Study of Hierarchical N-Body Methods for Network-on-Chip Architectures[*]

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 study two hierarchical N-Body methods for Network-on-Chip (NoC) architectures. The modern Chip Multiprocessor (CMP) designs are mainly based on the shared-bus communication architecture. As the number of cores increases, it suffers from high communication delays. Therefore, NoC based architecture is proposed. The N-Body problem is a classical problem of approximating the motion of bodies. Two methods, namely Barnes-Hut (Barnes) and Fast Multipole (FMM), have been developed for fast simulation. The two algorithms have been implemented and studied in conventional computer systems and Graphics Processing Units (GPUs). However, as a promising unconventional multicore architecture, the evaluation of N-Body methods in a NoC platform has not been well addressed. We define a NoC model based on state-of-the-art systems. Evaluation results are presented using a cycle accurate full system simulator. Experiments show that, Barnes scales better (53.7x/Barnes and 36.6x/FMM for 64 processing elements) and requires less cache than FMM. However, we observe hot-spot traffic in Barnes. Our analysis and experiment results provide a guideline for studying N-Body methods in a NoC platform.

1 Introduction

It is predictable that in the near future, hundreds or even more cores on a chip will appear on the market. 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 architectures will suffer from high communication delay and low scalability. To address these problems, NoC communication backbone was proposed for future multicore systems [1]. Network communication methodologies are brought into on-chip communication. More transactions can occur simultaneously and thus the delay of the packets is reduced and the throughput of the system is increased. Moreover, as the links in NoC are based on point-to-point mechanism, the communication among cores can be pipelined to further improve the system

[*] This work is supported by Academy of Finland and Nokia Foundation. The authors would like to thank the anonymous reviewers for their feedback and suggestions.

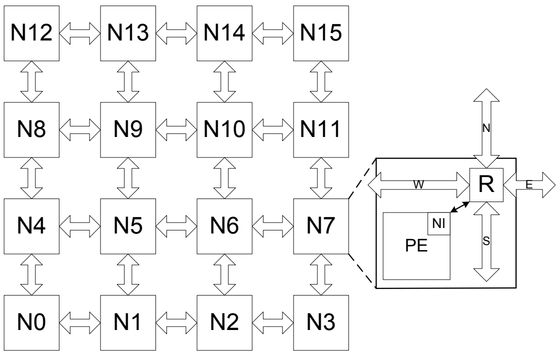

Fig. 1. An example of 4×4 NoC using mesh topology

performance. 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 the 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, 100M transistor, 275mm^2 2D NoC under 65nm technology [2]. An experimental microprocessor containing 48 x86 cores on a chip has been created, using 4×6 2D mesh topology with 2 cores per tile [2]. The TILE-Gx processor from Tilera, containing 16 to 100 general-purpose processors in a single chip, is available for commercial use [3].

The N-Body problem is a classical problem of approximating the motion of bodies that interact with each other continuously. The bodies are usually galaxies and stars in an astrophysical system. The gravitational force of bodies is calculated according to Newton's Principia [4]. The N-Body problem is used in other computations and simulations as well, e.g. the interference of wireless cells and protein folding [5]. Several algorithms have been developed for N-Body simulation. In principle, to be precise, the simulation requires the calculation of all pairs, since the gravitational force is a long range force. However the computation complexity of this method is $O(n^2)$ [6]. J. Barnes et al. and L. Greengard introduced two fast hierarchical methods [7,8]. A tree is build firstly according to the position of the bodies in the physical space. The interactions are calculated by traversing this tree. The computation complexities in these algorithms are reduced to $O(nlog_n)$, or even $O(n)$ in some cases.

The performance of these two algorithms has been studied in traditional cache-coherent shared address space multiprocessors, e.g. Standford DASH, KSR-1 and SGI-Challenge [9]. A simulator is used for examining the implications of the two algorithms in a multiprocessor architecture [10]. However, the previous works are based on conventional architectures, e.g. bus-based multiprocessors,

[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.

physically distributed main memory or cache-only memory architecture. NVIDIA has presented a CUDA-based N-Body simulation by calculating the gravitational attractions of all body-pairs [11]. Hierarchical methods for GPGPU-based systems have been implemented and compared in [12] and [13]. As a promising unconventional multicore architecture in the future, the implementation of these algorithms in a NoC platform has not been well studied. To design efficient NoCs, designers need to understand the characteristics of the applications, e.g. the amount of communication among cores, caches and memory controllers, as well as the scaling of the application with the designated architecture. In our paper, we study and discuss two hierarchical N-Body algorithms for the NoC architecture. To validate our study, we model and analyze a 64-core NoC with 8×8 mesh, present the performance and network traces of the two algorithms using a full system simulator.

2 Modeling of the Network-on-Chip

The Tilera TILE processor family includes TILE64, TILEPro and TILE-Gx members. The basic architecture of these processor are the same: an array of 16 to 100 general purpose RISC processor cores (tiles) in a on-chip mesh interconnect. Each tile consists a core with related L1 and L2 caches. The memory controllers are integrated on the chip as well.

Figure 2 shows the architecture diagram of TILE-Gx processor [3]. Each tile consists of a 64-bit VLIW core with private L1 cache (32KB instruction and 32KB data) and shared L2 cache (256KB per tile). Four 64-bit DDR3 memory controllers, duplexed to multiple ports, connect the tiles to the main memory.

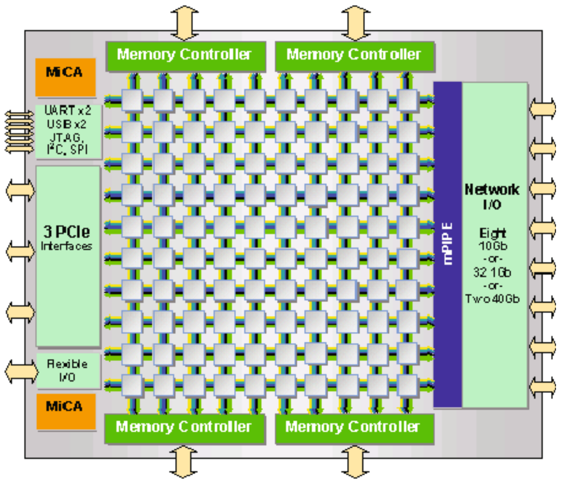

Fig. 2. The Tilera TILE multicore processor with 100 cores

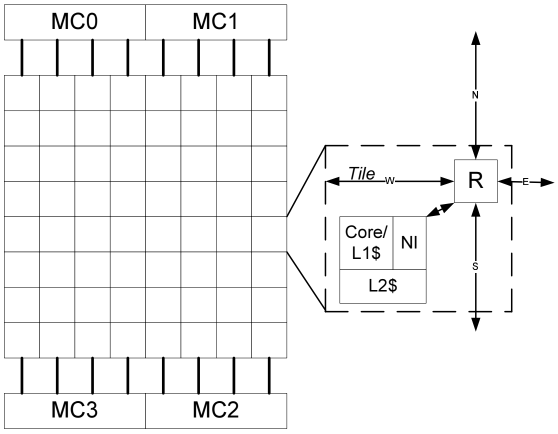

Fig. 3. An 8×8 mesh-based NoC with memory controllers attached to up and down sides

The L2 caches and the memory are shared by all processors. The processor operates at 1.0 to 1.5GHz, with typical power consumption of 10 to 55W. The I/O controllers are integrated on chip to save costs of north and south bridges. The mesh network provides bandwidth up to 200Tbps.

To analyze the low-level behavior of an application, we model a NoC similar to the Tilera TILE architecture. The processing core of the NoC is a Sun SPARC RISC core [14], the area is 14mm^2 with 65nm fabrication technology. Scaled to 32nm technology, each core has an area of 3.4mm^2. We simulate the characteristics of a 16MB, 64 banks, 64-bit line size, 4-way associative, 32nm cache by CACTI [15]. Results show that the total area of cache banks is 64.61mm^2. Each cache bank, including data and tag, occupies 1mm^2. Routers are quite small compared with processors and caches, e.g. we calculate a 5-port router to be only 0.054mm^2 under 32nm. The number of transistors required for a memory controller is quite small compared with a chip (usually billions). It is presented that a DDR3 memory controller is about 2,000 LCs with Xilinx Virtex-5 Field-Programmable Gate Array (FPGA) [16]. The total area of the chip is estimated to be around 300mm^2, comparable to the TILE-Gx. Figure 3 illustrates the architecture of the aforementioned NoC.

3 The Hierarchical N-Body Methods

In this section, we describe the two most important hierarchical N-Body algorithms that we used for analysis: the Barnes-Hut method [7] and the Fast Multiple Method (FMM) [8]. The two hierarchical methods build a structured tree firstly. The tree is built by subdividing space cells until a certain condition, e.g. reaching the maximum number of particles in a leaf cell. The physical space is represented by a hierarchical tree. The computation of interactions is done by traversing this tree. The two algorithms differ in the steps they use to calculate the interactions of particles.

In Barnes-Hut method, for each particle, the tree is traversed to compute the forces. It starts at the root of the tree, and traverses every cell. To reduce the computation complexity of long-range interactions, the subtree is approximated by the mass of the center cell, if the cell is far away from the particle. The accuracy of this methods is thus dependent on the approximation metrics. The Barnes-Hut method only computes the interactions for particle-particle and particle-cell.

The FMM computes the interactions for cell-cell as well, compared with Barnes-Hut. If two cells are far away from each other, the interaction between them is computed by the multipole expansion of the cells. The computation complexity is thus reduced. For uniform distributions, the complexity of FMM is $O(n)$, compared with $O(nlog_n)$ in Barnes-Hut. To develop a multithreaded program for both algorithms, the space is divided into several regions where each core is assigned with a region. A tree for the regions is built for the responsible core, and each core calculates its local tree. Most of the calculation time is spent in traversals of the tree to compute the forces. In a NoC platform, the performance of the algorithms will be affected by *(a)* long distance communication of nodes; *(b)* the initial distribution of particles; *(c)* the dynamic changing of position of particles; *(d)* hot-spot traffic.

4 Experimental Evaluation

4.1 Experiment Setup

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. State-of-the-art router in our platform includes a routing computation unit, a virtual channel allocator, a switch allocator, a crossbar switch and four input buffers. Deterministic XY routing algorithm has been selected to avoid deadlocks.

We use a 64-core network which models a single-chip NoC for our experiments. A full system simulation environment with 64 nodes, each with a core and related cache, has been implemented. The simulations are run on the Solaris 9 operating system based on the UltraSPARCIII+ instruction set in-order issue structure. Each processor core is running at 2GHz, attached to a wormhole router and has a private write-back L1 cache (split I+D, each 32KB, 4-way, 64-bit line, 3-cycle). The 16MB L2 cache shared by all processors is split into banks (64 banks, each 256KB, 64-bit line, 6-cycle). The simulated memory/cache architecture mimics SNUCA [17]. A two-level distributed directory cache coherence protocol called MOESI based on MESI [18] has been implemented in our memory hierarchy in which each L2 bank has its own directory. The protocol has five types of cache line status: Modified (M), Owned (O), Exclusive (E), Shared (S) and Invalid (I). We use Simics [19] full system simulator as our simulation platform. For both methods, we use the Plummer model [20] for particle generation, instead of uniform distribution. The multithreaded part of the programs utilizes the C/pthread model.

4.2 Result Analysis

We start by evaluating the computation time distribution and scalability of the two algorithms. Both algorithm applies same parameters. The results are listed in Table 1 and 2. The first five rows show the computation time from 4K to 64K particles, with 64 processors. In Barnes-Hut, around 90% of the total time are spent on force calculation (84.2% in 4K to 91.1% in 64K), while the time spent on other tasks (e.g. tree building) are relatively small. The Barnes-Hut method scales very well from 1 to 64 processors. The speedups for 64 processors are 53.7x and 61.8x for total execution time and force calculation time, respectively.

In Figure 4, the network request rates of 64 cores are illustrated. We simulate 64K particles in 5 time steps. The horizontal axis is time, segmented in 12.1M-cycle percentage fragments. The traffic trace has 165.2M packets. It is observed that, several nodes, especially N0 and N34, generate more data traffic than others (e.g. N0 14.18%, N34 12.19%, N12 5.3% and N20 2.76%). This introduces heavy hot-spot traffic in certain regions of the NoC. Notice that, the traffic patterns of other nodes are quite similar, they have a high traffic in the starting phase, and drop to a lower traffic after that. There are several time slices, for example 16% to 21%, when all processors are sending packets simultaneously. The reason is, the simulation has executed for 5 time steps, the positions of particles will change at the end of each time step. In terms of point-to-point traffic, several source-destination pairs, specifically originated from N0 and N34, generated a considerable amount of the traffic. We observe the top 5 pairs are: 34-62 (0.88%), 0-14 (0.63%), 0-58 (0.62%), 0-8 (0.60%) and 34-10 (0.51%). These hot-spot traffic can be alleviated with, e.g. long links between nodes, or increase the link bandwidth for hot-spot nodes.

Table 1. Time distribution and scalability of the Barnes-Hut Method

Configuration	**Total time**	**Treebuild**	**Forcecalc**	**Others**
64p/4K	19	1	16	2
64p/8K	41	2	35	4
64p/16K	87	5	76	6
64p/32K	184	8	168	8
64p/64K	385	15	351	19
4K/1p	1020	28	988	4
4K/2p	511	15	495	1
4K/4p	258	8	246	4
4K/8p	129	4	124	1
4K/16p	65	3	61	1
4K/32p	34	2	31	1
4K/64p	19	1	16	2

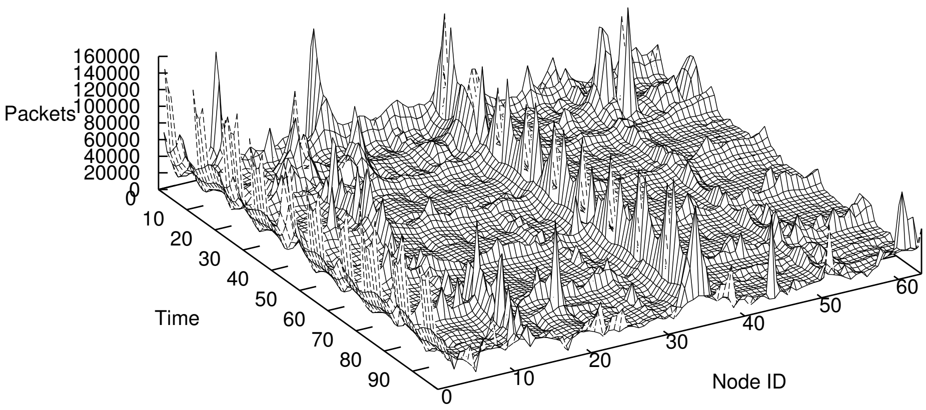

Fig. 4. Network request rate for 64-core NoC running Barnes

Table 2. Time distribution and scalability of the Fast Multipole Method

Configuration	Total time	Treebuild	Forcecalc	Barrier	Listbuild	Others
64p/4K	17	1	10	3	2	1
64p/8K	27	2	16	6	0	3
64p/16K	54	7	30	14	2	1
64p/32K	102	11	73	13	1	4
64p/64K	209	21	147	30	4	7
4K/1p	622	75	533	0	10	4
4K/2p	316	38	270	1	3	4
4K/4p	162	20	136	1	3	2
4K/8p	83	9	71	0	1	2
4K/16p	44	4	35	2	1	2
4K/32p	26	3	16	4	0	3
4K/64p	17	1	10	3	2	1

The time spent on force calculation in the Fast Multipole method is lower than Barnes-Hut (Table 2), e.g. 58.8% in 4K to 70.3% in 64K. Nearly 10% of the time are spent on tree building, and about 15% on barrier. The Fast Multipole method scales worse than Barnes. The speedups for 64 processors are 36.6x and 53.3x for total execution time and force calculation time, respectively. This is primarily due to the higher number of barriers in Fast Multipole method. It is noteworthy that, in spite of poor scaling, the Fast Multipole method spends less time for calculation. For example, it spends 54.3% of the total execution time in 64p/64K, compared with Barnes. In consideration of better scalability, the Barnes-Hut method could use shorter time in a systems with thousands of cores.

Figure 5 shows the network request rate of each processing core when running FMM in a 64-core NoC. The horizontal axis is time, segmented in 1.69M-cycle percentage fragments. The traffic trace has 57.4M packets. It is revealed that,

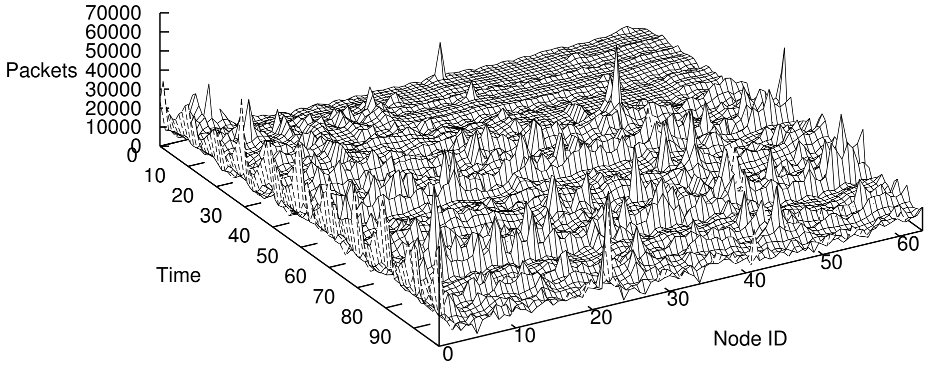

Fig. 5. Network request rate for 64-core NoC running FMM

several nodes (e.g. N0 7.6%, N46 4.15%, N13 2.72% and N7 2.71%) generate more data traffic than others. The network traffic is relatively low in the starting phase (before 30% of the time slice). After that time point, FMM shows similar traffic patterns as in Barnes. However, the hot-spot traffic in FMM is not as significant as Barnes. We note that, in terms of point-to-point traffic, a small portion of source-destination pairs generated a sizable portion of the traffic. For example, only 4 (19-60, 13-44, 60-19 and 0-29) of the pairs (in totally $64^2 = 4,096$) generated 1.42% traffic.

We evaluate other performance metrics of the two algorithms in terms of L2 cache miss rate (L2MR), misses per thousand instructions (MissPKI), Average Link Utilization (ALU) and Average Network Latency (ANL). ALU is calculated

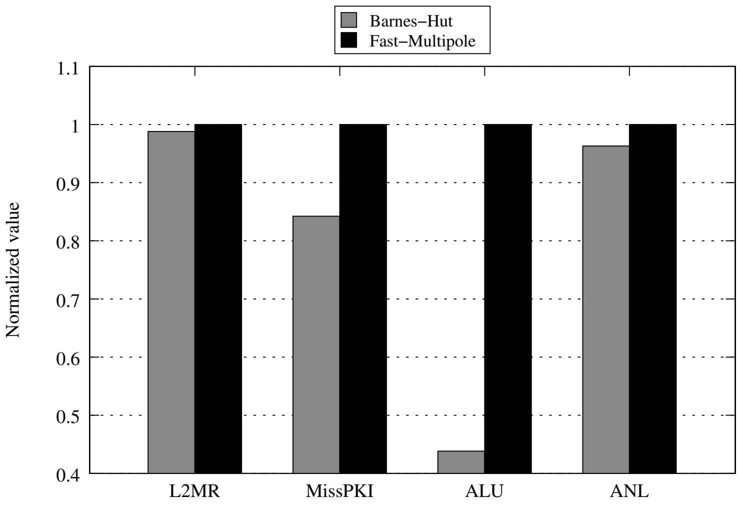

Fig. 6. Performance for Barnes and FMM

with the number of packets transferred between NoC resources per cycle. 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 of these metrics are favorable. The results are shown in Figure 6. We note that, in terms of L2MR and MissPKI, Barnes is lower than FMM (1.21% for L2MR and 15.77% for MissPKI respectively). This reflects, FMM requires more cache than Barnes. A system with limited cache could be unsuitable for FMM. The ALU of Barnes is only 43.83% of FMM, which means an alleviated network load. It is noteworthy that despite the fact that the value of Z axis in Figure 4 is twice as larger than Figure 5, each time slice in Figure 4 represents 12.1M cycles, compared with 1.69M cycles in Figure 5. Finally, the ANL of Barnes is 96.31% that of FMM, indicating that the network performance of Barnes is better, and hence having lower communication overhead.

5 Conclusion

The implementation of two hierarchical N-Body methods (Barnes-Hut and Fast Multipole) in a NoC platform was studied in this paper. Both scalability and network traffic for the two methods were analyzed. We studied an 8×8 NoC model based on state-of-the-art systems. The time distribution of the two methods, with 1 to 64 processing cores, were explored. We investigated the advantages and disadvantages of the two algorithms. The network requests rates of 64 processing cores were illustrated for both methods. Our experiments have shown that, the Barnes-Hut method generates more hot-spot traffic than Fast Multipole. However, it scales better, and has lower overall pressure to the on-chip network and caches, compared with Fast Multipole. The results of this paper gave guidance for analyzing hierarchical N-Body methods in a NoC platform.

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. Tilera: Tile-gx processor family (May 2011), `http://www.tilera.com/products/processors/TILE-Gx_Family`
4. Aarseth, S.J., Henon, M., Wielen, R.: A comparison of numerical methods for the study of star cluster dynamics. Astronomy and Astrophysics 37, 183–187 (1974)
5. Perrone, L., Nicol, D.: Using n-body algorithms for interference computation in wireless cellular simulations. In: Proc. of 8th Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 49–56 (2000)
6. Salmon, J.: Parallel n log n n-body algorithms and applications to astrophysics. In: Compcon Spring 1991, Digest of Papers, February-1 March, pp. 73–78 (1991)

7. Barnes, J., Hut, P.: A hierarchical o(n log n) force-calculation algorithm. Nature (1988)
8. Greengard, L.F.: The rapid evaluation of potential fields in particle systems. PhD thesis, New Haven, CT, USA (1987) AAI8727216
9. Holt, C., Singh, J.P.: Hierarchical n-body methods on shared address space multiprocessors. In: Proc. of 7th SIAM Conf. on PPSC (1995)
10. Singh, J.P., Hennessy, J.L., Gupta, A.: Implications of hierarchical n-body methods for multiprocessor architectures. ACM Tran. Comp. Sys. 13, 141–202 (1995)
11. Nyland, L., Harris, M., Prins, J.: Fast N-Body Simulation with CUDA. In: Nguyen, H. (ed.) GPU Gems 3. Addison Wesley Professional (August 2007)
12. Jetley, P., Wesolowski, L., Gioachin, F., Kalé, L., Quinn, T.: Scaling hierarchical n-body simulations on gpu clusters. In: SC 2010, pp. 1–11 (November 2010)
13. Hamada, T., Nitadori, K.: 190 tflops astrophysical n-body simulation on a cluster of gpus. In: SC 2010, pp. 1–9 (November 2010)
14. Tremblay, M., Chaudhry, S.: A third-generation 65nm 16-core 32-thread plus 32-scout-thread cmt sparc processor. In: ISSCC 2008, pp. 82–83 (February 2008)
15. Thoziyoor, S., Muralimanohar, N., Ahn, J.H., Jouppi, N.P.: Cacti 5.1. Technical Report HPL-2008-20, HP Labs
16. Global, H.: Ddr 3 sdram memory controller ip core (May 2011), `http://www.hitechglobal.com/IPCores/DDR3Controller.htm`
17. Kim, C., Burger, D., Keckler, S.W.: An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In: ACM SIGPLAN, pp. 211–222 (October 2002)
18. Patel, A., Ghose, K.: Energy-efficient mesi cache coherence with pro-active snoop filtering for multicore microprocessors. In: Proceeding of the Thirteenth International Symposium on Low Power Electronics and Design, pp. 247–252 (August 2008)
19. 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)
20. Dejonghe, H.: A completely analytical family of anisotropic Plummer models. Royal Astronomical Society, Monthly Notices 224, 13–39 (1987)