A Greedy Heuristic Approximation Scheduling Algorithm for 3D Multicore Processors*

Thomas Canhao Xu, Pasi Liljeberg, and Hannu Tenhunen

Turku Center for Computer Science, Joukahaisenkatu 3-5 B, 20520, Turku, Finland
Department of Information Technology, University of Turku, 20014, Turku, Finland
{canxu,pasi.liljeberg,hannu.tenhunen}@utu.fi

Abstract. In this paper, we propose a greedy heuristic approximation scheduling algorithm for future multicore processors. It is expected that hundreds of cores will be integrated on a single chip, known as a Chip Multiprocessor (CMP). To reduce on-chip communication delay, 3D integration with Through Silicon Vias (TSVs) is introduced to replace the 2D counterpart. Multiple functional layers can be stacked in a 3D CMP. However, operating system process scheduling, one of the most important design issues for CMP systems, has not been well addressed for such a system. We define a model for future 3D CMPs, based on which a scheduling algorithm is proposed to reduce cache access latencies and the delay of inter process communications (IPC). We explore different scheduling possibilities and discuss the advantages and disadvantages of our algorithm. We present benchmark results using a cycle accurate full system simulator based on realistic workloads. Experiments show that under two workloads, the execution times of our scheduling in two configurations (2 and 4 threads) are reduced by 15.58% and 8.13% respectively, compared with the other schedulings. Our study provides a guideline for designing scheduling algorithms for 3D multicore processors.

1 Introduction

The number of circuits integrated on a chip have been increasing continuously which leads to an exponential rise in the complexity of their interaction. Traditional digital system design methods, e.g. bus-based System-on-Chip (SoC) will encounter communication bottlenecks. To address these problems, Network-on-Chip (NoC) was proposed as a promising communication platform solution for future multicore systems [1]. Network communication methodologies are brought into on-chip communication. Figure 1 shows a NoC with 4×4 mesh (16 nodes). The underlying network is comprised of network links and routers (R), each of which is connected to a processing element (PE) via a network interface (NI). The basic architectural unit of a NoC is a tile/node (N) which consists of a router, its attached NI and PE, and the corresponding links. Communication among PEs is achieved via network packets. Intel[1] has demonstrated an 80 tile,

* This work is supported by Academy of Finland and Nokia Foundation.

[1] Intel is a trademark or registered trademark of Intel or its subsidiaries. Other names and brands may be claimed as the property of others.

M. Alexander et al. (Eds.): Euro-Par 2011 Workshops, Part I, LNCS 7155, pp. 281–291, 2012.

100M transistor, 275mm^2 2D NoC under 65nm technology [2]. Recently, an experimental microprocessor containing 48 cores (x86) on a chip has been created, using 4×6 2D mesh topology with 2 cores per tile [2].

Traditional 2D chip interconnection will result long global wire lengths, causing a high delay, high power consumption and low performance [3]. Besides 2D chips have larger die size in multiprocessor implementations. The 3D integration has the potential to increase device density, providing shorter wire lengths and faster on-chip communication compared with the 2D integration. Traditional stacking technologies such as System-in-Package (SiP) and Package-on-Package (PoP) have been integrated into manufacturing technology. Recent researches have focused on TSV [4]. TSV is a viable solution in building 3D chips by stacking IC layers together using vertical interconnects. These interconnects are formed through the silicon die to enable communication among layers. Layers with different functionalities can be implemented in a 3D chip. The manufacturing process of the TSV is complex and expensive [4], therefore finding an optimal number and placement of TSVs is critical. It is presented that, the balance between performance and manufacturing cost is essential in designing a 3D chip [5].

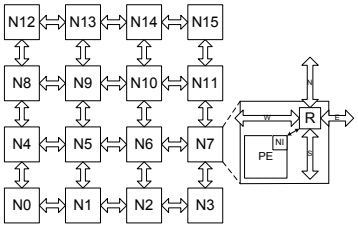

Fig. 1. An example of 4×4 NoC using mesh topology

With limited resources between layers, it is obvious that better or even optimal efficiency can be achieved through appropriate scheduling of multi-threaded tasks in large scale 3D multicore processors. The design of operating system schedulers is one of the most important issues for CMPs. Several multiprocessor scheduling policies such as round robin, co-scheduling and dynamic partitioning have been studied and compared in [6]. However, these policies are designed mainly for the conventional shared bus based communication architecture. Many heuristic-based scheduling methods have been proposed [7,8]. These methods are based on different assumptions, e.g. the prior knowledge of the tasks and execution time of each task in a program, presented as a directed acyclic graph. Hypercube scheduling has been proposed for off-chip systems [9]. Hypercube systems, usually based on Non-Uniform Memory Access (NUMA) or cache coherent NUMA architectures [10], are different from CMPs. Task scheduling for 2D NoC platforms is studied in [11] and [12]. The impact of limited resources between layers is not considered in these papers.

In our paper, we propose and discuss a novel greedy heuristic approximation scheduler for TSV constrained 3D multicore processors which aims to reduce the average network latency between caches and processing cores. With the decrease of the latencies, lower power consumption and higher performance can be achieved. To confirm our theory, we model and analyze a 64-core, 2-layer NoC with 8×8 meshes, present the performance of an application with different allocation strategies using a full system simulator.

2 3D NoC with through Silicon via Constraints

A modern multi-core processor is composed of several parts, e.g. processor core, shared last level cache, I/O and memory controller. Processor core and shared cache consume over 80% of the die area [13]. The total area of Sun SPARC chip is 396mm^2 with 65nm fabrication technology. Each core has an area of 14mm^2, thus with 16 cores the total area of cores is 224mm^2 (56.6%). Shared caches and other components occupy 172mm^2 (43.4%). As explained above, nearly half of the die area is devoted to cores and the other half is devoted to shared caches and other circuits. A natural way of applying 3D integration is to partition all the processors to one layer and other components to the other layer.

2.1 Processors and Caches

There is a significant concern for thermal hot-spots brought by packing layers vertically. It is expectable that since the processors consume overwhelming majority of power in a chip, stacking multiple processor layers would be unwise for heat dissipation. According to [5], heat dissipation is a major problem by stacking multiple processor layers even if processors are interlaced vertically. Therefore, in consideration of heat dissipation of current CMP, the processor layer should be on top of the chip.

In our paper, based on the above analysis, we use a 3D multicore processor model of two layers. The top layer is an 8×8 mesh of 64 Sun SPARC cores. Each core, scaled to 32nm technology, has an area of 3.4mm^2. We simulate the characteristics of a 64MB, 64 banks, 64-bit line size, 4-way associative, 32nm cache by CACTI [14]. Results show that the total area of cache banks is 204.33mm^2. Each cache bank, including data and tag, occupies 3.2mm^2. The cache layer has an 8×8 mesh of cache banks. Routers are quite small compared with processors and caches, e.g. we calculate a 7-port 3D router to be only 0.096mm^2 under 32nm. The total area of the processor is supposed to be below 300mm^2. Figure 2 shows a model with two layers and 16 processors only.

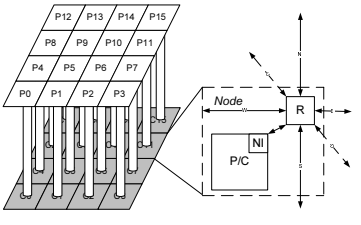

Fig. 2. A 3D NoC with one processor layer (upper) and one cache layer (lower), layers are fully connected by TSVs

2.2 Constraints of the through Silicon via

TSV is the most promising solution for building 3D chips. There are several types of TSVs, e.g. data signal transmission, control signal transmission, power distribution and thermal dissipation. In our paper, a pillar is defined as a bunch of TSVs, including TSVs for data, control and power distribution. On the assumption that the power supply voltage is 1V, a practical aspect ratio for TSVs is between 10:1 to 5:1, in which signal TSVs are dominant ones [15]. As it is shown in Figure 1, routers in the 2D NoC have five ports to connect to five directions,

namely, North, East, West, South and Local PE. For the vertical communication between different layers, routers in a generic 3D NoC model have two more ports and the corresponding virtual channels, buffers and crossbars to connect to the Up and Down pillars (Figure 2). It is noteworthy that routers in our 3D NoC require less than seven ports, e.g. router of P12 in Figure 2 has only four ports (East, South, Local PE and Down).

It is obvious that the maximum performance can be achieved by full layer-layer connection, e.g. all routers are connected with up/down routers by pillars. However, as the number of tiles grow, it might not be practical to assume that each tile will be connected with corresponding TSVs because of the manufacturing cost and chip area. Assuming that a flit in a NoC is 128 bits, full layer-layer connection for an 8×8 NoC would require $128 \times 8 \times 8 = 8,192$ TSVs for parallel data signals. Other TSVs are required for power, thermal and control. Several researches have shown that [4,16], TSV processing cost is the dominating cost for a 3D wafer. It is cheaper to manufacture a 3D chip with a fewer pillars between layers, in this case, multiple nodes have to share a pillar, high congestion could be created on a pillar, leading to communication bottlenecks. In [5], the placement of pillars is studied, an optimal placement of TSVs for an 8×8 mesh with 16 pillars is presented to minimize traffic contention between layers (Figure 3). The overall performance and total number of TSVs are 92% and 20% compared with full layer-layer connection respectively, achieving a good balance between performance and manufacturing cost.

56	57	58	59	60	61	62	63
48	49	50	51	52	53	54	55
40	41	42	43	44	45	46	47
32	33	34	35	36	37	38	39
24	25	26	27	28	29	30	31
16	17	18	19	20	21	22	23
8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7

Fig. 3. Each gray nodes is attached to a pillar. Numbers denote the sequence of nodes.

Assuming XYZ deterministic routing, Equation 1 shows the access time (latency) required for a core-cache communication. The latency involves in-tile links (Between NI and PE, L_{Link_delay1}), router (L_{Router_delay}), tile-tile links (L_{Link_delay2}), the number of hops required to reach the destination (n_{hop}) and the delay caused by TSV (L_{TSV_delay}). Since the delays of link, router and TSV are fixed, hop count is the most important metric in determining latency. Figure 4 shows the average hop counts required for a core to access the shared cache nodes (AHPC). Obviously, without proper schedule, the communication overhead can be an obstacle. For example, nodes at corners of the NoC have much higher AHPC than nodes in the center. However, nodes directly connected with a pillar usually have lower AHPC, sometimes even lower than inner nodes, e.g. the AHPC for the node 38 is 5.75, lower than 6.75 of the node 37. Scheduling a task to the node 38 is therefore preferable than 37, since the average delay to the shared caches is lower.

8.25	7.25	8.25	6.75	7.75	6.75	7.75	8.25
7.75	8.25	6.75	5.75	6.75	7.75	8.25	7.25
6.75	7.75	6.25	6.25	5.25	6.25	6.75	8.25
7.75	6.25	5.25	6.25	6.25	6.75	5.75	6.75
6.75	5.75	6.75	6.25	6.25	5.25	6.25	7.75
8.25	6.75	6.25	5.25	6.25	6.25	7.75	6.75
7.25	8.25	7.75	6.75	5.75	6.75	8.25	7.75
8.25	7.75	6.75	7.75	6.75	8.25	7.25	8.25

Fig. 4. Gray nodes are attached with a pillar, number means average hop counts to all cache nodes

$$L_{CoreCache} = 2\times L_{Link\_delay1} + (n_{hop} + 1)\times L_{Router\_delay} + n_{hop}\times L_{Link\_delay2} + L_{TSV\_delay} \quad (1)$$

3 The Scheduling Algorithm

Our proposed scheduling algorithm takes into consideration of on-chip topology and TSV placement, scheduling decisions are made based on these information. The aim of the algorithm is to minimize average network latency of the system, which is an important factor of system performance. We use a 3D NoC model as described below.

Definition 1. *A 3D NoC* $N(P(X,Y),C(X,Y))$ *consists of a layer of PE mesh* $P(X,Y)$ *(width* X*, length* Y*); and a layer of cache mesh* $C(X,Y)$*. Layers are connected by TSVs, only a quarter of nodes are connected (Figure 3).*

Definition 2. *Each node is denoted by a coordinate* (x,y)*, where* $0{\leq}x{\leq}X-1$ *and* $0{\leq}y{\leq}Y-1$*.*

Definition 3. *The Manhattan Distance between two PEs* $n_i(x_i,y_i)$ *and* $n_j(x_j,y_j)$ *is MD(*n_i*,*n_j*),* $MD(n_i,n_j)=|x_i-x_j|+|y_i-y_j|$*. Two nodes in the same layer* $n_1(x_1,y_1)$ *and* $n_2(x_2,y_2)$ *are interconnected by a router and related link only if they are adjacent, e.g. MD(*n_1*,*n_2*)=1.*

Definition 4. *A task* $T(n)$ *with* n *threads requests the allocation of* n *cores.*

Definition 5. n_{Free} *is a sorted list of all unallocated nodes in* P*, such that:* $AHPC_{nFree1}{\leq}AHPC_{nFree2}{\leq}AHPC_{nFree3}{\leq}...AHPC_{nFreek}$*.*

Definition 6. $R(T(n))$ *is a unallocated region in* P *with* n *cores for* $T(n)$*.*

To schedule a task efficiently, several metrics have to be considered, e.g. MD, AHPC and so on. Scheduling a task with only 1 thread is relatively easy. In this case, nodes 19, 29, 34 and 44 are considered in the first place, if they are not utilized by other applications. The reason is that, these four nodes have the lowest AHPC (5.25). However, as the requested number of threads grows, other metrics have to be included. For example, a 2-thread task can be scheduled to nodes 19 and 29. In this case, the Inter Process Communication (IPC) between threads will suffer higher delay, since the messages have to go through nodes 20 and 21 according to XY routing. Another problem is fragmentation. Non-contiguous allocation of cores in a dynamic system can cause degradation of

overall system performance. The 2-thread task can be scheduled to nodes 19 and 20 as well. Despite the fact that the AHPC is increased by 1 for node 20 compared with node 29, the adjacent allocation will alleviate IPC bottleneck, and reduce fragmentation. We introduce Average Core-access Time (ACT), which is defined as the number of nodes a message has to go through from a PE to other PEs, $\forall i, j \in P$.

$$ACT = \frac{\sum MD(n_i, n_j)}{n} \tag{2}$$

Such that: $\forall i \neq j \in P$ and $n_i \neq n_j$

According to the equation, the ACT is 3 and 1 for nodes 19/29 and 19/20, respectively. The delay for a core-core communication is shown in Equation 3. Obviously, allocation 19/29 will incur much higher router delay and delay of tile-tile links, comparing with allocation 19/20. It is noteworthy that a core-core communication is a intra-layer communication, while a core-cache communication is a inter-layer communication.

$$L_{CoreCore} = 2 \times L_{Link\_delay1} + (n_{hop}+1) \times L_{Router\_delay} + n_{hop} \times L_{Link\_delay2} \tag{3}$$

For a rectangular core allocation with $A \times B$ nodes, according to [17], ACT can be calculated in an easier way (Equation 4). For example, 4×4 and 2×8 are possible rectangular core allocations for a task with 16 threads. However, the value of ACT in 4×4 is smaller than in 2×8 (2.5 and 3.125). In consideration of ACT, an allocation shape have a lower ACT number if it is closer to a square. Figure 5a and 5b show two core allocation schemes for a task with 15 threads. In Figure 5b, the number of ACT is lower than in Figure 5a (2.4177 and 2.4888 respectively).

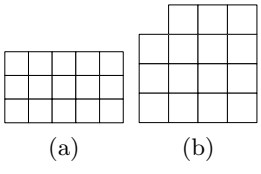

Fig. 5. Comparison of two core allocation schemes for 15 threads

$$ACT = \frac{A+B}{3} \times (1 - \frac{1}{A \times B}) \tag{4}$$

A scheduling algorithm should have a low computation complexity and should deliver an optimal or near-optimal results. This is due to the scheduling has to be solved online, and the time for solving the scheduling is a part of the overall system response time. It is clear that we should not try to solve the scheduling problem optimally, in case the computation complexity is too high. Given a task with n executing threads, we define the problem as determining the near-optimal core allocation for the task by selecting a region containing of n cores. The pseudo code of the algorithm is shown in Algorithm 1.

Algorithm 1. The Greedy Heuristic Approximation Scheduling Algorithm

Input: A mesh based NoC N with TSV constrains, a task with n threads
Output: An allocated region R, containing n processors

1 Pop the first node as an initial node u_0 from n_{Free}, and push u_0 to R

2 $n_{MD} := n_{Free}$ sorted as MD(u_0,n_{Free})

3 **while** n_{MD} *is not empty* **do**
4 Pick a node $u_n(x_i, y_i)$ from n_{MD} with smaller AHPC
if *several nodes with same AHPC* **then**
5 pick a node $u_n(x_i, y_i)$ with smaller ACT in the result region
6 **end**
7 **if** *several nodes with same ACT* **then**
8 pick a node $u_n(x_i, y_i)$ which $x_i \to \frac{X}{2}$ and $y_i \to \frac{Y}{2}$
9 **end**
10 Pop $u_n(x_i, y_i)$ from n_{MD}, and push u_n to R
11 **end**

Line 1 sets the starting node of the algorithm, which is the one with the lowest AHPC. A list n_{MD} contains nodes sorted based on MD from the starting node. The adjacent nodes are always considered first, in terms of AHPC. ACT will be calculated, in case several nodes are with the same AHPC. If ACTs for the allocation strategies are still the same, a node closer to the center of the network will be selected (considering the statistical variance of the coordinates of two nodes, Equation 5). This is due to the fact that, nodes in the center usually have lower AHPC than nodes in the border, following steps may have better results from this heuristic.

$$Var(n) = \frac{1}{2} \times [(x_i - \frac{X}{2})^2 + (x_j - \frac{Y}{2})^2] \tag{5}$$

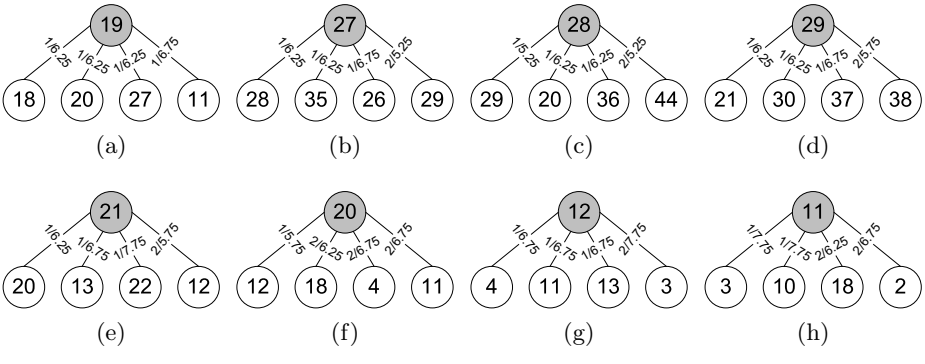

Fig. 6. The node selection steps for the algorithm

We analyze an example of the algorithm. Figures 6a to 6h shows the steps for node selection, starting from node 19. The number between two nodes n_i and n_j means MD(n_i,n_j) and AHPC(n_j). Note that we only show 4 child nodes in these figures. The actual list n_{MD} and n_{Free} may contain more nodes. As illustrated in Figure 6a, node 19 has 4 adjacent nodes and 3 of them are with the same AHPC and ACT. However, in terms of distance to the center, node 27 is selected ($Var(27) < Var(20) < Var(18)$). The selection of the next node follows the similar rule: same AHPC, same ACT, same variance. In this case we choose node 28, having a smaller node number than node 35. Figure 6c demonstrates that, node 29 is selected due to its lowest MD and AHPC. The next step involves different ACTs: both node 21 and node 30 have the lowest AHPC, however the ACTs for the two nodes are different (2 for node 30, and 1.8 for node 21). Node 20 and 12 are selected as the sixth and seventh node, respectively, due to their lowest AHPC. The next node (11) is picked out, on account of its lower ACT than the others. It is noteworthy that the aforementioned greedy heuristic approximation algorithm generates near-optimal scheduling solution in most cases. However, in our algorithm we put adjacent nodes as the first priority, the AHPC and ACT are considered next. This strategy may generate non-optimal scheduling for certain applications.

Take a 4-thread application for example. As shown in Figure 7, the algorithm will choose nodes 19, 27, 28 and 29 for allocation.

An IPC-intensive application may suffer from the long distance communication of node 19 and 29. In this case, node 20 is a better choice than 29 since the ACT is lower. Despite our goal is to find a near-optimal scheduling using MD, AHPC and ACT, the weight of these metrics should be considered as well. Different applications have their own profile: some have higher demand of caches, some have higher volume of IPC. It is difficult to determine the behavior of an application automatically beforehand, since there are millions of them and the number is still increasing. One feasible way is to add an interface between the application and the OS, the application will tell the OS its behavior. Another way is to add a low overhead profiling module inside the OS. Program access patterns are traced dynamically, and possibly migrated for better allocations.

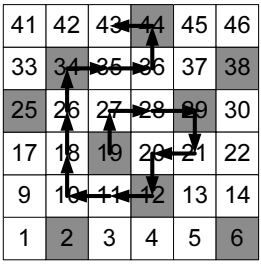

Fig. 7. The execution of our algorithm, starting from node 19 and selected 16 nodes

4 Experimental Evaluation

4.1 Experiment Setup and Application

The simulation platform is based on a cycle-accurate NoC simulator which is able to produce detailed evaluation results. The platform models the routers and links accurately. The router includes a routing computation unit, a virtual

channel allocator, a switch allocator, a crossbar switch and 4 input buffers. Deterministic XYZ routing algorithm has been selected to avoid deadlocks. We use a 64-core multiprocessor which models a single-chip CMP for our experiments. The 3D architecture in this paper has two layers; the first layer contains PEs (each running at 2GHz with a private L1 cache, split I+D, each 16KB, 4-way associative, 64-bit line, 3-cycle), the second layer consists of shared L2 caches (unified 64 banks, each 1MB, 64-bit line, 6-cycle). The simulations are run on Solaris 9 based on UltraSPARCIII+ instruction set in-order issue structure. The simulated memory/cache architecture mimics Static Non-Uniform Cache Architecture. A two-level directory cache coherence protocol called MOESI, based on MESI, has been implemented in our memory hierarchy in which each L2 bank has its own directory. We use Simics [18] as our full system simulation platform.

We select FFT [19] and Radix [20] as experiment applications. The FFT algorithm is a one-dimensional, radix-n, six-step algorithm optimized to minimize IPC. The communication between processors only take place at the last stage of the execution. However the network traffic and cache miss rate are very high. The Radix sort algorithm assigns each processor a part of the sorting keys. For every iteration in the algorithm, a permutation for the keys is required to create a new array for the next iteration. This will incur all-to-all communication among processes. Hence Radix represents an application with high IPC.

4.2 Result Analysis

We evaluate performance in terms of Average Network Latency (ANL), Execution Time (ET) and Cache Hit Latencies (CHL). ANL represents the average number of cycles required for the transmission of all messages. The number of required cycles for each message is calculated from the injection of the message header into the network at the source node, to the reception of the tail flit at the destination node. Under the same configuration and workload, lower values are favorable. We analyze two core allocations for a 2-thread task: *T2-1* is from our algorithm, which contains nodes 19 and 27. It has lowest ACT values, however the AHPC is not optimal. *T2-2* is an alternative allocation, which contains nodes 19 and 29. In this case, the AHPC is minimized. The *T4-1*, *T4-2* and *T4-3* are three core allocations for a 4-thread task: *T4-2* contains nodes 19, 20, 27 and 28, represents lowest ACT; *T4-3* contains nodes 19, 29, 34 and 44, represents lowest AHPC. Our algorithm selects *T4-1*, it has neither lowest ACT nor AHPC numbers. However we believe it could be a good balance for the two metrics.

The results are illustrated in Figure 8. The core allocation of our scheduling algorithm for 2 threads outperforms the other in terms of ANL. The improvement is more notable in 2-thread FFT and Radix, with 9.26% and 11.77% reduced latency, respectively, compared with the *T2-2* allocation. This is primarily due to the reduced communication overhead between two PEs. We note that the reduced AHPC in *T2-2* failed to compensate the increasing ACT, in terms of ANL. The CHL in *T2-2* directly reflects the reduced AHPC. However, the average runtime of two applications show that, our algorithm spends 15.58% shorter time than *T2-2*. Considering a 4-thread task, we note that both ACT and AHPC

play important roles in overall performance. For example, despite the fact that *T4-2* has lowest ACT, the ANL for two applications is 3.76% higher than in our algorithm. This leads to a higher ET as well. Allocation *T4-3* performs better than our scheduling in the 4-thread FFT. This is because of, in FFT, the communication between threads only happens at the last stage of the execution. In this case, we observe that the trade-off for ACT is worthy. However, applications that heavily rely on IPC, e.g. Radix, will suffer from the *T4-3*. The ET of *T4-3* is 24.24% longer than in *T4-1*.

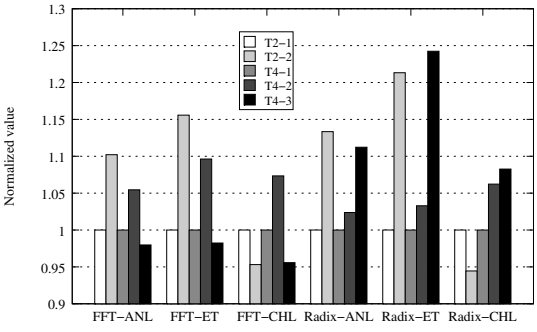

Fig. 8. Performance for FFT and Radix

5 Conclusion and Future Work

In this paper, we studied the problem of process scheduling for future 3D multicore processors. A model for NoC-based 3D CMP was defined. We analyzed process scheduling in terms of average hop counts for core-cache accesses (AHPC) and average core access time (ACT) in 3D CMPs with constraints of inter-layer connections. A greedy heuristic approximation algorithm was proposed to reduce overall on-chip communication latencies and improve performance. Results have shown that, with proper scheduling, performance improved significantly in most cases. The impact of ACT and AHPC was discussed. The results of this paper give a guideline in designing schedulers for future 3D CMPs. Our next step is to evaluate more applications with different access profiles and number of threads. The weight of AHPC and ACT will be analyzed and compared, and the trade-off for finding the best allocation strategy will be studied.

References

1. Dally, W.J., Towles, B.: Route packets, not wires: on-chip inteconnection networks. In: Proceedings of the 38th Conference on Design Automation, pp. 684–689 (June 2001)
2. Intel: Intel research areas on microarchitecture (May 2011), `http://techresearch.intel.com/projecthome.aspx?ResearchAreaId=11`
3. Sylvester, D., Keutzer, K.: Getting to the bottom of deep submicron. In: ICCAD 1998, pp. 203–211 (November 1998)
4. Velenis, D., Stucchi, M., Marinissen, E., Swinnen, B., Beyne, E.: Impact of 3d design choices on manufacturing cost. In: IEEE 3DIC 2009, pp. 1–5 (September 2009)
5. Xu, T.C., Liljeberg, P., Tenhunen, H.: Optimal number and placement of through silicon vias in 3d network-on-chip. In: Proc. of the 14th DDECS. IEEE (2011)

6. Leutenegger, S.T., Vernon, M.K.: The performance of multiprogrammed multiprocessor scheduling algorithms. In: Proc. of the SIGMETRICS, pp. 226–236 (April 1990)
7. Chen, C., Lee, C., Hou, E.: Efficient scheduling algorithms for robot inverse dynamics computation on a multiprocessor system. IEEE Transactions on Systems, Man and Cybernetics 18(5), 729–743 (1988)
8. Hakem, M., Butelle, F.: Dynamic critical path scheduling parallel programs onto multiprocessors. In: Proceedings of 19th IEEE IPDPS, p. 203b (April 2005)
9. Sharma, D.D., Pradhan, D.K.: Processor allocation in hypercube multicomputers: Fast and efficient strategies for cubic and noncubic allocation. IEEE Transactions on Parallel and Distributed Systems 6(10), 1108–1123 (1995)
10. Laudon, J., Lenoski, D.: The sgi origin: a ccnuma highly scalable server. In: Proc. of the 24th International Symposium on Computer Architecture, pp. 241–251 (June 1997)
11. Chen, Y.J., Yang, C.L., Chang, Y.S.: An architectural co-synthesis algorithm for energy-aware network-on-chip design. J. Syst. Archit. 55(5-6), 299–309 (2009)
12. Hu, J., Marculescu, R.: Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints. In: DATE 2004, p. 10234. IEEE Computer Society, Washington, DC (2004)
13. IBM: Ibm power 7 processor. In: Hot Chips 2009 (August 2009)
14. Shyamkumar, T., Naveen, M., Ho, A.J., Jouppi Norman, P.: Cacti 5.1. Technical Report HPL-2008-20, HP Labs (2008)
15. Semiconductor Industry Association: The international technology roadmap for semiconductors (itrs) (2007), `http://www.itrs.net/Links/2007ITRS/Home2007.htm`
16. Lau, J.H.: Tsv manufacturing yield and hidden costs for 3d ic integration. In: Proc. of the 60th ECTC, pp. 1031 –1042 (June 2010)
17. Lei, T., Kumar, S.: A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In: DSD, pp. 180–187 (September 2003)
18. Magnusson, P., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson, F., Moestedt, A., Werner, B.: Simics: A full system simulation platform. Computer 35(2), 50–58 (2002)
19. Bailey, D.H.: Ffts in external or hierarchical memory. The Journal of Supercomputing 4, 23–35 (1990), doi:10.1007/BF00162341
20. Blelloch, G.E., Leiserson, C.E., Maggs, B.M., Plaxton, C.G., Smith, S.J., Zagha, M.: A comparison of sorting algorithms for the connection machine cm-2. In: Proceedings of the 3rd SPAA, pp. 3–16. ACM, New York (1991)