A Multi-banked Shared-L1 Cache Architecture for Tightly Coupled Processor Clusters

Mohammad Reza Kakoee
DEIS, University of Bologna
m.kakoee@unibo.it

Vladimir Petrovic
Elsys Eastern Europe
vladimir.petrovic@elsys-eastern.com

Luca Benini
DEIS, University of Bologna
luca.benini@unibo.it

***Abstract*—A shared-L1 cache architecture is proposed for tightly coupled processor clusters. Sharing an L1 tightly coupled data memory (TCDM) among a significant (up to 16) number of processors is challenging in terms of speed. Sharing L1 cache is even more challenging, since operation is more complex, as it eases programming. The feasibility in terms of performance of shared TCDM was shown in ST Microelectronics platform 2012, but the performance cost of supporting shared L1 cache remains to be proven.**

In this paper we show that replacing TCDM with a multi-banked shared-L1 cache imposes limited speed overhead. Of course, it comes at the cost of area and power. We explore the shared L1 cache architecture in terms of number of processing elements (PEs) and cache banks. Experimental results show that our multi-banked shared-L1 cache can operate with almost the same frequency as that of related TCDM architecture if the cache controller uses a cache line of 4 words. Results also show that, the area overhead with respect to TCDM is less than 18% for a cluster containing 16 Leon3 processors and 32 cache banks. We also show that the overhead on $MIPS/Watt$ and $MIPS/mm^2$ is from 5% to 30% depending on the size of processor in the cluster for a 16x32 configuration (16 cores and 32 cache/memory banks).

I. Introduction

Main SoC and microprocessor manufacturers are migrating to Multi-processor System on Chips (MPSoCs) and Chip Multi Processors (CMPs) for their latest products. In these devices many processing elements and cores are put together in the same chip and, as Moore's law continues to apply in the multi-core era, we can expect to see a geometrically increasing number of processing elements and memories [2]. Intel has announced "Knights Corner" as the first product based on Intel's Many Integrated Core (MIC) architecture in 22nm which will scale to more than 50 processing cores on a single chip [3]. Previously, Intel also developed a chip prototype [4] that included 80 cores (known as TeraFlops Research chip).

When considering a chip with multiple cores, we should decide whether the caches should be shared or local to each core. Implementing shared cache imposes latency overhead, and more wiring and complexity. However, having one cache per chip, rather than core, greatly reduces the amount of space needed, and thus a larger cache can be included on the chip. This trade-off leads to a multi-level cache hierarchy which generally operates by checking the smallest level one (L1) cache first; if it hits, the processor proceeds at high speed. If the smaller cache misses, the next larger cache (L2) is checked, and so on, before external memory is checked. Most of today's CMPs/MPSoCs (including research designs) assume private L1 caches and a shared L2. In this architecture providing a consistent view of memory with various cache hierarchies is a key problem. This *cache coherence* problem is a critical correctness and performance-sensitive design point for supporting the shared-memory model. Several works in the literature target cache coherency at both hardware and software levels including interconnect snooping, directory coherence, token coherence, and etc [9], [10]. All these techniques impose performance, power and area overhead. In addition, they need a special interconnect which supports cache coherency protocols.

On the other hand, although sharing L1 cache removes the need for coherency, it is undesirable if the hit latency requires each core runs much slower than a single-core chip. This may happen if the number of cores which share an L1 cache is high and the interconnect between cores and L1 has large latency. Thus, going for a shared-L1 cache is reasonable if the number of cores are limited and the interconnect has very low latency.

Recently, several many-core architectures have been proposed that leverage tightly-coupled clusters as a building block. Examples include the HyperCore Architecture Line (HAL) processors from Plurality [5], ST Microelectronics Platform 2012 [1], or even GPGPUs like NVIDIA Fermi [6]. In a shared memory paradigm, these designs try to overcome the scalability limitations encountered when increasing the number of processing elements (PEs) that share a unique interconnection and memory system [12] by creating a hierarchical design where PEs are clustered into small-medium sized subsystems. The number of PEs inside each cluster is normally less than 16 which makes it possible to design a low-latency interconnect between processors and L1 (in-cluster) memories, while scaling to larger system sizes is enabled by replicating clusters and interconnecting them with a scalable medium like a NoC. However in these systems the shared memory is a tightly coupled data memory (TCDM) which is non-cachable like a scratch-pad memory. This explicitly managed memory can improve performance and is more predictable than cache. However, it comes at the cost of manually managing data transfers and data consistency which can make parallel application development more complex. Some of the architectures like NVIDIA Fermi [6], [7] support shared-L1 cache as an alternative to shared-memory; however, they have put limitations on either hit latency (more than 2 cycles) or cache bandwidth. Moreover, they have not explained the detail of their shared-L1 cache architecture [7].

978-1-4673-2896-8/12/$31.00 ©2012 IEEE

Contribution. In this work we propose a shared-L1 cache architecture which similarly to the above designs takes advantage of tightly coupled clusters as a building block. However, instead of TCDM we propose a multi-banked shared-L1 cache inside a cluster. Each cluster contains several processing elements communicating with the shared cache through a low latency interconnection network. The main contribution is to show that a shared L1 cache is feasible for large tightly coupled clusters where the number of processing elements is beyond (up-to 16) what can be achieved with basic Symmetric Multiprocessing (SMP) technology (i.e. 4).

We explore this architecture in terms of number of PEs and cache banks. Clearly, L1 processor-to-memory interconnects must provide a huge bandwidth, coupled with ultra-low latency. For the interconnection network we use the architecture proposed by Rahimi et.al in [8]. They propose a fully combinational Mesh-of-Tree (MoT) interconnection network which is suitable for tightly-coupled processor clusters. An enhanced version of this network is implemented in ST Microelectronics Platform 2012 on 28nm technology [1]. This network provides single-cycle transfer from processor to memory and round-robin arbitration for a fair access to memory banks, as well as fine-grained address interleaving to reduce memory bank conflicts. Since this network is fully combinational, it is vulnerable to timing and delay variations. Authors in [14] propose a variation-tolerant mechanism to make this network reliable in presence of delay variations.

Experimental results show that our multi-banked shared-L1 cache can operate with almost the same frequency as that of equally sized TCDM. Synthesis results also show that the area overhead of our architecture compared to that of TCDM is 18% for the 16x32 configuration (16 LEON3 cores and 32 cache/memory banks).

II. Architecture

A. Multi-banked shared cache

In this section we describe our multi-banked shared-L1 cache architecture. This architecture is suitable for tightly-coupled processor clusters. Each cluster contains several PEs which share a multi-banked L1 cache. The cluster architecture is fixed and scaling is enabled by replicating clusters. The number of PEs inside each cluster is large enough (more than 4) so that we cannot efficiently develop an L1 cache coherency (for instance based on snooping) mechanism if we go for a private-L1/shared-L2 architecture. It is also small enough (up-to 16) which makes it feasible to design a high performance and low latency interconnection infrastructure for communication between cores and the shared cache. Figure 1 shows an abstract view of this interconnect which connects 4 cores to 8 banks of caches.

The interconnection network supports non-blocking communication between the processing elements (PEs) and cache banks (CBs), within a single clock. As shown in Figure 1, a combinational path is created through a network of primitive building blocks: routing primitives (circles blocks) and arbitration primitives (square blocks). The former are used to create independent routing paths (routing trees) from the PCs to the arbitration tree (and vice-versa). The latter are used to arbitrate concurrent requests (arbitration tree) and to route them up to the CBs ports and vice-versa.

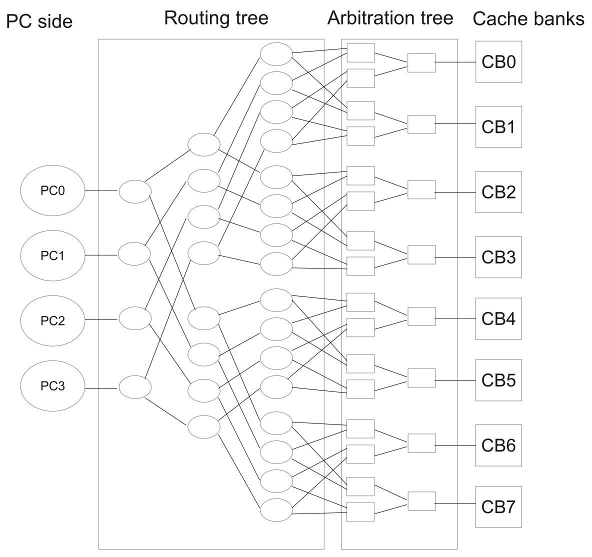

Fig. 1. Log. network 4x8: empty circles represent routing switches and empty squares represent arbitration switches.

The routing tree consists of simple routing switches which route each packet from processor side to memory side, and vice versa. The packets are routed individually based on packets address field. The switch has two directions: forward (PC ports) which sends out the incoming packet form its input port at processor side to one of its output ports at the cache side; backward which rolls packet back from cache side to processor side (CB ports). The forward packet contains address, data write, and control signal of cache while the backward packet contains the read data, and acknowledgment signal.

During a read/write operation, data and control signals are asserted by PCs. These signals are routed through routing switches, until they reach one of NxM ports of routing tree. In order to reach a cache bank the packet must be arbitrated among the other simultaneous requests for the same cache bank. After passing through all levels of arbitration switches, the request reaches the cache bank, and the read/write operation can be performed.

Request routing and arbitration are performed in a combinational way by using request and acknowledgment signals for arbitration, and address for routing across the switches. Once the request reaches the last level of the arbitration tree and gets the grant, a valid acknowledgement is asserted and propagated back to the related PC through the routing switches (backward). By receiving the acknowledgment signal, PC is able to issue the next read/write operation at the next clock cycle, otherwise it waits for ACK to be received.

In the following section we will briefly describe the internal architecture of a single cache bank.

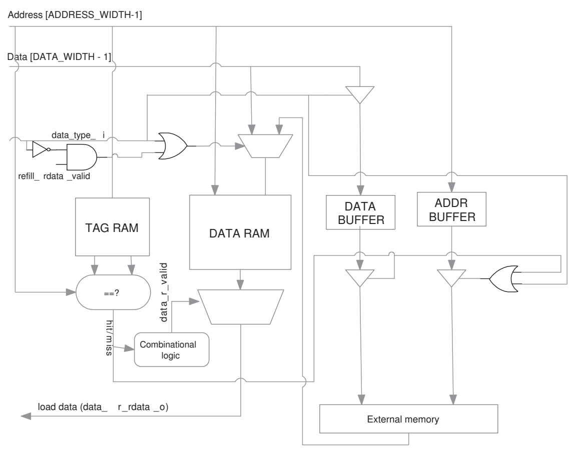

Fig. 2. Block diagram of a single cache bank.

B. Cache bank

For this work, we designed and developed a simple write-through cache controller. Figure 3 shows its block diagram. Like any other cache, our simple cache has separated memory blocks for data and tags. It is a direct mapped cache with parameterizable multi-word cache line. The cache also has a write-through buffer allowing us to temporary store data while the data is waiting to be written into the next level memory. We have chosen write-through mechanism as our write strategy because it provides better support for multi-cluster consistency models; write-back strategy is under development for future work.

From the processor side the cache controller accepts requests which are either read or write to the memory. All cache modules are connected to the slave side of logarithmic network. Therefore, every processor which is connected to the master side of this network can send request to every cache. Cache controller gets requests from the network and sends response back to the processor through the response path of the network. If the request is write, cache takes the data from the data input and stores it into the data memory at the address received from the processor and updates tag memory. Also, the same data is stored in the write-through buffer in order to be forwarded and written into the external memory. If the request is read and it hits the cache, related data will be sent to the processor at the next cycle. However, if a cache miss occurs on the read request, we first stall the processor. Then, the cache controller first writes the content of write-through buffer (if it is not empty) into the next level memory and then performs a refill from the external memory. Having completed the refill phase, the related data is sent to the processor and the stall signal goes down.

C. Critical Path

To clearly compare our multi-banked L1-cache with TCDM, we carefully analyzed the timing paths of both architectures. Based on our analysis, the request paths which start from processor, goes through the network and ends at the memory/cache banks are the same in both architectures. Therefore,

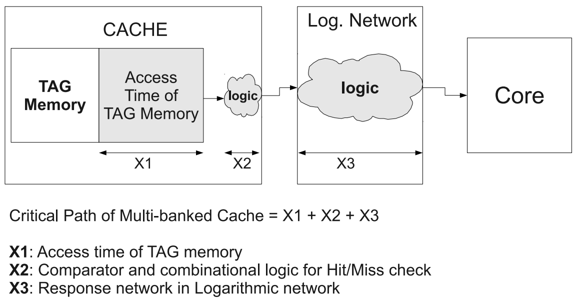
CACHE
TAG Memory
Access Time of TAG Memory
logic
Log. Network
logic
Core
X1
X2
X3
Critical Path of Multi-banked Cache = X1 + X2 + X3
X1: Access time of TAG memory
X2: Comparator and combinational logic for Hit/Miss check
X3: Response network in Logarithmic network

Fig. 3. Critical path of our multi-banked cache architecture.

we do not have any overhead on the request path compare to that of TCDM. However, the response path related to stalling the processor in case of miss is new in multi-banked cache. Based on our analysis this path is the critical path of our architecture which is shown in Figure 3.

The path shown in Figure 3 does not exist in TCDM, but there is a similar response path in TCDM which starts from the memory bank, goes through the network and ends at the processor. Comparing to the response path in TCDM, the critical path of our multi-banked L1-cache has the overhead of logic path inside the cache controller for checking the tag's data against the request address to determine miss/hit (X2 in Figure 3). However, since the data memory is usually larger than tag memory, the delay related to access time of data memory in TCDM is more than that of tag memory in the cache module (i.e. X1 in Figure 3). Therefore, if we select a small and high speed memory for tags, the delay of the critical path in our architecture becomes close to that of TCDM. We should note that by increasing the cache-line we can reduce the size of tag memory resulting in a lower access time. We will explore this in more details in the experimental results section.

III. Experimental Results

In this section, we discuss the experimental results for the multi-banked shared-L1 cache architecture in terms of delay, power, and area. We quantify the cost of replacing memory banks in TCDM with cache banks for several network cardinalities. To get these results, we synthesized the network and cache modules with the Farady 65nm technology library using Synopsys Design Compiler. In TCDM we considered 8KB memory for each memory module. For multi-banked cache, we put the same data memory (8KB) for each cache bank and we explored the tag memory from 512x20 bits (cache line of 4 words) to 2048x20 bits (cache line of 1 word).

A. Timing Analysis

As described before, the main overhead of replacing TCDM with cache banks on critical path is the response path which starts from tag memory and ends at the processor side. This is the critical path of our architecture. This path contains the access time of tag memory plus the logic inside cache controller to determine hit/miss plus the network's response path. tag's access time depends on the type and size of the

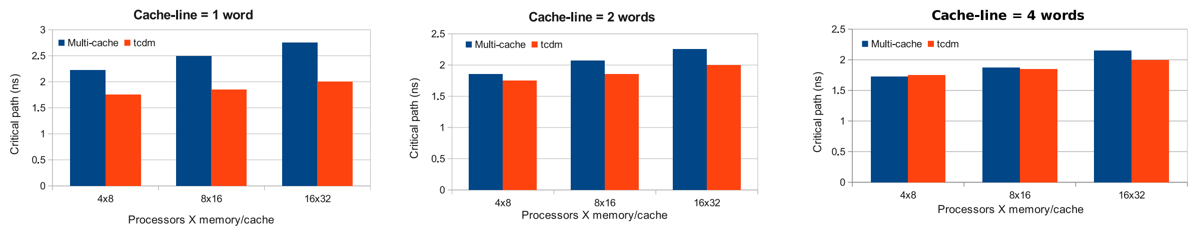

Fig. 4. Timing (ns) analysis of multi-banked cache Vs. TCDM.

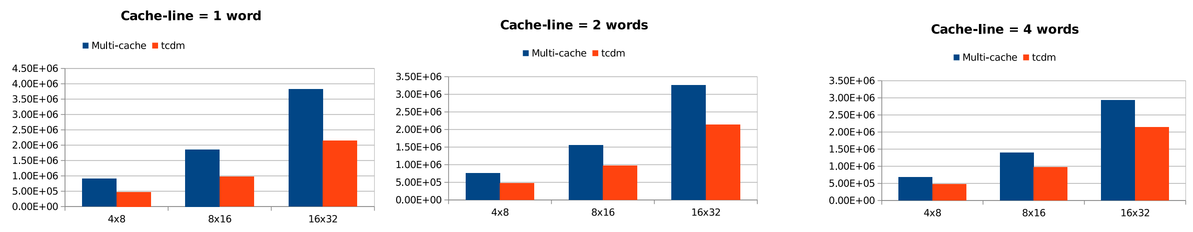

Fig. 5. Area $(um^2)$of multi-banked cache Vs. TCDM.

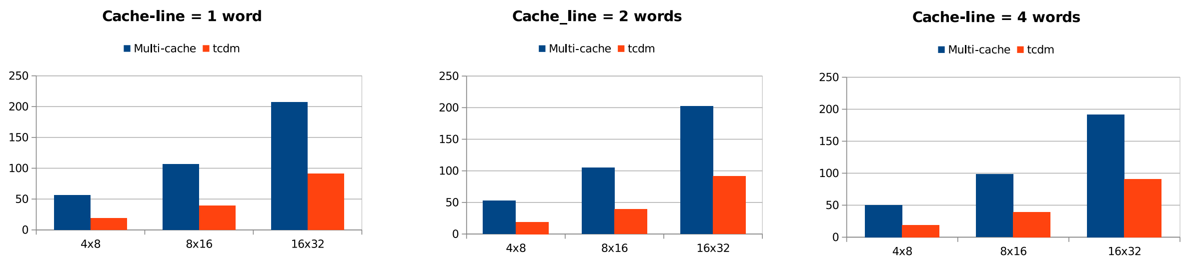

Fig. 6. Power (mW) consumption of multi-banked cache Vs. TCDM.

memory. The wider the cache line the smaller the tag memory. We performed experiments on the architecture with different cache lines and compared the critical path of our design with that of TCDM for 3 different network cardinalities including 4x8, 8x16 and 16x32. Figure 4 shows the related charts for timing analysis.

As it is shown in Figure 4, for all network cardinalities, when the cache line increases the difference between critical path of TCDM and that of our architecture decreases due to the smaller access time of the tag memory. It is clearly seen that for the cache line of 4 words the delay of the critical path in multi-banked cache is almost the same as that of TCDM. For the configuration of 4x8 and cache line of 4 words, the critical path of our architecture is even better than that of TCDM. This is due to the fact that in this configuration delay saving in access time of tag memory compare to the access time of data memory is more than the additional delay imposes by hit/miss logic in cache controller.

B. Area and Power Analysis

We also performed synthesis experiments to compare the power and area of our multi-banked cache architecture with those of TCDM. Figure 5 shows the charts related to area comparison. It is clear that the area of multi-banked cache is more than that of TCDM ($2\times$ for cache line of 1 word and $1.5\times$ for cache line of 4 words) due to extra memories for tags, write-through buffer and cache controller itself. However, the area overhead decreases when the cardinality of the network and the cache line increase. The area overhead for the cardinality of 16x32 and a cache line of 4 words is 36%. We should note that, in these experiments we considered only the area of the network and the cache/memory, but in a real system this is a portion of the whole design. In other words, if we consider the area of a whole cluster including processors then the percentage overhead significantly reduces. For instance, considering Leon3 which is a small-medium sized processor [13] with an area of $0.127mm^2$, the area overhead of multi-banked L1-cache with respect to the TCDM is 18% for the 16x32 configuration. The percentage overhead decreases even more if we consider the area of other components inside a cluster like the peripheral network, network interface, cluster controller and etc.

Figure 6 shows the charts related to the power consumption comparison. As can be seen, the trend for the power consumption is different from that of area. This is due to the power overhead in the write-through buffer of our cache module. In

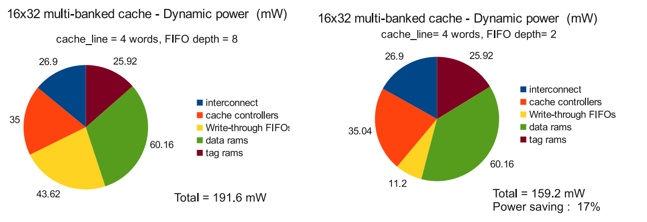

Fig. 7. Break-down of dynamic power consumption in multi-banked cache.

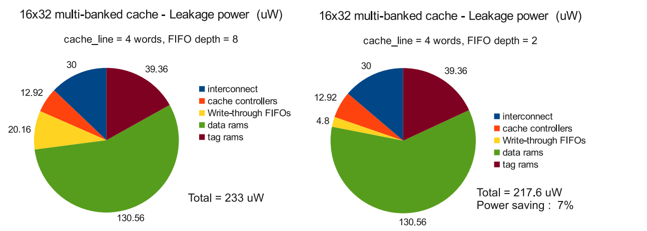

Fig. 8. Break-down of leakage power consumption in multi-banked cache.

these charts the write-through buffer for our cache has a width of 8x64 bits. However, if we reduce the depth of this buffer to 4 or 2 the power overhead decreases.

Figures 7 and 8 show the dynamic and leakage power break-down of multi-banked cache with a cardinality of 16x32. It is shown that around 44% of dynamic power and 21% of static power is consumed in write-through buffers when their depth is 8. However, it can be seen in the right-side of the figures that if we reduce the depth of write-through FIFO to 2, we can save up-to 17% dynamic power and 7% static power consumption for the 16x32 configuration.

We also performed experiments to see the overhead of shared-L1 cache on $MIPS/mm^2$ and $MIPS/watt$ with respect to those of TCDM when we consider processor's area and power. When considering the processor, the power and area overhead will be heavily reduced in percentage terms, while speed remains the same; therefore, the $MIPS/Watt$ and $MIPS/mm^2$ of L1-cache with respect to those of TCDM complete cluster are not so low. We performed this experiment for various processor areas including that of LEON3. Figure 9 shows the $MIPS/watt$ and $MIPS/mm^2$ of L1-cache normalized to those of TCDM for the 16x32 configuration and for different processor areas. As can be seen, as the area of the processors attached to the cluster increases, the overhead of cache with respect to TCDM on $MIPS/watt$ and $MIPS/mm^2$ decreases. This overhead is between 5 and 30 percent depending on the processor size.

IV. Conclusions

We proposed a shared-L1 cache architecture for tightly coupled processor clusters. Inside each cluster a number of processing elements communicate with the shared multi-banked cache through a low latency interconnection network. We explored this architecture in terms of number of processing elements (PEs) and cache banks and analyzed overheads in terms of timing, power and area. Our experiments demonstrate

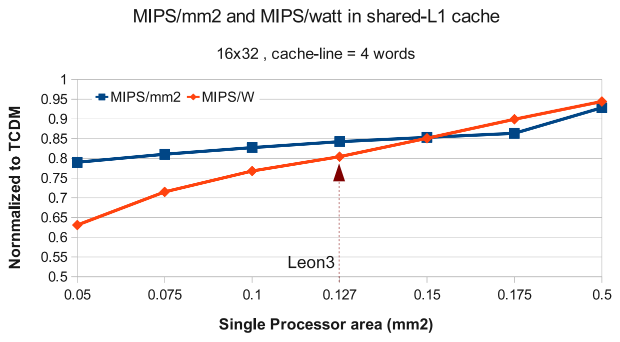

Fig. 9. $MIPS/mm^2$ and $MIPS/Watt$ in 16x32 shared-L1 cache architecture with different processor area attached to the cluster.

that the multi-banked shared-L1 cache can operate with almost the same frequency as an equally sized TCDM architecture if the cache controller uses a cache line of 4 words. We also showed that when considering a complete cluster, the overhead of shared-L1 cache on $MIPS/Watt$ and $MIPS/mm^2$ with respect to TCDM is relatively small for a configuration of 16 medium-sized processors and 32 cache banks. However, we should note that although shared-L1 cache has almost no overhead on the speed with respect to TCDM, architects should also account for the loss in $MIPS/mm^2$ and $MIPS/Watt$ (5%-30% depending on the processor size) coming with the cache.

Acknowledgment

This work is supported by projects: FP7 Virtical (CA 288574) and ERC Multitherman (CA 20110209).

References

[1] L. Benini, E. Flamand, D. Fuin, D. Melpignano, "P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator," Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012 , pp.983-987, 2012.
[2] ITRS Home-2010, 2010 International Technology Roadmap for Semiconductors. [Online]. Available: http://www.itrs.net/home.html
[3] George Chrysos, "Knights Corner, Intel's first Many Integrated Core (MIC) Architecture Product," Hotchips 24, Cupertino, CA, August 2012.
[4] http://techresearch.intel.com/articles/Tera-Scale/1449.htm.
[5] Plurality Ltd. The HyperCore Processor. www.plurality.com/hypercore.html
[6] C.M. Wittenbrink, E. Kilgariff, A. Prabhu, "Fermi GF100 GPU Architecture," IEEE Micro, vol.31, no.2, pp.50-59, March-April 2011.
[7] J. Nickolls, W.J. Dally, "The GPU Computing Era," Micro, IEEE , vol.30, no.2, pp.56-69, March-April 2010
[8] A. Rahimi, I. Loi, M.R. Kakoee, L. Benini, "A Fully-Synthesizable Single-Cycle Interconnection Network for Shared-L1 Processor Clusters ," in Proc. of the ACM/IEEE DATE, 2011.
[9] M.R.Marty, "Cache coherence techniques for multicore processors," PhD Dissertation,University of Wisconsin - Madison, 2008.
[10] Yuang Zhang; et.al , "Towards hierarchical cluster based cache coherence for large-scale network-on-chip," in DTIS , pp.119-122, 2009
[11] D. Hackenberg, D. Molka, W.E. Nagel, "Comparing cache architectures and coherency protocols on x86-64 multicore SMP systems," Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on , pp. 413-422, 2009
[12] Tilera Corp. Product Brief. Tilepro64 processor. 2008.
[13] http://www.gaisler.com
[14] M.R. Kakoee, I. Loi, L. Benini, "A resilient architecture for low latency communication in shared-L1 processor clusters," Design, Automation & Test in Europe (DATE), 2012 , pp.887-892, March 2012.
[15] S. Mitra, K. Brelsford, Kim Young Moon, Lee Hsiao-Heng Kelin, Li Yanjing, "Robust System Design to Overcome CMOS Reliability Challenges," Emerging and Selected Topics in Circuits and Systems, IEEE Journal on , vol.1, no.1, pp. 30-41, March 2011