View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Enlighten: Publications

Al Khanjari, S., and Vanderbauwhede, W. (2016) Evaluation of the Memory Communication Traffic in a Hierarchical Cache Model for Massively-Manycore Processors. In: 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP), Heraklion Crete, Greece, 17-19 Feb 2016, pp. 726-733. ISBN 9781467387767.

There may be differences between this version and the published version. You are advised to consult the publisher's version if you wish to cite from it.

http://eprints.gla.ac.uk/137276/

Deposited on: 20 February 2017

Enlighten – Research publications by members of the University of Glasgow
http://eprints.gla.ac.uk

Evaluation of the Memory Communication Traffic in a Hierarchical Cache Model for Massively-Manycore Processors

Sharifa Al Khanjari
School of Computing Science
University of Glasgow
Glasgow, UK
Email: s.al-khanjari.1@research.gla.ac.uk

Wim Vanderbauwhede
School of Computing Science
University of Glasgow
Glasgow, UK
Email: wim.vanderbauwhede@glasgow.ac.uk

***Abstract*—The scaling of semiconductor technologies is leading to processors with increasing numbers of cores. A key enabler in manycore systems is the use of Networks-on-Chip (NoC) as a global communication mechanism. The adoption of NoCs in manycore systems requires a shift in focus from computation to communication, as communication is fast becoming the dominant factor in processor performance. Many researchers have focused on direct communication between cores in the NoC; however in a manycore processor the communication is actually between the cores and the memory hierarchy. In this work, we investigate the memory communication traffic of shared threads in a hierarchical cache architecture.**

We argue that the performance scalability for shared-memory applications in a hierarchical cache architecture for systems with thousands of processor cores depends on the distance between threads sharing memory in terms of the cache hierarchy (the "memory distance"). We present latency and throughput results comparing fat quadtree, concentrated mesh and mesh topologies as a function of the "memory distance" between the threads. Our results using the ITRS physical data for 2023 show that the model of thread placement and the distance of placing them significantly affects the NoC performance, and that scale-invariant topologies perform better than flat topologies.

***Index Terms*—Manycore, Shared-Memory Architecture, Network on Chip, Quadtree.**

I. Introduction

If the semiconductor industry can maintain scaling according to Moore's law, then in the next decade, multiprocessor systems on chip will contain hundreds to thousands of cores. Such a massively-manycore system requires high performance interconnections to transfer data between the cores on the chip. Already, for manycore processors with close to 100 cores such as the Tilera Tile64 [1] or the Intel MIC [2], Networks-on-chip (NoC) have become the preferred on-chip communication infrastructure, and for processors with larger numbers of cores, NoCs constitute the only option. Performance of NoC-based manycore systems is highly dependent on the traffic patterns and the NoC topologies: in manycore systems communication, not computation, is the performance-limiting factor. In a NoC-based manycore system with global shared memory, the communication is between the cores and the memory. Many NoC researchers have focused on point-to-point traffic, implicitly assuming the system does not provide global shared memory. These are two different traffic patterns and the resulting NoC performance is very different, as is shown by comparing the results in [3] to the results in this work.

To make effective use of the resources provided by manycore processors, parallel programming is essential. As a result, applications have increasing numbers of threads, and thread placement and inter-thread communication have become important topics of research. Besides, as the number of cores increases, memory locality will become even more important. The main focus of our research is the effect of memory locality on the placement of threads sharing memory, and the network performance resulting from different topologies and locality models.

In the presence of caches, the traffic is determined by both the location of the cache with respect to the thread and by the distribution of data over the cache hierarchy. For example, in a distributed cache system with hashing such as used by the Tilera TilePro, the average distance from each core to the cache is constant; in the MIC architecture, the L3 cache is the union of the L2 caches for each core, and the ring topology of the NoC means the memory distance between two threads that are located on physically adjacent cores can be up to half the total number of cores. Neither of these architectures can scale to thousands of cores.

In order to investigate the effect of memory locality on performance, we propose abstract models of placing threads in shared-memory applications with different distance metrics. These models let us control the distance between the threads as well as the shape of the local area, and thus provide general insights into the suitability of a given topology for a given shared-memory thread placement model.

Our assumption is that the future manycore architecture will have a distributed memory architecture, with memory embedded in the processor tiles or stacked on top of them (if not physically then at least logically). This assumption is supported by the advent of embedded DRAM and 3D memories [4],[5] and the deployment of 3D memory in e.g. Intel's 'Knights Landing' Xeon Phi. Thus we propose a

hierarchical cache architecture model and the allocation of the hierarchical cache in three topologies.

A crucial claim in our work is that in order to optimise performance, the communication patterns in multi-threaded shared-memory programs will need to exhibit locality with respect to the memory hierarchy, because the thread scheduler will consider both the load of the cores and the amount of data sharing when placing the threads, see e.g. [6, 7, 8]. Our paper shows that enforcing locality of access will improve the latency.

The remainder of this paper is organized as follows. In Section II we present our thread placement locality models. In Section III we show that cache coherency traffic will not affect the over all performance of the network. Moreover, the tracking sharers will have a small overhead on the cache size. In Section IV we describe our proposed hierarchical cache model and the topologies used. In Section V we detail the cost model of the topologies, the links and buffers per virtual channel used for each topology, the technology node assumptions used in this work, and the overheads for a 1024 core chip. In Section VI we describe evaluation methodologies and present results and analysis. We conclude in Section VII.

II. Thread Placement Locality Models

We propose a simple level-based model for locality of placement of threads sharing a memory address. To model locality, we group the cores of the chip and create hierarchical groups to encompass the whole system. Using $0 < l \leq n$ for the levels of the hierarchy, we can express the probability for communication across level-l as:

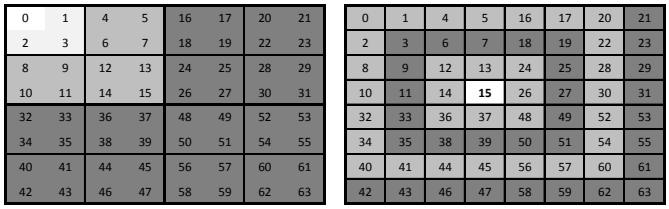
$$p(l) = (1-\alpha)\alpha^{l-1}, 1 \leq l < n \qquad (1)$$
$$p(n) = \alpha^{n+1}$$

The parameter α relates to the locality of placing strongly coupled threads. It expresses the probability that a memory request has to travel a certain distance in the hierarchy. Lower α means higher locality: if $\alpha = 0.2$, then according to equation (1) 20% of the memory requests will have a shared memory address with a thread in the first level cache, and 20% of that portion in the second level cache, etc. When $\alpha = 1$, it means that most of the memory requests will have a shared memory address in the last level cache.

In this paper, we use two different instances of locality-based shared-memory thread placement models to evaluate the performance of the NoC topologies. These are abstract models representing different types of thread placement for different applications.

- In the first model (*Group Clustering*), we place the threads in cores of the chip in fours, and create hierarchical groups to encompass the whole system, in a scale-invariant fashion. In this case $n = log_4(N)$ with N the number of cores, and each level contains 4^l cores. The group clustering locality captures the physical hierarchical caches architecture. Figure 1.a shows an example of group clustering of threads where the requesting thread is on core 0. This is a generalisation of the memory access pattern resulting from e.g. tree-based reduction of values from all threads.
- In the second model (*Ring Clustering*), we place the threads in cores of the chip in concentric rings around the sender core. In this case the number of cores per level is $8l$ as long as the rings don't meet the edge. This is a generalisation of nearest-neighbour (stencil) traffic. Figure 1.b shows an example of ring clustering of threads where the requested thread is on core 15. This model is a generalisation of nearest-neighbour stencils typical for many finite-difference based applications such as encountered in e.g. weather prediction or computational fluid dynamics.

The core will generate a memory request to a cache that is shared with a thread, which is selected based on the clustering model. The memory request will result in a cache miss until it reaches the shared cache between the threads. The shared cache will then return a cache line back to the requested thread. All the memory requests generated by the cores are to a shared memory address with other threads, which is the worst case scenario.



a) Group clustering b) Ring clustering

Figure 1. Example of thread placement models in 8x8 mesh

III. Cache Coherency and Tracking Sharers

In their seminal paper "Why on-chip cache coherency is here to stay", Martin et al. [9] discuss how on-chip coherence can scale as the number of cores increases. The conventional wisdom has been that cache coherence could not scale because of exploding storage and interconnection network traffic requirements and concerns over latency and energy consumption. Before investigating the issues involved in future for coherence, state-of-the-art cache coherence mechanisms were examined, and found to be more scalable than expected. Five potential concerns were examined when scaling on-chip coherence. First, regarding the traffic on the on-chip interconnection network, they showed that it scales when precisely tracking sharers. Secondly, regarding storage costs for tracking sharers, they showed that a hierarchy combined with inclusion enables efficient scaling of the storage cost for exact encoding of sharers. Thirdly, for inefficiencies caused by maintaining inclusion, they found that using chip architects to design a system with an inclusive shared cache with negligible recall rate can efficiently embed the tracking state in the shared cache. Fourthly, latency of cache misses was shown to be tolerable, as misses to actively shared blocks have

greater latency than other misses. Finally, the energy overhead analysis shows that based on the traffic and storage scalability analyses, the energy overhead of coherence will not increase with the number of cores.

In our paper, building on these arguments, we have assumed a shared-memory model of computation. In order to maintain a global shared memory caches are essential. Based on the findings of Martin et al. [9] we can assume that the traffic overhead of the cache coherency traffic will be negligible. Furthermore, [10] showed that the amount of application data in the NoC is much larger than the amount of cache coherence data for almost all cache sizes. Our proposed architecture provides a good baseline for a hierarchical cache architecture with cache coherency. Using the directory based protocol and the tracking protocol on our hierarchical cache architecture would only require a 1.5% storage overhead on the cache size.

IV. Hierarchical Cache Model

3D memory stacking has received great attention in recent years [4, 11], since it resolves the memory bandwidth challenges of 2D integration. The latest Xeon Phi (Knights Landing) already uses a hybrid form of stacked memory, Micron's Hybrid Memory Cube. 3D memory allows each processor core to have fast and high bandwidth access to the cache banks directly stacked on top of it using very dense vertical interconnects. The processor core can also access cache banks stacked on the other processors [12],[13]. The advantages of 3D stacking are lower latency and higher bandwidth. We propose a 3D stacked cache memory hierarchy that has a similar concept to the memory hierarchies of the traditional multiprocessors. The stacked cache can be either private or shared [14]. Some research focuses on the techniques for DRAM caches architecture [15], however, our focus is on the network communications between the cores and the memory.

We present a hierarchical cache architecture for three different network topologies namely mesh, concentrated mesh and fat quadtree, introduced in the next sections. The caches are stacked on top of one another. Because of their inherent scalability, hierarchical cache architectures are becoming an interesting alternative for manycore systems. Grouping cores and their caches in clusters reduces network congestion by localizing traffic among several hierarchical levels, potentially enabling much higher scalability. In our proposed architecture, every four cores are clustered and share a level L2 cache, each 4×4 cores are clustered and share a level L3 cache and so on. Using $0 < l \leq n$ for the levels of the cache, and assuming $n = log_4(N)$ with N the number of cores, figure 2 shows the proposed hierarchical cache architecture. L1 caches are embedded in the cores. C1, C2 etc. are the individual cores.

A. Mesh

The mesh topology has been the most popular NoC topology so far and it has been used in most of the recent manycore chips such as the Intel SCC (48 cores) [16], TFlops (80 cores) [17], Tilera (64 cores) [1]. It organises the routers in a grid, one router per core. The mesh has a radix (number of ports) of 5. Deadlock is avoided by using a deadlock free routing algorithm, e.g. XY routing.

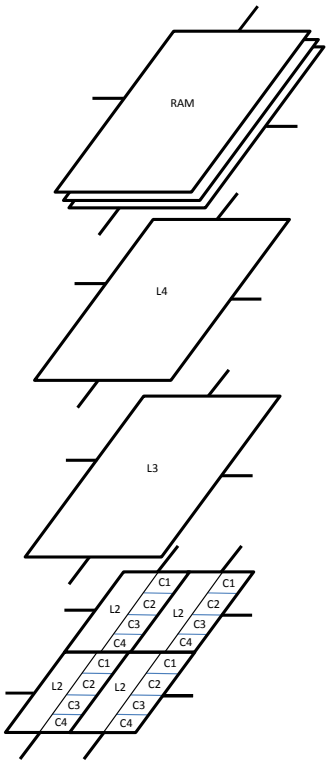


Figure 2. The proposed 3D stacked memory ($N = 256$)

The allocation of the memory controllers for the hierarchical caches were placed as near as possible to the centre of the cores that share the cache. A router can only represent a single cache level. Figure 3 shows an example of the hierarchical cache allocation for 8×8 mesh. The core can calculate the location of all its hierarchical caches using the following equation, where $\setminus$ represents integer division.

$$4^{l-1}(coreID \setminus 4^{l-1}) + 4^{l-2} - 1, 1 \leq l < n \qquad (2)$$

Our novel Mesh Memory Routing (MMR) is a routing mechanism where the routing decision is made at the routers. When a memory request arrives at a cache router, the cache router will check if this cache is shared between the requester core and the other thread which was obtained using group or ring clustering. If the cache router is shared (cache hit), then a cache line will be sent to requesting core. If the cache router is not shared (cache miss), the request will be sent to the cache level above. The requests are routed through the network between the caches following XY routing. If the request reaches the highest level router (DRAM), then a memory line will be returned to the requesting core passing through all the caches that the request came from.

Figure 3 shows an example for a ring clustering case (core 27 sending a memory request to a shared memory with cores 18 or 19 or 52 and back). It illustrates the path that a request from core 27 has to pass through if it shares memory with

cores 18, 19 and 52. The requester has to go to L2 (router 21) then L3 (router 19) then L4 (router 15), then pass back through all the caches to the requester core 27.

B. Concentrated Mesh

The concentrated mesh (Cmesh) has been introduced by [18] to preserve the advantages of a mesh with decreased diameter. The number of cores sharing a router is called the concentration degree of the network (4 in this work). The Cmesh topology requires fewer routers resulting in reduced hop count and consequently improved latency. It has a radix of 8.

The cache placement of the Cmesh is similar to the mesh but the Cmesh has one less level of cache. In a Cmesh each router represents L2 and might represent another level of cache. The memory traffic routing is similar to the mesh.

C. Fat Quadtree

The fat tree connects routers in a tree with the cores at the leaves. To avoid congestion towards the root of the tree, a fat tree uses an increasing number of links as described in [19]. A fat quadtree of size N is a structure that can be regarded as a rooted 4-ary tree of height $log_4(N)$. In a way this exactly reflects the group clustering model.

The quadtree topology reflects our hierarchical cache architecture perfectly, as the leaves represent the cores and the nodes represent the caches. Similarly, four cores share a L2 cache, four L2 caches share a L3 cache ... etc.

Fat Quadtree Memory Routing (FQMR) is a routing mechanism. When a memory request arrives at a cache router, the cache router will check if this cache is shared between the requester and the other thread which was obtained using group of ring clustering. If the cache router is shared (cache hit), then a cache line will be sent to the requesting core. If the cache router is not shared (cache miss), the request will be sent to the parent which is the cache level above. The requests are routed through the network between the caches following nearest-common ancestor routing. If the request reaches the highest level router (DRAM), then a memory line will be returned to the requesting core passing through all the caches that the request came from.

V. NoC Topologies Overhead

The overhead imposed by the NoC is potentially an important factor to take into account when selecting a topology, because the cost of the die is proportional to its area. Therefore we have created analytical models for the overhead for the three topologies as a function of the number of cores in the chip, the number of buffers and the number of virtual channels.

A. Cost Model

We present the cost model of mesh, Cmesh and fat quadtree in terms of link complexity, number of routers and buffers. Table I shows the notations used for the cost model.

Link Complexity is the total number of links in the topology. In section V-C, we will compute the wire overhead for mesh, Cmesh and fat quadtree. The total number of buffers in mesh and Cmesh are straightforward as in each router there are 5 and 8 ports, respectively. The total number of buffers in fat quadtree is more complex since the buffer size doubles at every level because the wire lengths are doubling at every level. Table II shows the cost model for 1024 cores.

Table I
TABLE OF NOTATIONS

Symbol	Description	Symbol	Description
N	Number cores	N_L	Number of links
n_B	Number of buffers	N_R	Number of routers
n_{VC}	Number of virtual channels	N_B	Number of buffers

Table II
COST MODEL FOR 1024 CORES

Mesh	
N_L	$2\sqrt{N}(\sqrt{N}-1).n_{VC}$
N_R	N
N_B	$5n_Bn_{VC}N$
Fat Quadtree	
N_L	$N.log_4(N)$
N_R	$\frac{N-1}{3}$
N_B	$n_B(N-4)(\sqrt{N}-2)+2n_B\sqrt{N}$
Concentrated Mesh	
N_L	$\sqrt{N}(\frac{\sqrt{N}}{2}-1).n_{VC}$
N_R	$\frac{N}{4}$
N_B	$2n_Bn_{VC}N$

To calculate the wire link overhead, we get the links width $Link_{width}$ as in equation 3, where (W) is the wire pitch, (N_{bits}) is the number of bits in parallel for one packet and $N_{layers}$is the number of layers.

$$Link_{width} = \frac{W \times N_{bits}}{N_{layers}} \tag{3}$$

The number of vertical wires in a fat quadtree can be obtained $Vertical_{wire} = 2^{(log_4N-1)}log_4N$, and for the mesh $Vertical_{wire} = \sqrt{N}$ where N is the number of cores. Starting from Eq. 3 and the number of vertical wires, we can compute the area overheads as follows:

$$Area_{Wire} = 2 \times Width_{VerticalLink} \times Width_{Chip} \tag{4}$$

$$Area_{Chip} = Area_{Core} \times N + Area_{Wire} \tag{5}$$

$$Area_{Overhead} = \frac{Area_{Wire}}{Area_{Chip}} \tag{6}$$

B. Links and Buffers per Virtual Channel

In the memory traffic model, the flits are required to travel between the caches. The fat quadtree reflects the hierarchical cache architecture perfectly, hence all the links will be utilised. However, in a mesh and Cmesh some of the links won't be used at all. Figure 4 shows an example in an 8×8 mesh, where 12.5% of the total links are not used. With mesh and Cmesh more resources (VC and number of buffers) were, however, assigned to them to compensate for unused resources. Mesh and Cmesh will congest faster compared to the fat quadtree

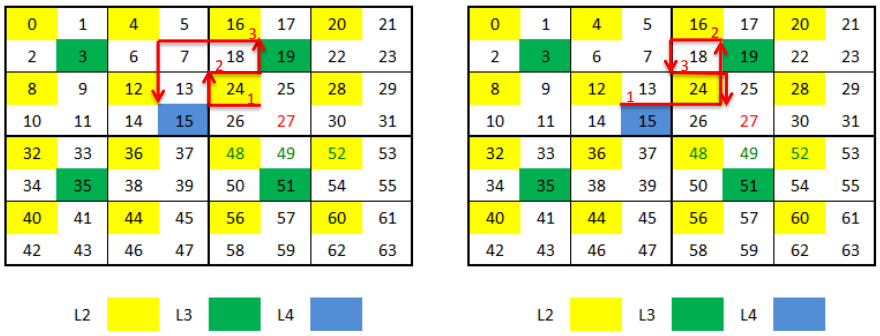


Figure 3. Hierarchical cache allocation for 8×8 mesh

if the resources are not equally distributed because the cache router and the links between the caches will congest faster.

The fat quadtree has 5120 links while the mesh has 1984 links in a 32×32 network and 22% of those links are not used. Hence, 4 VC were assigned to the mesh to compensate for the unused links, to give 6188 utilised links. The Cmesh has 480 links with 16% not used links. Therefore, 14 VC were assigned to the Cmesh to give 5600 links. Now the mesh and the Cmesh have more links than the fat quadtree. Similarly the number of buffers in a virtual channel in a mesh and Cmesh are now 48 flit/VC and 24 flit/VC, respectively.

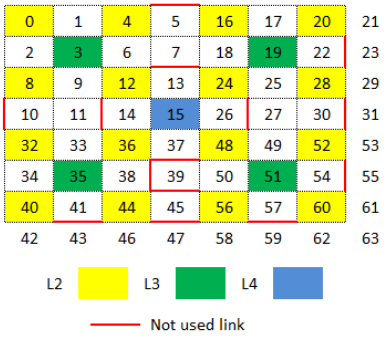


Figure 4. 12.5%of the links in an 8×8 mesh are not used

C. Technology Node Assumptions

To ensure realistic simulations of the next decade's manycore systems, we assume the $10nm$ process in 2023 as projected by the International Technology Roadmap for Semiconductors. Table III lists the physical parameters for this technology node from the 2011 ITRS data. We used the die size of the 64-core Tile64 from [20] ($433.5\,mm^2$) to estimate the core size and scaled it to the 2023 node. From ITRS [21] the chip size at production in 2013 was $140mm^2$ so it is less than the estimated core size by a factor of $3.1(434/48 = 140)$. In 2023, the chip will contain $20\times$ more cores than today's chip. Consequently, the chip size in 2023 with nearly 1280 (20×64) cores will be approximately $3.1 \times 111 = 344.1\,mm^2$. Hence, the area of one core is $344.1/1280 = 0.3\,mm^2$. This area corresponds to dimensions of about $0.5\,mm \times 0.5\,mm$. The width of one core is thus $0.5\,mm$ and the wire delay can be estimated at $0.5 \times 33827/1000 = 17.5\,ns$.

Table III
TECHNOLOGY PARAMETERS

Year	2013	2023
Chip size at production [21]	140 mm^2	111 mm^2
Global wire delay [21]	999 ps	33827 ps
Estimated core size	$6.8mm^2$	$0.3mm^2$
Estimated wire delay	$2.6ns$	17.5 ns

D. Overheads for a 1024-core chip, 10nm (2023) ITRS node

In terms of buffer space overhead, the mesh will require $5.1\,KB$ of storage per core, the Cmesh $2\,KB$ and the fat quadtree $15.3\,KB$. Although the total number of buffers is a lot more in fat quadtree than mesh, it is only a very small fraction of the total size of the chip: e.g. the per-core L2 cache on the 60-core Xeon Phi is already $512\,KB$. With the above assumptions, the area of a $15.3\,KB$ SRAM buffer would be 0.14% of the estimated core size (memory density $37.6\,MB/mm^2$). In terms of wire overhead, our cost model shows that the wire area overhead for the fat quadtree would be 0.3% of the estimated chip size for a 1024-core chip (wire pitch $17\,nm$).

These results are very important as they indicate that for this type of manycore architecture, the NoC overhead is negligible, which means that the choice of the NoC can be based solely on performance.

VI. SIMULATION RESULTS AND DISCUSSION

The simulation was implemented using the HNOCS (Heterogeneous Network-on-Chip Simulator) package, which is an open source NoC simulator [22] based on OMNeT++ [23]. OMNeT++ is an extensible, modular, open source component based C++ simulation library and framework, primarily aimed at building network simulators. The original HNOCS uses a mesh topology with wormhole switching with virtual channels and XY routing. We extended HNOCS with the Cmesh and fat quadtree topologies and their memory routing algorithms, as well as our locality-based traffic distributions.

We model the process-to-cache communication using Poisson-distributed traffic because it typically offers a good

estimate on the average performance of networks and it has been widely used in both the evaluation of interconnection networks and in cache modelling, see e.g. [24].

The different topologies were simulated on 1024-core chip (cores placed in a regular 32×32 grid). Four virtual channels are used for the mesh and fourteen for Cmesh as explained in section V-B while for fat quadtree one physical channel is used for the lowest-level links and it quadruples at each level to simulate a fat quadtree. Hence, the fat quadtree has no virtual channels. The wire delay is proportional to the distance between the routers so in a fat quadtree it doubles at each level. The destination was selected using different degrees of localisation. Table IV summarises the simulation parameters used in our simulations.

In the simulation, all cores generate a memory request at the same time. A core will generate the next request only if the source queue is not full; hence there will be no dropped requests.

Table IV
SIMULATION PARAMETERS

Topology	Mesh	Cmesh	Fat quadtree
Number of virtual channels	4	14	0
Wire delay (ns)	17.5	35	$35 \times 2^{l-1}, 1 \leq l < n$
Flit size (bytes)	64		
Buffer size (flits/VC)	48	24	16
Channel datarate (Gb/s)	128		

We evaluated the performance of the three topologies as a function of the transmission rate and the miss rate α, where $\alpha = 1$ means that the threads are replaced the furthest and the main memory will be hit, and $\alpha = 0$ means the first level cache will be 100% hit. Figures 5 and 6 show the results of our experiments in terms of latency and throughput. Here, latency is the latency of the network, not taking into account the wait time of a packet at the transmitting core when the link is busy.

For group clustering, the fat quadtree performs best, as Figure 5 shows: the Cmesh and the fat quadtree have lower latencies when $(0 \leq \alpha \leq 0.1)$, then the latencies start to increase as α becomes higher. The Cmesh congests faster than the fat quadtree because it has fewer routers. The mesh has high latencies although it has more routers than both the fat quadtree and the Cmesh. This is because in memory traffic, where the requests travel between the caches, the path of the requests is deterministic hence some routers have more traffic while others might not receive any traffic. Moreover, most of the links that connect between the caches are nearly 100% utilized. Some of the cores do not receive any flits due to the congestion so we set the latency penalty for those cores to the simulation time $1\,ms$.

In terms of throughput, we observe that in nearly all the cases the throughput increases rapidly as the request rate increases. In case of $(0.5 \leq \alpha \leq 1)$ the throughput is lower because the latencies were high and networks are congested.

For ring clustering, the mesh and the Cmesh have high latencies as they congest faster and they perform worse than the fat quadtree. The fat quadtree has low latencies. This is because in the mesh and the Cmesh the memory request traffic always travels on the same path between the caches hence the network gets congested. In terms of throughput, the fat quadtree has higher throughput compared to the mesh and Cmesh. Overall the memory bandwidth of our system is $64\,TB/Sec$ compared to the $288\,GB/Sec$ and $352\,GB/Sec$ for present systems e.g. NVIDIA's Kepler K40 GPU and Intel's Xeon Phi 7120P respectively.

Figure 7 shows the latencies of the topologies when the network is not congested and packets are generated every $80\,ns$ for different α parameter. In both models, the fat quadtree performs best even when the threads are placed further apart. Overall, group clustering results in lower latencies than ring clustering; this is an important result for the placement of threads in group neighbours that match the cache architecture.

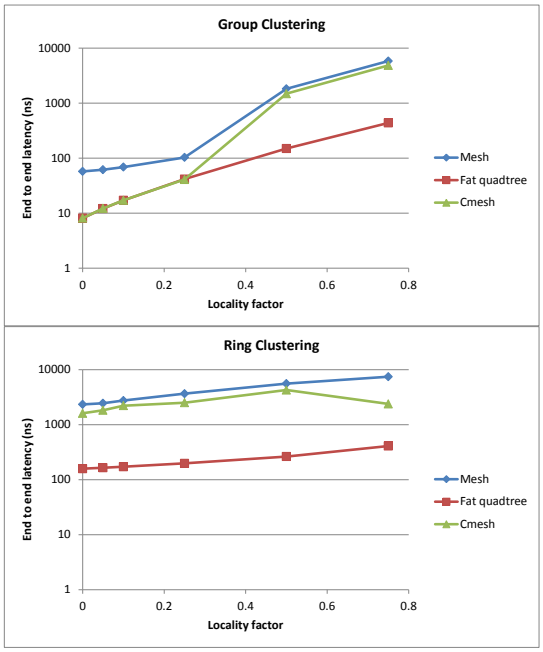


Figure 7. Group and Ring clustering

VII. CONCLUSION

We have investigated the overhead and performance of flat (mesh, Cmesh) and scale-invariant (fat quadtree) NoC topologies for future manycore systems with thousands of cores using two different models of locality for thread placement in shared-memory systems, group clustering and ring clustering. We show that the overhead of the NoC on a thousand-core system is negligible for all three topologies, so that the choice of topology depends solely on the performance. We show that the distance between the threads and the clustering model strongly affects the performance of the network. Scale-invariant topologies such as the fat quadtree perform better

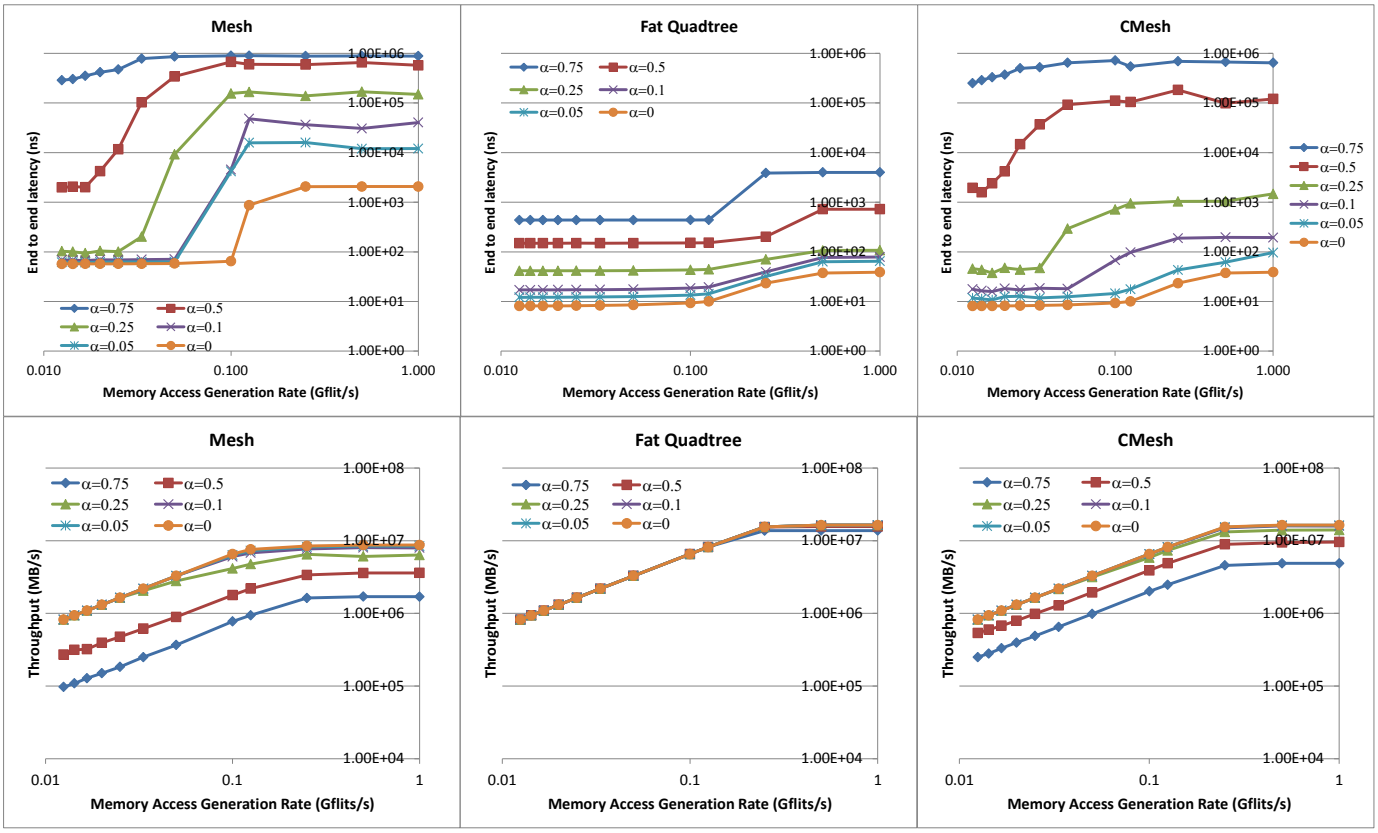


Figure 5. Group clustering results

than flat ones because their structure matches the hierarchical cache architecture. Our results clearly show the importance of thread placement locality for very large manycore systems.

References

[1] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, *et al.*, "Tile64-processor: A 64-core soc with mesh interconnect," in *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, pp. 88–598, IEEE, 2008.

[2] A. Duran and M. Klemm, "The intel® many integrated core architecture," in *High Performance Computing and Simulation (HPCS), 2012 International Conference on*, pp. 365–366, IEEE, 2012.

[3] S. Al Khanjari and W. Vanderbauwhede, "The impact of traffic localisation on the performance of nocs for very large manycore systems," *Procedia Computer Science*, vol. 56, pp. 403–408, 2015.

[4] G. H. Loh, "3d-stacked memory architectures for multi-core processors," in *ACM SIGARCH Computer Architecture News*, vol. 36, pp. 453–464, IEEE Computer Society, 2008.

[5] S. K. Lim, "3d-maps: 3d massively parallel processor with stacked memory," in *Design for High Performance, Low Power, and Reliable 3D Integrated Circuits*, pp. 537–560, Springer, 2013.

[6] F. Broquedis, N. Furmento, B. Goglin, R. Namyst, and P.-A. Wacrenier, "Dynamic task and data placement over numa architectures: an openmp runtime perspective," in *Evolving OpenMP in an Age of Extreme Parallelism*, pp. 79–92, Springer, 2009.

[7] D. Tam, R. Azimi, and M. Stumm, "Thread clustering: sharing-aware scheduling on smp-cmp-smt multiprocessors," in *ACM SIGOPS Operating Systems Review*, vol. 41, pp. 47–58, ACM, 2007.

[8] E. Z. Zhang, Y. Jiang, and X. Shen, "Does cache sharing on modern cmp matter to the performance of contemporary multithreaded programs?," in *ACM Sigplan Notices*, vol. 45, pp. 203–212, ACM, 2010.

[9] M. M. Martin, M. D. Hill, and D. J. Sorin, "Why on-chip cache coherence is here to stay," *Communications of the ACM*, vol. 55, no. 7, pp. 78–89, 2012.

[10] G. Girão, B. C. de Oliveira, R. Soares, and I. S. Silva, "Cache coherency communication cost in a noc-based mpsoc platform," in *Proceedings of the 20th annual conference on Integrated circuits and systems design*, pp. 288–293, ACM, 2007.

[11] S. K. Lim, *3D-MAPS: 3D massively parallel processor with stacked memory*, pp. 537–560. Springer, 2013.

[12] N. Madan, L. Zhao, N. Muralimanohar, A. Udipi, R. Balasubramonian, R. Iyer, S. Makineni, and D. Newell, "Optimizing communication and capacity in

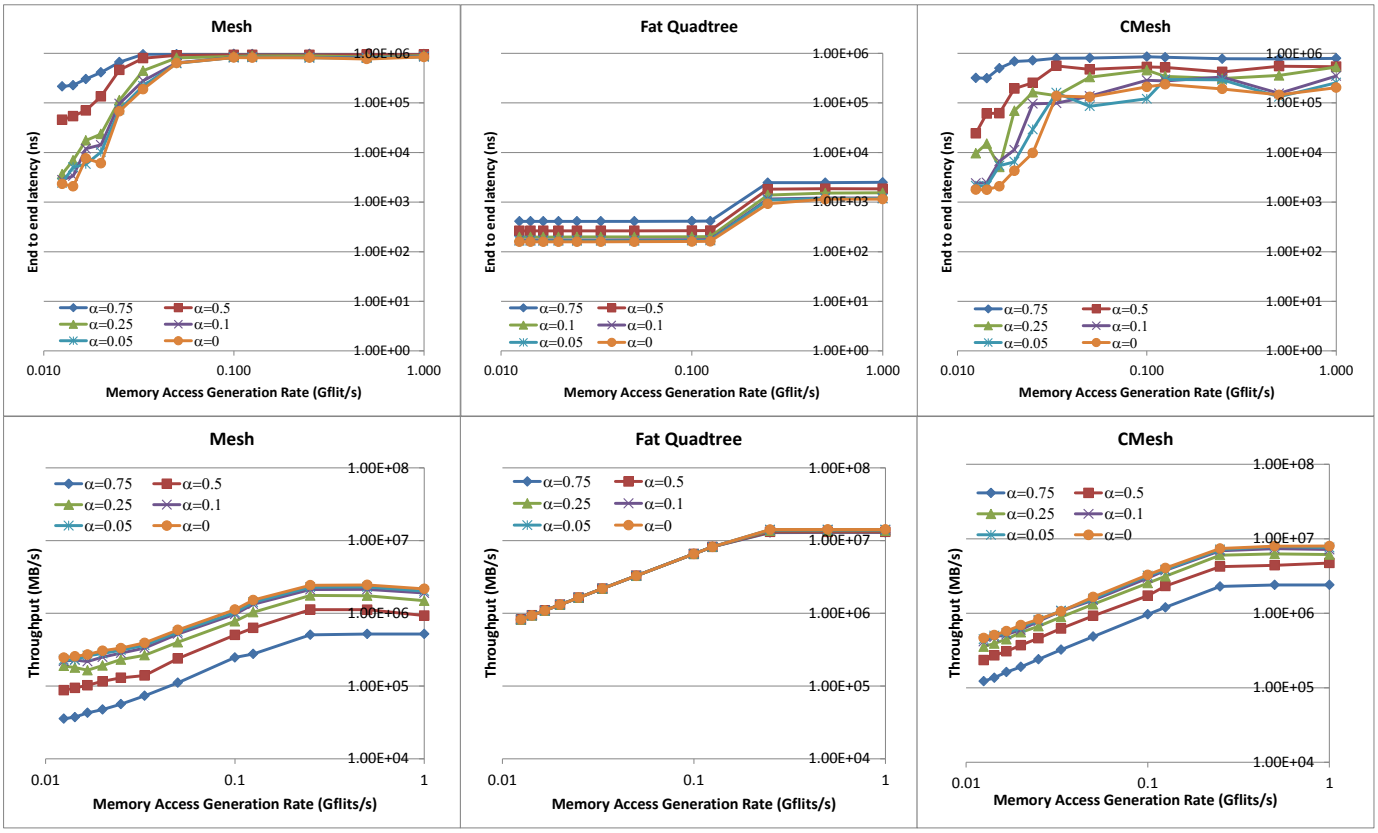


Figure 6. Ring clustering results

a 3d stacked reconfigurable cache hierarchy," in *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pp. 262–274, IEEE.

[13] A. Zia, P. Jacob, J.-W. Kim, M. Chu, R. P. Kraft, and J. F. McDonald, "A 3-d cache with ultra-wide data bus for 3-d processor-memory integration," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, no. 6, pp. 967–977, 2010.

[14] J. Jung, K. Kang, G. De Micheli, and C.-M. Kyung, "Runtime 3-d stacked cache management for chip-multiprocessors," 2013.

[15] S. Mittal and J. Vetter, "A survey of techniques for architecting dram caches," *Parallel and Distributed Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[16] J. Howard, S. Dighe, S. R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, *et al.*, "A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and dvfs for performance and power scaling," *Solid-State Circuits, IEEE Journal of*, vol. 46, no. 1, pp. 173–183, 2011.

[17] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, *et al.*, "An 80-tile 1.28 tflops network-on-chip in 65nm cmos," in *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, pp. 98–589, IEEE, 2007.

[18] J. Balfour and W. J. Dally, "Design tradeoffs for tiled cmp on-chip networks," in *Proceedings of the 20th annual international conference on Supercomputing*, pp. 187–198, ACM, 2006.

[19] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *Computers, IEEE Transactions on*, vol. 100, no. 10, pp. 892–901, 1985.

[20] C. Killebrew *et al.*, *L2 Cache to Off-chip Memory Networks for Chip Multiprocessor*. PhD thesis, Department of Electrical Engineering and Computer Sciences, University of California, 2008.

[21] "International technology roadmap for semiconductors (itrs)," 2011.

[22] Y. Ben-Itzhak, E. Zahavi, I. Cidon, and A. Kolodny, "Hnocs: Modular open-source simulator for heterogeneous nocs," in *Embedded Computer Systems (SAMOS), 2012 International Conference on*, pp. 51–57, IEEE, 2012.

[23] A. Varga *et al.*, "The omnet++ discrete event simulation system," in *Proceedings of the European Simulation Multiconference (ESM2001)*, vol. 9, p. 185, sn, 2001.

[24] C. M. Krishna, *Performance modeling for computer architects*, vol. 11. John Wiley & Sons, 1996.