 and nt_2 ($pck_{0,2}$) is composed of two links:l_{nt_0,nt_3} and l_{nt_3,nt_2}. The first link is utilized by two edges: $e_{0,3}$ and $e_{0,2}$, so its w_{l_k} value is $w_{0,3} + w_{0,2} = 20$. The w_{lk} for the second link is 7 as it is utilized only by $e_{0,2}$. The $ICE_{0,2} = 27$ is calculated by the summation of two w_{l_k} values.

According to the definition, the w_{l_k} value of a given link increases as being shared among several packets of an application. This reflects the increased probability of internal congestion. Moreover, the $ICE_{i,j}$ of a given data flow relates to the MD of corresponding nodes (nt_i and nt_j). This reflects the dissipated energy in the network. Consequently, ***ICE per Bit*** ($ICEB$) value for a mapped application is given by the sum of ***ICE*** values of all edges averaged by the total weights of the edges:

$$ICEB_{map(Ap)} = \frac{\sum ICE_{i,j}}{\sum w_{i,j}} \tag{4}$$

The larger the $ICEB$ value, the higher energy dissipation and congestion probability. The $ICEB = 1$ means that all communications of the application are handled in one hop distance without sharing any common link. Fig. 2(b) shows the $ICEB$ evaluation of the example mapping of Fig. 2(a). Later in Section IV, CASqA uses $ICEB$ metric to select among available candidate nodes for allocating a task.

III. Related Work

There are several works dealing with management of dynamic workload in massively-parallel systems. Early works in the supercomputer domain map applications only onto convex set of nodes. While recent works focus on non-contiguous processor allocation methods for supercomputers and many-core systems. In the following we study some of them in both domains. They start the allocation from an available *first node* and allow non-contiguous allocation.

Bender et al. [17] presented the MC1x1 method for processor allocation problem. Starting from an available node, MC1x1 explores the smallest square around that node; i.e. the square with radius 1; and collects all available nodes. The square radius is increased until the required number of available nodes is found. They start the same procedure from all available nodes, and finally select the *first node* which resulted in the smallest MRD. However, the communication pattern (task graph) of the application is not considered in their task allocation.

Carvalho et al. [21] presented Nearest Neighbor (NN) heuristic. NN maps each task of an application onto the nearest free neighbor of the task communicating with. This algorithm considers neither the energy dissipation nor the congestion. Later, they introduce Best Neighbor (BN) [8] which selects the best nearest free neighbor according to communication weights. They do not provide any method to converge the mapping area around the selected *first node* which significantly increases the congestion.

Chou et al. [10], in their incremental approach, broke down the mapping problem into two steps: region selection, and task allocation. First, in the region selection, they try to find a set of nodes to minimize MRD for both the application region and remaining nodes. Afterwards, the application tasks are mapped onto the chosen region to minimize the energy dissipation. Their method, however, results in increased dispersion, as they frequently might select an isolated *first node* in their region selection.

Algorithm 1 CASqA pseudo-code in expanding the current radius according to the level of allowed dispersion.

```
Inputs:
  $TG(T, E)$: Task graph of the requested application, $Ap$.
  $t_f$ and $n_f$: The first task of the application and first node of the allocation process.
  $\alpha$: The allowed level of dispersion, $0 \le \alpha \le 1$.
Output:
  $map : T \to N$.
Variables:
  $UNM$, $MET$ and $MAP$: Sets of unmet, met and mapped tasks.
  $R_{max}$: The maximum radius the application is allowed to get dispersed around the $n_f$.
  $r$: The current radius of the square around the $n_f$.

Body:
 1: $nt_f \leftarrow n_f$;
 2: $MAP \leftarrow t_f$;
 3: $MET \leftarrow$ set of tasks connected to $t_f$ in $TG$;
 4: $UNM \leftarrow T - (MAP + MET)$;
 5: for $r = 1 \to R_{max}$ do
 6:     allocate the available nodes within the current radius (Algorithm 2);
 7:     if $MAP = T$ then
 8:         return success;
 9:     else if $r = R_{max}$ and $|T| - |MAP| < \tau$ then
10:         $R_{max} \leftarrow R_{max} + 1$;
11:         $\tau \leftarrow \tau \times \alpha$;
12:     end if
13: end for
14: return failure;
```

Fattah et al. [9] aim at minimizing the internal congestion via CoNA approach. CoNA selects the node which has the largest number of free neighbors as the *first node* in the allocation. Then the task with largest degree is mapped onto that node. The task graph is traversed in breadth-first order, and neighbors making smaller squares are preferred. Later, the effect of *first node* selection method on the mapping performance is studied [19]. They show that a square shape is preferred for processor allocation. However, the utilized mapping algorithm, CoNA, does not collect the available nodes in a square shape.

There are several other works dealing with the dynamic workload management of many-core systems and/or supercomputers. They either limit the allocation to contiguous set of nodes or allow non-contiguous allocation with no limits. However, to the best of our knowledge, this is the first work allowing an adjustable level of allowed dispersion in the processor allocation. Results show that this improves the network costs while keeping the throughput as high as in non-contiguous solutions.

IV. CASqA Mapping Algorithm

The pseudo-code of the proposed mapping algorithm with adjustable contiguity is listed in Algorithm 1[1]. The mapping is started by exploring the available nodes in the smallest square around the *first node*. CASqA expands the radius by one only after all the available nodes in the current radius are allocated.

Expanding the current radius continues up to the maximum radius, R_{max}, which is initialized according to (5). This is the minimum radius for the square which can fit $|T|$ nodes. The R_{max} value is then adapted at run-time (line 9–12) according to the level of allowed dispersion which is adjusted through the α. The α is a real value, where $\alpha = 0.0$ means a tight mapping while $\alpha = 1.0$ stands for no limit on R_{max}.

$$R_{max} = \left\lfloor \frac{\left\lceil \sqrt{|T|} \right\rceil}{2} \right\rfloor \tag{5}$$

[1]The source code is available at http://users.utu.fi/mofana/CASqA.html

The maximum radius in which CASqA looks around for available nodes (R_{max}) is adapted through the τ threshold. It shows the maximum number of tasks that can be remained unmapped in order to allow the increase of R_{max}. In other words, once the current radius reaches the R_{max}, more dispersion can happen only if a limited number of tasks (max. τ) are remained for mapping (line 9). At each iteration of increasing the R_{max}, the τ threshold is tightened by multiplying it by α. Note that, the initial value for τ is $|T| \times \alpha$.

For instance, having $\alpha = 0.0$ keeps $\tau = 0$ from the beginning and the R_{max} is never increased from its initial value in (5). Hence, CASqA will map the application only if the required number of available nodes ($|T|$) can be collected within the smallest square. Otherwise, it will return a *failure* (line 14) and the mapping will be restarted from another *first node* or paused until the required area is available. On the other hand, when $\alpha = 1.0$ the τ is always kept to $|T|$ and thus the R_{max} can be increased with no limits until all the tasks are mapped onto the system.

As mentioned before, in addition to the provided adjustable contiguity, CASqA deals with the order in which tasks of the application are mapped onto the system nodes. The following subsection describes the CASqA approach in reducing the power dissipation and congestion probability of the network via $ICEB$ metric.

A. Task Mapping Details

CASqA assumes the task graph to be undirected, and meets and maps tasks through their predecessor tasks (called parents). For a given application, there are three sets of task: Tasks that are already mapped onto the system (MAP), tasks that are met through their mapped parent but are not mapped yet (MET), and the tasks that are not met yet and thus not mapped (UNM).

Through the initializing phase of the algorithm (lines 1–4), the *first node* is allocated to the first task of the application. The first task is added to MAP set and the tasks connected to it are added to MET set. The rest of the tasks are also moved to UNM set. Note that a basic random algorithm or a sophisticated heuristic, like SHiC [19], can be utilized for the *first node* selection. The task with the maximum degree is also selected as the first task.

The pseudo-code of CASqA for mapping application tasks onto the available nodes of the current radius, r, is listed in Algorithm 2. Within the current radius, MET tasks are preferred to get mapped onto the closest node to their parent. Accordingly, for each MD value, all MET tasks are examined for possible mappings. Given a MET task (t_c), list of available candidate nodes ($\widetilde{N}$) in the current radius which are MD hop far from parent of t_c is extracted (line 3). If at least one candidate node exists to map t_c onto it (line 4), then the one which leads to minimum value of $ICEB$ is selected and the task is moved from the MET set to the MAP set (lines 5–6). Consequently, those of unmet tasks (UNM) that exchange data with the t_c will be moved to the MET set (lines 7–8). Now that new tasks are added to MET set, it might be possible to map them within one hop distance from their parent, t_c. Hence, the MD will be reset to one (line 9).

Let us follow the mapping process of the Gaussian Elimination application of Fig. 1 (with 9 tasks) onto the system configuration as shown in Fig. 3 (a)–(f). According to (5), the initial value for R_{max} will be 1. Since there are 8 nodes

Algorithm 2 *CASqA pseudocode in mapping application tasks within the available nodes of the current radius.*

```
1: for $MD = 1 \to 4 \times r$ do
2:     foreach $t_c \in MET$ do
3:         $\widetilde{N} \leftarrow$ available nodes in current radius with $MD$ hop distance from $t_c$ parent;
4:         if $\widetilde{N} \neq \emptyset$ then
5:             $nt_c \leftarrow$ the node of $\widetilde{N}$ which results in a smaller $ICEB$ value;
6:             move $t_c$ from $MET$ to $MAP$;
7:             if there are tasks in $UNM$ connected to $t_c$ then
8:                 move tasks connected to $t_c$ from $UNM$ to $MET$;
9:                 $MD \leftarrow 1$;
10:            end if;
11:        end if
12:    end for
13: end for
```

available within this radius around the selected *first node* CASqA will need to increase the search radius, R_{max}, to collect one available node beyond the first square. Hence, CASqA cannot map the application if one sets $\alpha \leq 1/9$; i.e. the initial value for $\tau \leq 1$. For the sake of simplicity, we assume $\alpha = 1.0$.

In the beginning, first task of the application (t_6) is mapped onto the *first node*, its connected tasks $\{t_3, t_4, t_8\}$ are added to MET set, and both r and MD are initialized to 1, Fig. 3 (a). As the next step, $t_3 \in MET$ can be mapped onto the smallest square ($r = 1$) around the *first node* with 1 hop distance ($MD = 1$) from its parent, t_6. All the candidate nodes (asterisk ones) result in the same $ICEB$. Hence, t_3 is mapped onto one of them alternatively and moved from MET set to MAP set, Fig. 3 (b). Consequently, t_0 is added to MET set as it is connected to t_3 in the task graph. Within next three iterations, MET tasks t_4, t_8, t_0 are mapped one by one with r and MD values equal to 1 and new tasks are MET through them, as shown in Fig. 3 (c). Note that r and MD values are still equal to 1.

Now, two nodes (asterisks in Fig. 3 (c)) can be selected for t_1, regarding the r and MD values. As t_1 also exchanges data with t_0, the bottom node results in less $ICEB$ value and thus is selected for t_1. Afterwards, t_5 is mapped onto the upper node of t_4, as it is the only node available regarding the current r and MD values. The system configuration after t_1 and t_5 being mapped is shown in Fig. 3 (d). Now that none of the MET tasks can be mapped onto 1 hop distance from their parents, the MD value is increased to 2. Consequently, t_2 is mapped within 2 hop distance from its parent (Fig. 3 (e)). Now that there is not any node available in the square with $r = R_{max} = 1$ and $\alpha = 1$, the r and R_{max} are increased to 2 and the MD is reset to 1. The only unmapped task is t_7 which should be mapped onto t_8 neighborhood. As it is not feasible to map t_7 within one hop distance from t_8, the MD is increased to 2 and t_7 is mapped in the last step, Fig. 3 (f).

Accordingly, the proposed CASqA algorithm lets adjust the level of expected contiguity of allocated nodes and tries to reduce the power dissipation and congestion probability of applications. In the next Section, we study the throughput and performance of the system, regarding different levels of contiguity.

V. Results and Analysis

In this section, we assess our CASqA mapping algorithm in two main aspects. First we evaluate the impact of adjusting the level of desired contiguity on the system performance. Then we compare CASqA with other state-of-the-art methods. Several applications with 4 to 35 tasks are generated using TGG

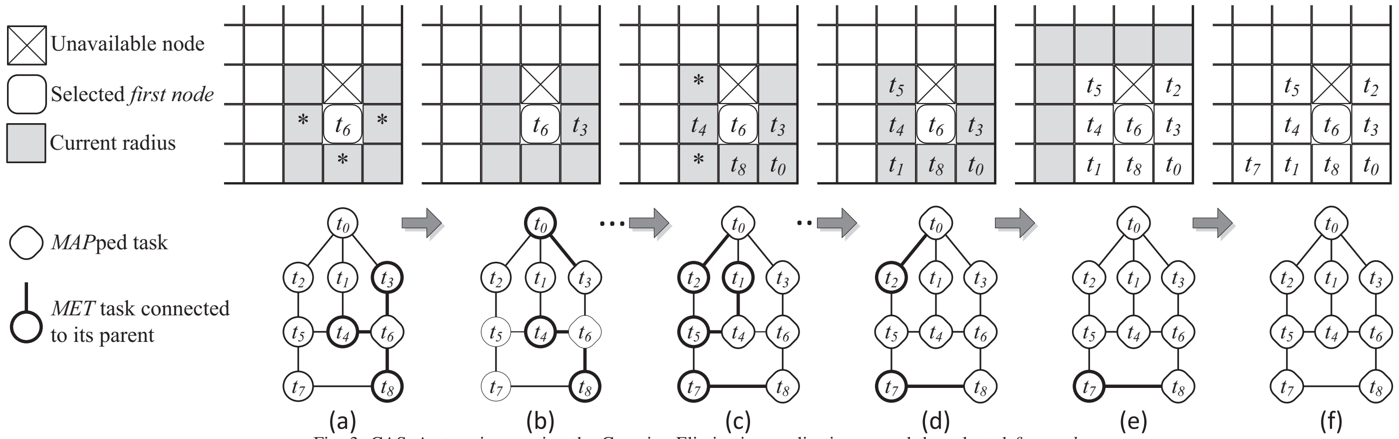


Fig. 3: CASqA steps in mapping the Gaussian Elimination application around the selected *first node*.

[13] 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 [22], as its communication architecture. A 16×16 network is instantiated, and each set of experiments are performed millions of cycles where thousands of applications enter and leave the system.

A random sequence of applications are entered the scheduler FIFO. This 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 an application is scheduled if and only if there is enough available nodes in the system. An allocation request for the scheduled application is sent to the central manager (CM) of the platform residing in the node $n_{0,0}$. CM selects the *first node* using SHiC [19] method, and executes CASqA algorithm with the target α (denoted as CASqA$^{\alpha}$). Upon a mapping failure, new *first nodes* are selected until the current application being mapped. The successfully mapped application is allocated to the corresponding nodes, where each node emulates the behavior of its allocated task.

A. Contiguity Adjustment

In this subsection we study the effect of the level of allowed dispersion on the system performance. To this end, different experiments are done with α varying from 0 to 1 by 0.1 steps. The next application is pushed into the scheduler FIFO as soon as there is enough resources available for the application. This is to achieve the maximum possible throughput of the system. Accordingly, applications do not experience any waiting time in CASqA$^{1.0}$. Note that the turnaround time of an application is calculated from the moment it is pushed into the scheduler FIFO. While, the execution time is calculated from the moment the application tasks are allocated to the nodes.

A summary of the obtained results is demonstrated in Fig. 4. As can be seen, limiting the system to only contiguous allocations ($\alpha = 0$) minimizes the execution time of the applications. However, the achievable throughput is degraded by 17% as a result of the increased turnaround time.

Although releasing CASqA from contiguous allocations will improve the achievable throughput and decreases the turnaround time, this is not the case for $\alpha > 0.6$. Afterwards, the increased execution time will affect the turnaround time and the system throughput starts degrading. Note that the amount of the throughput degradation is less that 3%. In other

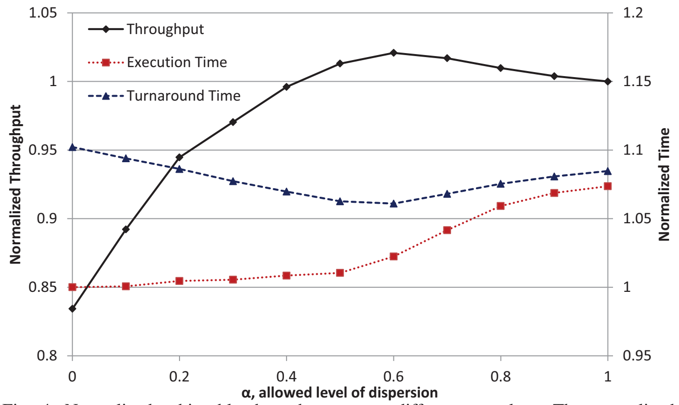


Fig. 4: Normalized achievable throughput across different α values. The normalized average execution and turnaround time of applications are also shown in the right axis.

words, the throughput gain of CASqA$^{0.6}$ over CASqA$^{1.0}$ is quite negligible.

The main conclusion is that the same throughput as CASqA$^{1.0}$ can be achieved when $\alpha \approx 0.4$. This equal throughput is achieved while there is almost 6% improvement in execution time, and 25 and 35 % gain in the latency and energy dissipation of the network, respectively, as shown in Fig. 5.

This is worth mentioning that in only 8% of the mapping failures of CASqA$^{0.4}$, trying new *first nodes* within the same configuration leads to mapping success. Put differently, in 92% of the mapping failures, the required contiguity is obtained only when some nodes get available (their application execution is completed). This proves that the obtained gains are not due to the change of the *first node*, but because of the adjusted

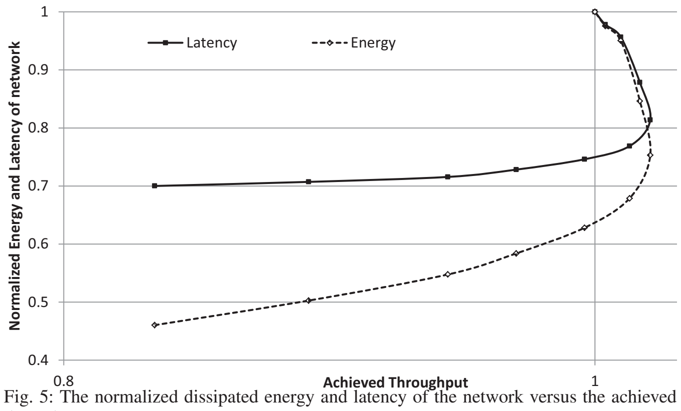


Fig. 5: The normalized dissipated energy and latency of the network versus the achieved throughput.

TABLE I: Results of Different Mapping Algorithms

Mapping	$NMRD$	$E_{norm.}$	L_{avg}	$\%Congestion$
CASqA$^{0.5}$	1.13	1.00	41.00	30.52
CASqA$^{0.5*}$	1.13	1.25	48.23	37.70
CoNA	1.69	1.48	50.32	38.40
NN	1.97	1.56	55.53	41.98

contiguity. Moreover, according to Fig. 4, there is around 1% time overhead for mapping of an application and allocating its tasks to the selected nodes. This is the difference between the execution and turnaround time in CASqA$^{1.0}$.

B. ICEB Effect and Comparison

In this subsection, we make a comparison between CASqA and other state-of-the-art mapping algorithms and study the effect of the order of task placement within a area (Algorithm 2). Accordingly, several experiments are performed with exploiting different mapping algorithms: CASqA$^{0.5}$, NN [21] and CoNA [19]. Note that the experiments for CASqA case are executed twice, once (denoted as CASqA$^{0.5*}$) with arbitrary mapping of tasks onto the available nodes of the current radius. The rate in which the random sequence of applications are injected to the scheduler FIFO is set to the achieved throughput for $\alpha = 1.0$ in the previous subsection.

The extracted results are summarized in Table I. The normalized values for the dissipated energy of the network is shown in $E_{norm.}$ column, while the next column indicates the average latency, in clock cycles, of the network. The last column is the percentage of data packets experienced congestion along their path. As can be seen, thanks to the moderated dispersion, CASqA leads to at least 32% energy saving as well as 19% improvement in average latency of the network.

The comparison between two first rows, emphasizes the importance of tasks arrangement ($ICEB$ metric) within a area. However, the two middle rows show the superiority of contiguous mapping. Arbitrary task arrangement does not degrade the performance as non-contiguous allocation does.

Last but not least, CASqA$^{0.5}$ reduces the standard deviation and worst case latency of the network by 2 and 4 times, respectively, compared to CoNA. The reduction is 2.2 and 10.5 times for CASqA$^{0.0}$. This shows the potential of incorporating the run-time requirements into the selection of α values to deliver QoS at system-level. More details are left as future work.

VI. Conclusion and Future Work

In this paper, we proposed a contiguity adjustable square allocation (CASqA) method. The proposed method attempts to form a square shaped area of available nodes around the selected *first node*. CASqA expands the exploration square according to the level of desired contiguity at run-time.

Experiments showed around 35% improvement versus non-contiguous approaches with the same throughput. Moreover, CASqA is equipped with $ICEB$ metric which resulted in improved network performance characteristics.

Regarding the variety of user demands and the application requirements, a different level of contiguousness can be suited for each run-time case. As a future work, we plan to incorporate the run-time criteria into α adjustment. Accordingly, a system-level adaptive approach,for e.g. QoS management, can be obtained. This will lead to increased performance while delivering the required functionality to each application.

References

[1] S. I. Association *et al.*, “International technology roadmap for semiconductors (ITRS), 2011 edition,” 2011.

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

[3] A. K. Singh *et al.*, “Mapping on multi/many-core systems: survey of current and emerging trends,” in *Proceedings of the 50th Annual Design Automation Conference*, 2013, pp. 1:1–1:10.

[4] A. Bhatele and L. Kale, “An evaluative study on the effect of contention on message latencies in large supercomputers,” in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, 2009, pp. 1–8.

[5] T. Ye, L. Benini, and G. De Micheli, “Analysis of power consumption on switch fabrics in network routers,” in *Design Automation Conference, 2002. Proceedings. 39th*, 2002, pp. 524–529.

[6] J. W. van den Brand *et al.*, “Congestion-controlled best-effort communication for networks-on-chip,” in *Proceedings of the conference on Design, automation and test in Europe*, 2007, pp. 948–953.

[7] V. Leung *et al.*, “Processor allocation on Cplant: achieving general processor locality using one-dimensional allocation strategies,” in *Cluster Computing, 2002. Proceedings. 2002 IEEE International Conference on*, 2002, pp. 296–304.

[8] E. de Souza Carvalho, N. Calazans, and F. Moraes, “Dynamic Task Mapping for MPSoCs,” *Design Test of Computers, IEEE*, vol. 27, no. 5, pp. 26–35, 2010.

[9] M. Fattah *et al.*, “CoNA: Dynamic Application Mapping for Congestion Reduction in Many-Core Systems,” in *Computer Design (ICCD), 2012 IEEE 30th International Conference on*, 2012, pp. 364–370.

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

[11] 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*, 2011, pp. 119–128.

[12] A. Amoura, E. Bampis, and J.-C. Konig, “Scheduling algorithms for parallel Gaussian elimination with communication costs,” *Parallel and Distributed Systems, IEEE Transactions on*, vol. 9, no. 7, pp. 679–686, 1998.

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

[14] A. Agarwal and M. Levy, “The KILL Rule for Multicore,” in *Design Automation Conference, 44th ACM/IEEE*, 2007, pp. 750–753.

[15] H. Esmaeilzadeh *et al.*, “Dark silicon and the end of multicore scaling,” in *38th Annual International Symposium on Computer Architecture (ISCA)*, 2011, pp. 365–376.

[16] J. Mache and V. Lo, “Dispersal Metrics for Non-Contiguous Processor Allocation,” Tech. Rep., 1996.

[17] M. A. Bender *et al.*, “Communication-Aware Processor Allocation for Supercomputers: Finding Point Sets of Small Average Distance,” *Algorithmica*, vol. 50, no. 2, pp. 279–298, Jan. 2008.

[18] J. Mache, V. Lo, and K. Windisch, “Minimizing message-passing contention in fragmentation-free processor allocation,” in *In Proceedings of the 10th International Conference on Parallel and Distributed Computing Systems*, 1997, pp. 120–124.

[19] M. Fattah *et al.*, “Smart Hill Climbing for Agile Dynamic Mapping in Many-Core Systems,” in *Proceedings of the 50th Annual Design Automation Conference*, 2013, pp. 39:1–39:6.

[20] C. M. Bender *et al.*, “What is the optimal shape of a city?” *Journal of Physics A: Mathematical and General*, vol. 37, no. 1, p. 147, 2004.

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

[22] F. Fazzino, M. Palesi, and D. Patti, “Noxim: Network-on-chip simulator,” *URL: http://sourceforge.net/projects/noxim*, 2008.