Allocating Irregular Partitions in Mesh-Based On-Chip Networks*

Samuel Rodrigo[1], Frank Olaf Sem-Jacobsen[1], and Tor Skeie[1,2]

[1] Simula Research Laboratory, Norway
[2] Dept. of Informatics, University of Oslo
{srodrigo,frankose,tskeie}@simula.no

Abstract. Modern CMPs require sophisticated resource management in order to provide good utilisation of the chip resources. There exists good allocation algorithms for compute clusters, but these are restricted to specific routing algorithms and not easily transferable to the on-chip domain. We present a novel resource allocation algorithm, TSB, that allows petitions with any shape that is supported by an algorithm implemented using LBDR/uLBDR, and show that this has low complexity and comparable utilisation to UDFlex.

1 Introduction

Chip multiprocessors (CMPs) are becoming as powerful and rich on resources as small computer clusters. Intel's Single Chip Cloud computer (SCC) [1] with 48 cores is a good example of this, and as the name implies, the chip is intended to support multiple concurrent users and applications. With this abundance of resources, efficient management of the on-chip cores is becoming as important as for regular cluster computers.

The efficiency of a CMP to support multiple concurrent applications depends on how well the applications can be allocated resources (cores) on the chip. Some of the key performance points are contiguous versus non-contiguous allocation, performance isolation, and location relative to the memory controllers.

Contiguous allocation means that the application is allocated a set of cores that form a contiguous region, as opposed to non-contiguous allocation where an application can be assigned cores randomly throughout the chip. This impacts the communication bandwidth and latency internal to the application, as well as fragmentation of the on-chip resources. Contiguous allocation gives better application performance, and increases fragmentation since there are strict requirements on which types of partitions that can be allocated.

Performance isolation is a set of different mechanisms that try to make the performance of a single application independent of the state of the rest of the chip. This can be achieved in two ways, using virtual channels or ensuring that the physical network resources are accessible by only a single application. With

* This work has been supported by the project NaNoC (grant agreement no. 248972) which is funded by the European Commission within the Research Programme FP7.

non-contiguous allocation all applications will have to share the physical network resources, so the only option for achieving performance isolation is to assign each application to its own virtual channel. For contiguous regions, however, the routing algorithm can be designed in such a way as to guarantee that all traffic internal to that application remains within the allocated partition. We call this *routing containment.*

Finally, the location relative to the memory controller dictates the cost of accessing the off-chip memory, and also the amount of traffic that will cross through the application partition from other applications accessing the controller.

The effects of the allocation strategies mentioned here have been studied by Triviño et al. [11]. The authors show that contiguous allocation with routing containment gives the overall best performance for the individual applications.

Contiguous allocation with routing containment requires that the application is allocated into logical sub topologies that are supported by the routing algorithm. Current CMP systems rely on XY routing, and routing containment using XY routing requires that the applications are located in sub-meshes. This is very restrictive and can lead to large fragmentation and low chip utilisation. Solheim et al. [9] show that high network utilisation (for clusters) can be achieved by allowing more irregular shapes. However, this requires more flexible routing algorithms to support these shapes. In this case the allocation algorithm presented, UDFlex, relies on Up*/Down* routing.

Recently more sophisticated, yet simple, routing algorithms have been developed for on-chip mesh networks to replace XY. FDOR [8] and LBDR [6] require only a single or a few configuration bits to support a much larger number of different sub-topologies than the simple rectangular meshes supported by XY routing. Evolutions of these simple algorithms such as FJE [7] and uLBDR [6] support a wider range of sub-topologies, even arbitrary shapes, but at an increased implementation complexity.

In this paper we study the improvement in chip utilisation that can be achieved by using these more sophisticated routing algorithms and we devise an allocation algorithm (TSB) that can be efficiently implemented with low complexity and supports any routing implementation from LBDR. The rest of paper is organised as follows. In Section 2 we present some basic resource allocation algorithms from the literature and review studies that concern resource allocation in on-chip systems. We present our novel resource allocation algorithm, TSB, and a minor description of other allocation algorithms in Section 3, and study the efficiency of this algorithm in Section 4. The paper is concluded in Section 5.

2 Related Work

CMP virtualisation (i.e. the ability to efficiently support multiple applications) has been studied in several papers. Flich et al. [2] studied the concept of CMP virtualisation and identified the key challenges: routing, reconfiguration, and allocation (routing containment, etcetera). The application performance with different application isolation mechanisms has been studied in [11] where the authors find that full physical performance isolation (using dedicated network resources) gives the best application performance.

In order to exploit the flexibility of the new routing algorithms such as LBDR, an allocated region needs to be configured to provide routing isolation. This requires efficiently configuration mechanisms. Triviño et al. [10] presents an efficient configuration mechanism for the purpose of reconfiguring allocated regions.

Finally, Triviño et al. [12] combined the configuration mechanism and the LBDR routing function with the UDFlex allocation algorithm. The chip is pre-configured with segment-based routing [5], and UDFlex is used to allocate partitions that are contained segments. As we will see in this paper, UDFlex is quite complex in its implementation which can be challenging for short-lived jobs.

The two most common allocation algorithms for mesh networks are First Fit and Best Fit [14]. These algorithms simply search for a rectangular (sub-mesh) area of the desired size and either choose the first one, or the best one which leaves the maximum contiguous region remaining. UDFlex [9] relies on the well-known Up*/Down* routing algorithm to support more flexible partition shapes. The paper shows how this allows for larger system utilisation/less fragmentation while at the same time maintaining good application performance.

3 Allocation Algorithms

3.1 First Fit and Best Fit

First Fit and Best Fit allocation strategies were presented in order to improve the pool of available shapes for allocating an application. Previous strategies only allowed square sub-meshes. To improve the utilisation and minimise the fragmentation, First Fit and Best Fit, allow for rectangular shapes of a requested size $a \times b$, where a can be different to b. Both strategies keep track of the busy and free nodes, and they differentiate in one key strategy. First Fit allocates the job in the first free sub-mesh area. Best Fit, on the other hand, does an exhaustive search to find the smallest free area to allocate the task, in order to reduce external fragmentation. Both of them operate to allocate each task in a contiguous mode. See an example in Figure 1 of both strategies. The scheduler/allocator wants to allocate a 4-cores parallel application. The area composed of nodes 1, 2, 3, 5, 6, and 7 is enough to allocate the job, but the area of nodes 10, 11, 14, and 15 is also valid. Best Fit would choose the second area over the first one.

Notice that while supporting rectangular sub-meshes over square sub-meshes is an improvement to increase chip utilisation, both strategies suffer from internal fragmentation in certain cases by not allowing for more flexible shapes. See an example in Figure 2. As you can see, with XY routing, we want to allocate a 5-core job into a free sub-mesh area of 3×2 area, the only one available. One of the cores, at node 7 for example, will be reserved for the rectangular region, but it will remain unused and unavailable for other applications. XY routing is a very simple mechanism, but cannot cope with irregular shapes, no matter the reason behind the irregularity.

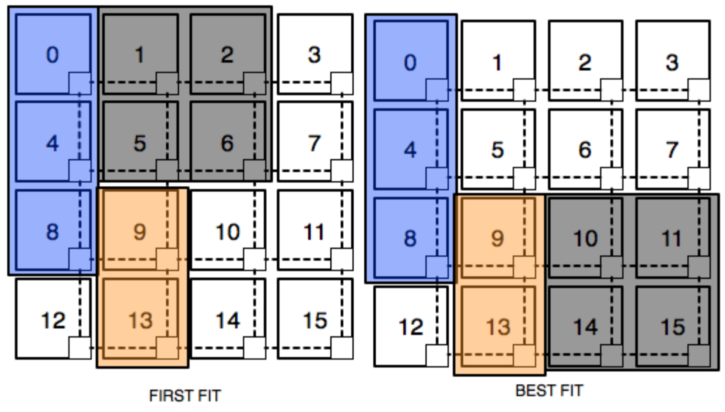


Fig. 1. First Fit and Best Fit example

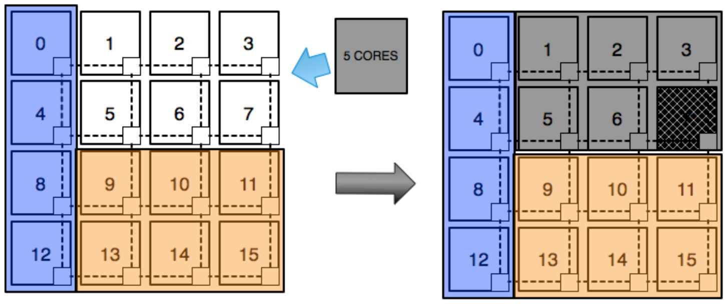


Fig. 2. Internal fragmentation

In the evaluation section, we also present Random Fit. Random Fit is based on the same idea as First Fit and Best Fit, but when allocating a job, instead of choosing the first free sub-mesh area or the smallest one, just performs a pseudo-random choice from the pool of the available ones.

In [14] the complexity of the strategies is presented. First Fit a complexity of $O(a \times b)$, where a and b are the dimensions of the mesh, and Best Fit displays the same time complexity.

3.2 UDFlex

UDFlex is a tree-based allocation strategy based on the Up*/Down* routing algorithm. In Figure 3 we can see an example of a possible Up*/Down* routing configuration, where node 0 has been assigned as the root of the graph/tree. In order to ensure deadlock-freedom, directions in a link are marked as *up* and *down* (in the figure up is the direction of the arrow), and packets are not allowed to be routed in the up direction after being routed in any down direction.

Once the Up*/Down* routing is defined, UDFlex will allocate the job in a sub-tree graph as long as there is enough free contiguous nodes. Solheim et al. [9]

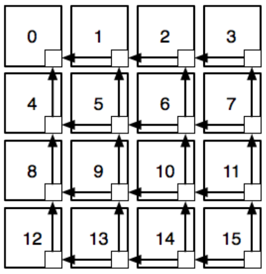


Fig. 3. Up*/Down* routing on a 4 × 4 mesh

define two main aims to describe the strategy of UDFlex. UDFlex aims to find the smallest valid Up*/Down* sub-tree graph in a breadth-first search strategy. In the event of a tie, the algorithm chooses the shallowest sub-root relative to the general root of the routing tree to keep that area as free and defragmented as possible. In Figure 4 there are two areas where we can allocate a 4-core job, but the area at nodes 18, 19, 23, and 24, is the one finally chosen, far from the root, which is placed at node 0.

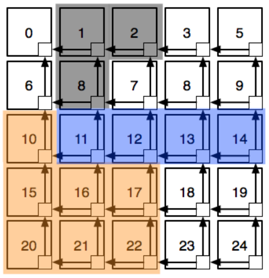


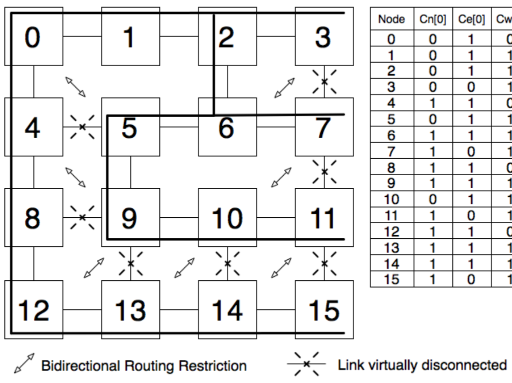
Fig. 4. Placing a 4-core application in UDFlex

UDFlex presents better flexibility for the allocation of different shapes, mostly irregular, to minimise fragmentation. The presence of irregular application domains requires routing mechanisms able to perform on such topologies, like FDOR or LBDR. In order to get better chip utilisation in the design trade-off, we need to aim for routing mechanisms that can display better coverage, coupled with the allocation algorithms. In [9], the authors state that UDFlex presents a complexity of $O(n^2)$, where n is the size of the network.

Although UDFlex allows for different tree formations, depending on which node is configured as a root, it still relies on Up*/Down* routing. One of the problems with this routing algorithm is that most of the traffic tends to focus on the root, which can lead to traffic imbalance and congestion problems. Segment-based Routing (SR) [5], for example, displays better performance than Up*/Down*. We need an allocation algorithm that supports other routing algorithms.

3.3 TSB

In this section we present Tree Segment-Based (TSB), our novel allocation algorithm. TSB is based on the idea of the tree allocation of SBBM [3]. SBBM is a NoC tree-based broadcast routing mechanism that is based on the configuration bits of the LBDR-based routing mechanisms. In SBBM, the switches are able to perform a broadcast operation, ensuring that each one of the nodes receives only one copy of the packet, and they are routed along the allocated tree. The LBDR configuration bits show a flexible routing implementation which can encode different routing algorithms, not only Up*/Down*. The formation of the tree in TSB (as in SBBM) is the result of how these bits are configured, based on the routing restrictions of the routing algorithm, so the designers can aim for different algorithms that display better routing performance or traffic balance, as long as the routing algorithm encoded contains also the property of deadlock-freedom.



0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Bidirectional Routing Restriction
Link virtually disconnected

Node	Cn[0]	Ce[0]	Cw[0]	Cs[0]	Cn[1]	Ce[1]	Cw[1]	Cs[1]
0	0	1	0	1	0	1	0	1
1	0	1	1	0	0	1	1	0
2	0	1	1	1	0	1	1	1
3	0	0	1	1	0	0	1	0
4	1	1	0	1	1	0	0	1
5	0	1	1	1	0	1	0	1
6	1	1	1	0	1	1	1	0
7	1	0	1	1	0	0	1	0
8	1	1	0	1	1	0	0	1
9	1	1	1	1	1	1	0	0
10	0	1	1	1	0	1	1	0
11	1	0	1	1	0	0	1	0
12	1	1	0	0	1	1	0	0
13	1	1	1	0	0	1	1	0
14	1	1	1	0	0	1	1	0
15	1	0	1	0	0	0	1	0

Fig. 5. Forming the TSB tree, change in connectivity bits

In Figure 5 the formation of the tree in TSB is shown. A routing algorithm is encoded in the routing and connectivity configuration bits. The routing bits display information about the routing restrictions and the connectivity bits present the connectivity of the links at each switch. In order to form the tree, the first phase is to produce a copy of the connectivity bits. The algorithm then proceeds to perform a *virtual disconnection pattern* of the links that share the directions of a routing restriction. You can see an example in Figure 6. The rules are simple, proceed to virtually disconnect (reflected by the connectivity bits) one of the links associated by a routing restriction, and in the case that a link is shared by two restrictions, disconnect that one. Although the disconnection can be done randomly, we aim for a balanced wide tree for maximising the bisection bandwidth.

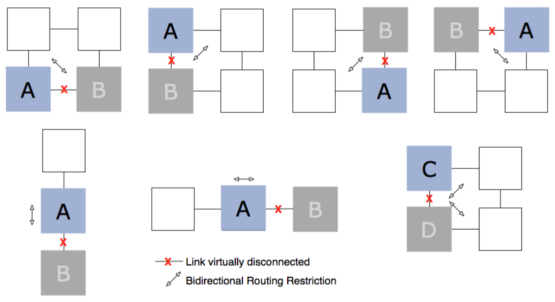

A
B
C
D
Link virtually disconnected
Bidirectional Routing Restriction

Fig. 6. Disconnection pattern

The second phase of the allocation algorithm, once we have the tree formed, and unlike UDFlex, is to mark the *leaves* of the three and store them in a list. In order to identify a *leaf* node, the algorithm performs an AND logic on the two sets of connectivity bits on each possible direction (north, east, south and west). If one of the resulting bits is set to one, then we proceed to mark the node as a *leaf*. Notice that the first and second phases are only performed each time the configuration bits are changed, meaning the routing algorithm has changed.

The next phase is to allocate the job choosing from the list of *leaves*. Starting from a *leaf*, the algorithm follows the tree, checking if the nodes are free. If the allocation scheme finds enough resources for the application, then the nodes are marked as busy. As soon as we allocate a job, in order to ensure traffic containment, the connectivity bits in the original layer perform their duty to ensure traffic isolation. The new *leaves* are marked for future jobs, and once the job is deallocated, a change in the connectivity bits of the free nodes starts another *leaf* marking phase. In Figure 7 a tree has been formed upon a given routing algorithm. Nodes 0, 4, 11, 12, 13 and 15 are marked as *leaves*. A job is placed in nodes 10, 13, 14, and 15. Then, node 6 is marked as a new *leaf*.

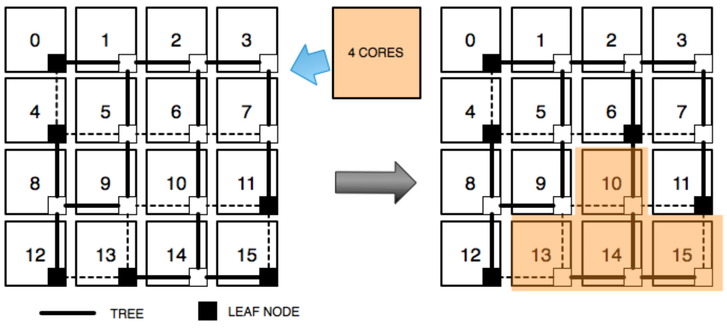


Fig. 7. Result of allocating a job, and new leaves

Notice that in the allocation scheme we are following the tree structure, but we can use another scheme that actually follows the original layer of connectivity bits. This is possible because the formation of the tree is still a subset of the original routing algorithm configuration (the first layer of connectivity bits), and therefore valid for routing purposes inside the application domain, so is up to the designer choose which scheme to follow.

Regarding the complexity of TSB, the main component is searching through the list of *leaf* nodes. As we mentioned before, we aim for a balanced tree as possible. In the conservative case, searching through the list of nodes has a complexity of the $applicationsize \times O(\log_2 n)$, which is $O(n \times \log_2 n)$, where n is the size of the network.

4 Evaluation

In this section we describe our evaluation framework. Our simulation framework is based on GEM5 [4], which allows to run a CMP-like scenario with the applications in the SPLASH-2 benchmark [13]. A FCFS (first come, first served) scheduler, which implements the different allocation algorithms, forming a hypervisor is the module who manages the allocation mechanism for the evaluation platform. We performed the evaluations on a 10×10 2-D mesh. The results are averaged from 10 uniform distributions where each application requires randomly 3, 6, 9, or 12 cores for the execution.

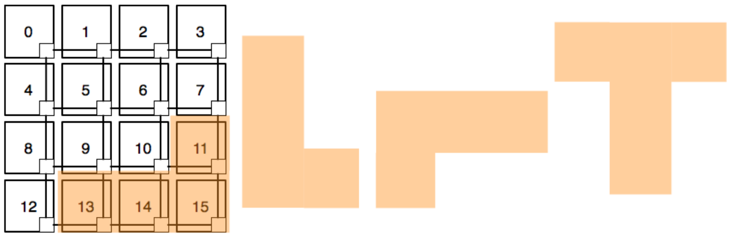


Fig. 8. Irregular partitions

In order to reflect the impact of the flexibility of irregular partitions for the allocation task, we evaluated two different scenarios. In the first scenario, we restricted the pool of allowed partitions to rectangular sub-meshes to each one of the allocation algorithms. In the second scenario, we allowed for UDFLEX and TSB to use irregular partitions where FDOR or LBDR can perform routing, like in Figure 8.

In Figure 9 the results for the first scenario are shown. As expected, restricting UDFLEX and TSB to rectangular sub-meshes results in that they perform as well as Best Fit in terms of utilisation. First Fit delivers lower utilisation than Best Fit due to not considering the smallest region to allocate and Random

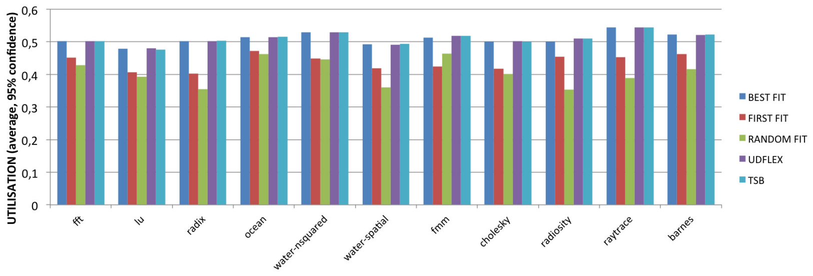


Fig. 9. Rectangular sub-meshes scenario

Fit gets the lowest results, indicating that a smart allocation delivers better performance.

Now, in Figure 10 we see the results for the second scenario. Here UDFLEX and TSB take the advantage of more flexible irregular regions and thus, they minimise the total fragmentation resulting in better utilisation of the chip. They are able to achieve 60% utilisation of the total chip in average.

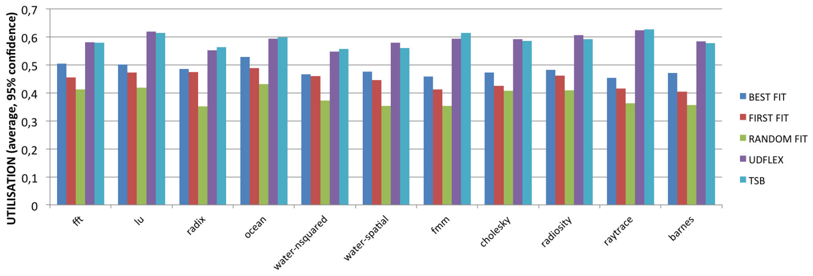


Fig. 10. Scenario with more flexible irregular partitions

5 Conclusion

We have studied the effects of more flexible partitioning when allocating different parallel applications in a CMP. Being restricted to the use of rectangular sub-meshes results in a poor chip utilisation. In this paper, we have contributed with a new idea, TSB, for an allocation algorithm that allows for flexibility in the routing layer, and performs as good as UDFlex, but with support for other routing algorithms.

Future work includes a thorough analysis of more metrics, like queuing and service time of the different allocation algorithms, application performance, overall system utilisation, plus a further evaluation of the impact with near-convex partitions, not just semi-irregular ones.

References

1. Intel Corp. The Single-chip Cloud Computer, `http://techresearch.intel.com/ResearchAreaDetails.aspx?Id=27`
2. Flich, J., Rodrigo, S., Duato, J., Sodring, T., Solheim, A.G., Skeie, T., Lysne, O.: On the potential of noc virtualization for multicore chips. In: International Conference on Complex, Intelligent and Software Intensive Systems, CISIS 2008, pp. 801–807. IEEE (2008)
3. Flich, J., Bertozzi, D.: Designing Network On-Chip Architectures in the Nanoscale Era. Chapman & Hall/CRC (2010)
4. Gem5. The gem5 simulator system, `http://gem5.org`
5. Mejía, A., Flich, J., Duato, J., Reinemo, S.A., Skeie, T.: Segment-based routing: An efficient fault-tolerant routing algorithm for meshes and tori. In: International Parallel and Distributed Processing Symposium, p. 84 (2006)
6. Rodrigo, S., Flich, J., Roca, A., Medardoni, S., Bertozzi, D., Camacho, J., Silla, F., Duato, J.: Addressing manufacturing challenges with cost-efficient fault tolerant routing. In: NOCS 2010: Proceedings of the 4th ACM/IEEE International Symposium on Networks-on-Chip, pp. 25–32 (2010)
7. Sem-Jacobsen, F.O., Rodrigo, S., Skeie, T., Strano, A., Bertozzi, D.: An efficient, low-cost routing framework for convex mesh partitions to support virtualisation. ACM TECS Special Issue on OnChip and OffChip Network Architectures (2011)
8. Skeie, T., Sem-Jacobsen, F.O., Rodrigo, S., Flich, J., Bertozzi, D., Simone, M.: Flexible DOR Routing for Virtualization of Multicore Chips. In: International Symposium on System-on-Chip (2009)
9. Solheim, Å.G., Lysne, O., Sødring, T., Skeie, T., Libak, J.A.: Routing-Contained Virtualization Based on Up*/Down* Forwarding. In: Aluru, S., Parashar, M., Badrinath, R., Prasanna, V.K. (eds.) HiPC 2007. LNCS, vol. 4873, pp. 500–513. Springer, Heidelberg (2007)
10. Triviño, F., Alfaro, F.J., Flich, J., et al.: NoC Reconfiguration for CMP Virtualization. In: 2011 10th IEEE International Symposium on Network Computing and Applications, NCA, pp. 219–222. IEEE (2011)
11. Triviño, F., Sánchez, J.L., Alfaro, F.J., Flich, J.: Virtualizing network-on-chip resources in chip-multiprocessors. Microprocessors and Microsystems, 1–16 (October 2010)
12. Triviño, F., Sánchez, J.L., Alfaro, F.J., Flich, J.: Exploring noc virtualization alternatives in cmps. In: 2012 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP, pp. 473–482. IEEE (2012)
13. Woo, S.C., Ohara, M., Torrie, E., Singh, J.P., Gupta, A.: The SPLASH-2 programs: characterization and methodological considerations. In: ISCA 1995: Proceedings of the 22nd Annual International Symposium on Computer Architecture, pp. 24–36. ACM, New York (1995)
14. Zhu, Y.: Efficient processor allocation strategies for mesh-connected parallel computers. Journal of Parallel and Distributed Computing 16, 328–337 (1992)