A Multi Banked - Multi Ported - non Blocking Shared L2 Cache for MPSoC Platforms

Igor LOI* and Luca Benini*†
†ETHZ, Department of Information Technology and Electrical Engineering, Zurich, Switzerland
*DEI, University of Bologna, Bologna, Italy
{igor.loi, luca.benini}@unibo.it

***Abstract*—On-chip L2 cache architectures, well established in high-performance parallel computing systems, are now becoming a performance-critical component also for multi/many-core architectures targeted at lower-power, embedded applications. The very stringent requirements on power and cost of these systems result in one of the key challenges in many-core designs, mandating the deployment of highly efficient L2 caches. In this perspective, sharing the L2 cache layer among all system cores has important advantages, such as increased utilization, fast inter-core communication, and reduced aggregate footprint because no undesired replication of lines occurs. This paper presents a novel architecture for a shared L2 cache system with multi-port and multi-bank features. We target this L2 cache to a many-core platform based on hierarchical cluster structure that does not employ private data caches, and therefore does not require complex coherency mechanisms. In fact, our shared L2 cache can be seen logically as a Last Level Cache (LLC) adopting the terminology of higher-performance many-core products, although in these latter the LLC is more often an L3 layer. Our experimental results show a maximum aggregate bandwidth of 28GB/s (89% of the maximum channel capacity) for 100% hit traffic with random banking conflicts, as a realistic case. Physical implementation results in 28nm Fully-Depleted-Silicon-on-Insulator (FDSoI) show that our L2 cache can operate at up to 1GHz with a memory density loss of only 20% with respect to an L2 scratchpad for a 2 MB configuration.**

I. Introduction

Continuous improvements in technology processes and many-core integration are placing increasing pressure on the speed of memory hierarchies. A wide range of techniques have been developed to reduce or tolerate memory subsystem delays, most prominently the usage of multi-level caches with varying delay parameters. At the same time, memory performance is not the only concern, because with the emergence of multi-core systems in embedded fields, another crucial target is energy efficiency. The performance and energy axes are unfortunately not orthogonal and give rise to a complex set of trade-offs.

State-of-the-art memory hierarchies may include private L1 data and instruction caches, a shared L2 cache, and an off-chip L3. The L2 cache plays a particularly crucial role, because accessing an off-chip L3 is much slower and requires significantly more power, while L1 caches, which are private, must be kept small for latency reasons and because they are instantiated in association with each core. Therefore, the L2 cache hit rates must be maximized to be significantly higher than the L1's, while still accounting for the area constraints of an on-chip implementation. This leads to shared L2 cache designs, which allow for high cache utilization (avoiding duplicating the cache resources), significantly boosting processor performance and extending battery lifetime.

The issue of data coherence between L1 and L2 caches implies a varying degree of performance overhead or hardware cost [1]. While high-performance designs, aimed e.g. at servers, choose to devote area and power in order to implement complex cache coherence mechanisms, this cost is not acceptable in low-power embedded systems. An approach which has gained traction [4], [5] is to simply omit the L1 cache layer, replacing it with an addressable L1 memory bank that must be allocated explicitly by the programmer. Such a memory bank, if efficiently exploited, can provide the same general performance of an L1 cache while reducing the control logic and eliminating the coherence overhead altogether. The price to pay is that of a Partitioned Global Address Space (PGAS) model which is not as transparent as a caching mechanism, thus requiring more manual effort by the programmer. Regardless, this methodology is presently widely accepted in the embedded community for efficiency reasons, and for instance the OpenCL [2] libraries for embedded system endorse these requirements.

Fig 1 shows a typical example of a platform such as the one described above. The many-core system is organized in clusters. In each cluster, the L1 caches are replaced by an intra-cluster explicitly addressable local memory (Tightly Coupled Data Memory - TCDM). The L3 memory is off-chip. For the model to work without complex coherence mechanisms, the L1 TCDM is non-cacheable and mapped to a memory address space disjoint from that of L3 accesses. The lack of a mechanism equivalent to cache refilling in L1s means that data must be explicitly copied into the L1 memories, and off of them after the computation. To this end, each cluster embeds, in addition to multiple Processing Elements (PEs), also one or more Direct Memory Access (DMA) controllers. DMA controllers are the instrument of choice to transfer large chunks of data among the off-chip L3 storage and the working-copy L1 memories; however, due to the system parallelism, the bandwidth requirements towards the L3 storage can quickly become overwhelming and L3 bandwidth saturation can represent the bottleneck for overall system performance. This justifies the addition of an L2 caching layer, which is interposed in accesses to the L3's cacheable address range. The presence of the L2 reduces pressure on the L3, speeds up the perceived L3 access latency, and provides an on-chip shortcut for L1-to-L1 transfers that would otherwise need to transit off-chip. The L2 cache therefore fulfills the role of Last-Level Cache (LLC). The L2 optimizes system performance and therefore also power consumption, both according to the "race to sleep" principle and just by reducing energy-expensive off-chip traffic.

Following this direction, this paper introduces a novel architecture and synthesizable soft IP design for a shared L2 cache system for many-core platforms that exhibits highly non-blocking behavior under concurrent accesses thanks to an internal multi-banked structure and by providing multiple ports towards the chip clusters. Our L2 cache is targeted at the relevant use case of an L1-cache-free many-core platform, where the L2 acts as LLC to speed up off-chip memory accesses. We choose to focus on the design of such an IP, concentrating on how to provide energy efficiency, high throughput and low latency; we do not discuss architectural explorations or specific

978-3-9815370-2-4/13/DATE14/

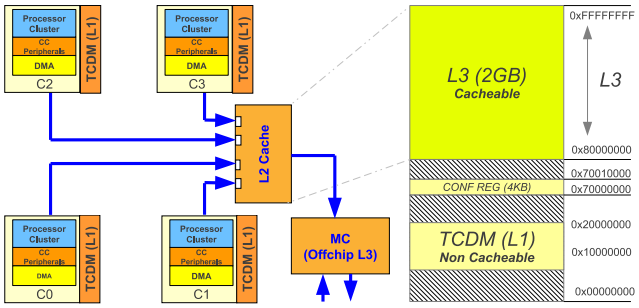

Fig. 1. Reference memory hierarchy architecture for a many-core system: each cluster holds a low-latency, non-cacheable, explicitly-addressable L1 data memory which acts as a scratchpad (Tightly Coupled Data Memory - TCDM). A single L2 cache is shared among clusters, and is interposed to accesses to the L3 Memory space. The L1 is mapped on non-overlapping address ranges with the L2/L3.

optimizations at the policy level, which can be tackled as future work as they can be seen as complementary to this paper.
Experimental results show a maximum aggregate bandwidth of 28 GB/s (89% of the maximum channel capacity) for 100% hit traffic with random banking conflicts. Physical implementation results in 28nm Fully-Depleted-Silicon-on-Insulator (FDSOI) show that our L2 cache can operate at up to 1 GHz with a memory density loss of only 20% with respect to an L2 scratchpad for a 2 MB configuration.

II. Related Work

While the PC and server markets gradually evolve from four to six or eight massive x86 cores, the rest of the world is moving in a different directions: many-cores with large numbers of relatively simple CPUs and with stringent energy requirement. For such systems the state of the art is to build a memory system based on private L1 caches, and shared L2 cache as Last level Cache LLC leaving the L3 off-chip. Cavium Octeon II CN68XX [8] is a cluster-based many-core platform composed by 16 or 32 MIPS64 v2 cores. Each processor has its private L1 cache, and globally they are grouped in four clusters. The L2 cache is multi-bank and shared among cluster. Each L2 Bank has its independent memory controller for off-chip DRAM DDR3.
The Blue Gene/Q chip [9], the basic processing element for the next generation of supercomputers, has similar characteristic with respect to Cavium Octeon, but cores embed large and power hungry FPUs, shared L2 cache is much larger (32MB is 16 banks) and implemented with embedded DRAM. Data coherency is based on mutual-exclusion lock instruction and on an additional hardware block that watches the bus and detects deadlock.
Several mobile platforms have been introduced in recent years. The Exynos 5 Octa [10] by Samsung is a multi-core based on ARM A17-A7 big.LITTLE for smart-phones and tablets. It is composed by a first quad-core optimized for performance, and a second quad-core to maximize energy efficiency. The former features a 2MB private L2 cache, while the latter only 512 kB. Depending on the required performance, tasks are moved from one quad-core to the other and vice versa.
Multi-bank caches have been widely adopted to increase the cache bandwidth. In [11] authors analyze the best trade-off for several cache bank interleaving granularities in case of Non Uniform Cache Architecture (NUMA) for different corners. The authors investigated pros and cons of private and partitioned shared LLCs but with a fixed number of cache banks.
In architectures like the Cell B.E. [3] or GPUs, the PEs incorporate local memories which are fed with data transferred from memory using Direct Memory Access (DMA) controllers. A similar approach has been adopted in P2012/SThorm [4] and Adapteva Parallella [5]. The main concern for these platforms is maximize the GOPS/W. P2012 is a many-core platform based on four clusters of PEs, each hosts sixteen ST xP70 processors. Adapteva Parallella offers 16 or 64 processors, grouped in 4 clusters as well, and both SoCs adopt for the L1 memory subsystem an intra-cluster shared L1 scratchpad. At level 2, Adapteva Parallella has an L2 cache, while STMicroelectronics P2012 has 1 MB scratchpad.

III. Many-core system architecture

This section presents the overall memory hierarchy where our L2 cache is deployed. Fig 1 shows the memory architecture, where each cluster (C0 to C3) includes 16 PEs that share L1 memory. L1 is as a scratchpad therefore is not cacheable. Since L3 is physically off-chip (external DRAM), the L2 cache can be used to cache L3 accesses, therefore boosting the performance when accessing offchip memory. In this type of platform, the L2 represent the LLC, and since L1 is not cacheable, there is no need to support coherency mechanism. This memory architecture knows as Partitioned Global Address Space (PGAS) is becoming a common design choice for many-core embedded accelerators, and examples comes from STm "P2012/STHORM" [4] and Adapteva "Parallella" E16G301 and E64G401 [5]. In [4], the L3 traffic comes from the 4 clusters, and each one includes 16PEs and two dual-channel DMAs. In total there are 65 processors and 8 dual channel DMAs making accesses to external memory, therefore the main role of the L2 cache is to reduce average latency for L3 accesses, while at the same time reducing the bandwidth toward the L3 memory controller.
In case the clusters employ L1 caches in place of TCDMs, the L2 cache can be extended to support a snoopless cache coherency protocol, such as the recently-proposed VIPS protocol [6], [7].

IV. L2 Cache System Architecture

This section will focus on the main features of our L2 multi-bank and multi-port shared cache. As depicted in Fig 2 the cache subsystem is composed by several L1 ports, used by Processor Elements (PEs) or cluster of PEs to assert L2 memory accesses, a request interconnect to multiplex incoming requests, then multiple cache banks that are interleaved by cache lines (to better spread accesses), a configuration block to program the cache, and finally a collector interconnect, to canalize all the requests directed to L3.
Since shared cache performance is critical when several PEs make memory access, we adopt a flexible interconnect protocol, namely STBus type T3 which is fully pipelined and supports burst operations and multiple outstanding transactions. Thus, the L2 cache banks have been natively designed to support advanced, non-blocking on-chip communication protocols which can be transported over a network-on-chip infrastructure (STm STBus). The ratio between the number of cache banks and master ports (L1) is called Banking Factor (BF) and to reduce the probability of bank collision this number should be greater than 2. In our reference architecture we selected a BF of 2, in order to have a good trade-off between cost and performance, even though our soft IP allows different banking factors.

A. Protocol and Interconnect

As introduced at the beginning of section IV, the ports from L1 and to L3 are compliant with the STBus T3 specifications, therefore the whole L2 protocol is capable to exploit the maximum performance for a given workload. The configuration port, since it used merely to reconfigure the cache and for

debug activity, it does not need strict bandwidth requirements, therefore we opted for a peripheral T1 STBus protocol.
The request interconnect that is depicted in Fig 2, right after the L1 ports, is used to multiplex the incoming requests, and the routing criteria is based on cache line interleaving. Since type T3 supports up to 64 byte load/store operations, we fixed the upper bound for the cache line to 64 byte (so in any case, a single burst will hit only one cache bank), leading to a bank interleaving granularity of 64 byte. The latency of the request interconnect is one cycle (in both directions) and it includes 4 small FIFOs attached on each L1 port.

B. Configuration Block

The configuration block is used to program the cache features and to debug errors or collect statistics. It is composed by registers mapped on a 4KB memory range, as depicted in Fig 3. This block is reachable both from the initiator L1 ports, and from the dedicated configuration port. This double option, allows both PEs and Host (in case the cache is initialized from the host system) to program the L2 cache.
The configuration registers provide both information about the hardware (such as version, cache line width, set-id size, number of ways etc), and give the access to global pins used to bypass or configure the cache.
The main configurable features are the following:

- Region decoding: these registers specify up to 4 regions with a particular behavior. Through start and stop address, and region type, the user can choose between 4 types of L2 accesses: UnCached (UC), Write Through (WT), Write Back with Read Allocate (WBRA) and finally Write Back Read and Write Allocate (WBRWA). Fig 3 shows an example of 4 non adjacent regions mapped with 3 different behavior.
- Lock: a lock attribute can be placed over a set-id/way, or on the whole way. This avoid data to be evicted, or in general can be set to use the L2 cache as scratchpad.
- Flush/Invalidate/Purge: These operations allow to flush/invalidate or purging a particular set-id/way or the whole L2 multi-bank cache.
- Error tracking and statistics: Address alignment and other internal errors are tracked internally on each cache bank. Hit and miss counters are also available to track performance.

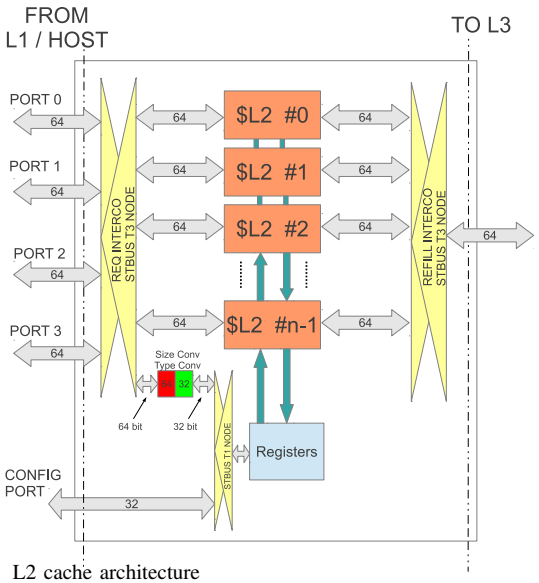

Fig. 2. L2 cache architecture

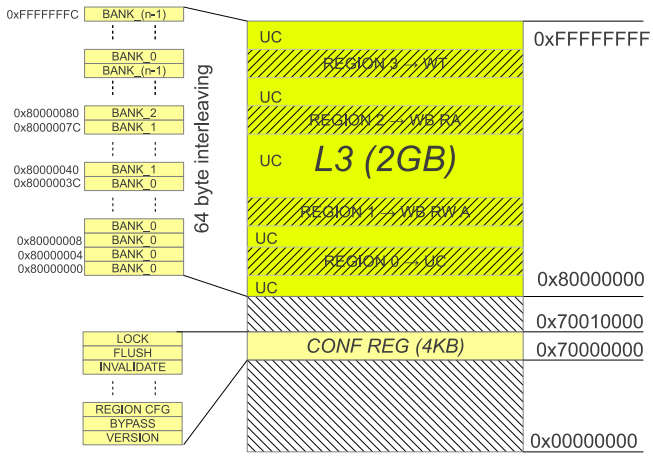

Fig. 3. Physical memory map: L2 cache is mapped on L3 region (upper 2GB)

C. L2 Cache Bank

A single cache bank is replicated several times to form a multi-bank cache. Banks are interleaved in the memory space (at cache line granularity), therefore each address is unequivocally mapped on specific bank, and globally the multi-bank cache covers the entire L3 space. Having more than on bank reduces the probability to have collisions, and the main benefits is the ability to provide more bandwidth at the expense of additional hardware resources. The single bank is highly flexible and can be configured at compile time to tune the following parameters: 1) cache size, 2) set associativity and 3) size of internal buffers. Other parameters can be switched at run time (as illustrated in section IV-B).

V. Cache Bank Architecture

In many-core system it is unimaginable to employ caches with blocking behavior, therefore our cache has been designed to be fully pipelined and to process data at full speed. Fig 4 depicts the block diagram for a single bank in a two-way set associative configuration. Transactions from L1 are injected through the initiator port, while evictions, refills, write-through and un-cached accesses to L3 are asserted from the target port. The Cache Controller (CC) conducts the thorniest task: by means of dedicated pipe it decodes the addresses in 3 cycles to perform hit/miss checks, to update the valid/dirty bit, to assert evictions and refills and to route back the responses. Since CC processes only atomic transactions, we introduced two atomizers and two de-atomizers.

A. Transaction Atomizer

STBus type T3 supports several burst granularities, ranging from 1 byte to 64 bytes. Since the interconnect data-width in our case is 64 bit, and the maximum supported burst is 64 byte, the transaction atomizer packs together up to 8 cells [1] of the packet [2], in a single “atomic transaction”. Atomization is a mandatory requirement that packs together cells that belong to the same cache line. This requirement allows to reduce the amount of internal traffic in the CC, and at the same time simplifies the CC design. The atomizer is composed by a configurable input FIFO, a deserializer (output data-width is 64byte) and a small output FIFO (needed to cut

[1] A cell is the basic amount of information that can be transferred within a clock cycle

[2] A packet is a collection of cells; Eg. the store 64 byte is a packet made up 8 cells for a 64bit data-width

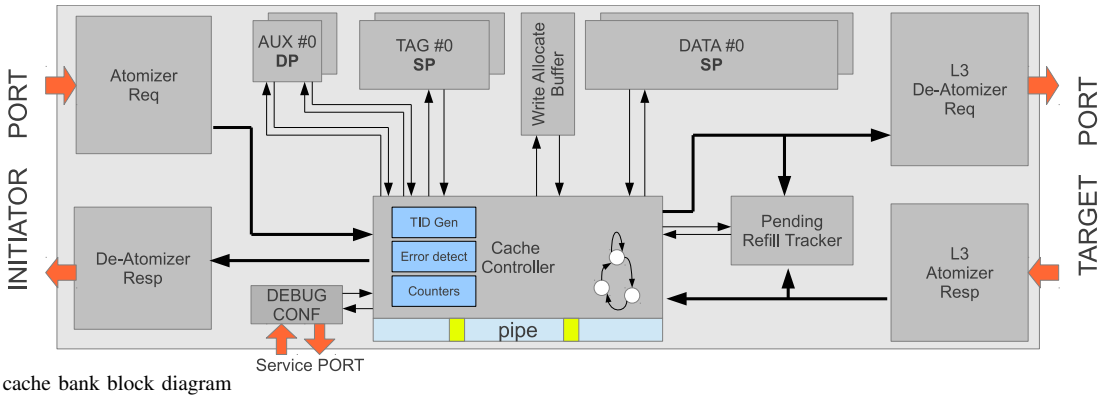

Fig. 4. Single cache bank block diagram

the internal critical paths). This block is also used to pack the responses from the L3 target ports, while on the target request (L3) and initiator response (L1) port, we implemented the "de-atomizer", which exploits the serialization of atomic transactions.

B. Cache Pipeline

The pipeline is controlled by CC, and processes incoming transactions at full speed, with a latency of 3 cycles. Fig 5 illustrates the main actions exploited by this block.

1) A transaction reaches the pipe in STAGE_0, then the TAG RAM is prepared to read in parallel all the TAGs (n ways).
2) In STAGE_1, the address (latched the in STAGE_0) is compared in parallel with the output of the TAG RAMs. A match vector indicates hit or miss, and in case of hit, it identifies the hitting way. In case of miss the CC uses the information available in this stage to perform the right operation, while in case of hit, it prepares the data RAM to be written/read in the following cycle.
3) Only in case of hit, STAGE_2 holds the information needed to send back the response to the request "de-atomizer".

The pipe also samples all the incoming signals (STAGE_1 and STAGE_2) to be used in the last stage in case of hit, or in the first stage in case of miss.

C. Cache Controller

The cache controller manages L1 memory accesses with different policies, provides statistic information, and tracks internal errors. It supports uncached access, write through, write back with read and write allocate. The cache line replacement policy is pseudo-random, and a lock mechanism is also implemented to prevent evictions of critical data. CC manages all the TAG, DATA and AUX RAMs, write buffer and pending refill tracker as illustrated in Fig 4. When cache is bypassed or in case of UC accesses, transactions still flow through the pipe, but no action is taken internally. In WT and WB mode the CC annotates the cache line status in a dedicated memory called "AUX RAM" (see Fig 4). This is a dual port SRAM, and for each cache line it stores the following

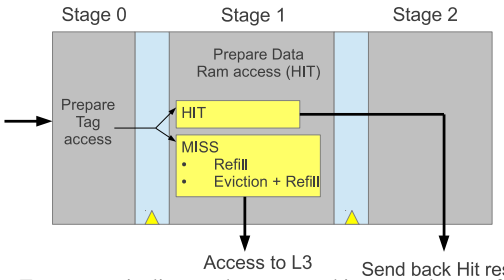

Fig. 5. Two stage pipeline used to process hit transactions at full speed

information: (1) Valid cache line, (2) Dirty bit (one dirty bit each 64 data bit), (3) Lock bit and (4) Pending refill.
In case of miss, depending on the status of the cache lines (n. ways) for a given set-id, the CC asserts a refill (or eviction followed by refill) or a direct L3 access (eg for write through). The cache line selection or replacement is taken during the refill request phase, and these information (target way, set-id and TAG), are stored in the pending refill buffer, which is capable to track up to 16 pending refills. When the response comes back from L3 memory, the CC packs the response retrieving the target information (which way, set-id and TAG to use) from the pending refill tracker. All the other types of L3 accesses do not need to be tracked. If a lock attribute is set on a cache line, then in case of line deallocation, the CC skips all the locked lines. Table I resumes the whole parameters that can be switched at run time and compile time.

Compile-time parameters		Run-time Parameters	
TYPE	RANGE	TYPE	RANGE
Associativity	2-4-8-16	Region Dec.	UC, WT, WBRA, WBRWA
Cache line	8-16-32-64-128 B		
Cache Size	Any (power of 2)		
Internal FIFOs	≥ 2		
Number of banks	Any (power of 2)		
Number of inputs	Any		

TABLE I. RUN TIME AND COMPILE TIME PARAMETERS TABLE

VI. SIMULATION FRAMEWORK

Performance comparison studies are primarily carried out through simulations. Since our shared L2 cache is described at RTL and no high level model has been developed to be integrated in virtual platforms, we built a cycle accurate test-bench to allow performance measurement and characterization. Fig 6 shows the top level implementation of our simulation framework. First, memory transactions are generated inside the traffic generators (TGENs written in verilog as well), and then routed to the cache through intra-cluster STBus nodes. These memory transactions are representative of miss traffic of the L1 caches that are embedded inside the TGENs. Therefore we model a multi-core system, where 64 TGENs are grouped in 4 clusters (16 TGENs per cluster), and each cluster multiplexes the requests in one output port which is connected to one of the L2 cache initiator port. The target port of the cache is connected to L3 memory, that can be tuned to emulate several L3 latencies.

Fig 6 also depicts the overhead for a miss in L2. In case of hit, the transaction that crosses the cluster and reaches one L2 cache is first arbitrated in the initiator node and "atomized", then flows through the L2 pipeline, and in case of hit (STAGE_2), the response is fed to the "de-atomizer" (as highlighted with the green line). Finally the packet is arbitrated through the STBus initiator node and back-routed to the right TGEN. In case of miss there is an additional overhead which is marked with red dotted line. It includes the "de-atomization" the arbitration through the target STBus node,

the complete elaboration in the L3 subsystem (eg, could be a DRAM controller with off-chip DRAM), then the response ”atomization“ and cache line update (TAG + DATA + AUX).

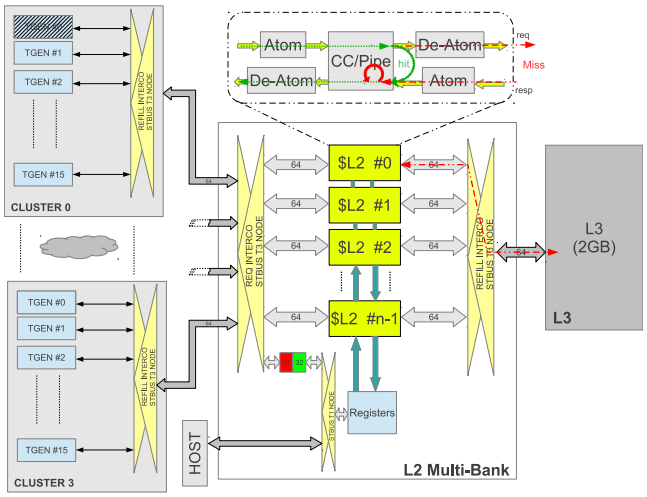

Fig. 6. Overview of the simulation framework used to test and to collect statistics

VII. Performance Analysis

Detailed cycle-accurate simulation has been performed with Mentor Graphics ModelSim. Two types of traffic have been generated:

- Synthetic: used to characterize bandwidth and latency in extreme traffic corners in a controlled way.
- Traces: generated with a virtual platform with application benchmarks, mainly useful to show how the cache behaves under typical application traffic.

Two types of synthetic traffic have been generated: the former for min-max hit latency and bandwidth measurement and for different demanded bandwidth, the latter for miss bandwidth measurement with several miss ratios. Fig 7 shows the cache characterization with synthetic traffic. Fig 7a illustrates the average cache hit latency (load 64 byte, which is the most important and probable) and standard deviation for different bandwidth demand corners. The first bar represents the latency for unloaded cache and without any bank contention. Note that we achieved a latency of 16 cycles for a load 64 byte, and this value scales to 9 cycles for a load/store 8 byte. Starting from the second bar, we increased the injected bandwidth in several steps, until we reached the maximum demanded bandwidth of 100%. Since transactions are generated randomly, there is probability of banks conflict that lead to increase both the average latency and its dispersion. In the last bars of Fig 7a, we measured the collision overhead that is imputable to arbitration contentions both request and response path. The last bar represents an unrealistic case where each initiator port always hits the same bank (BANK_0), with the maximum bandwidth demand (100%), as a result transactions are always stalled (with a ratio of 1:4) leading to saturate all the buffers therefore increasing both latency and its dispersion.
Fig 7b is linked with the experiments in Fig 7a, and depicts the bandwidth efficiency (expressed as ratio between demand and available bandwidth). The plot shows that the cache is capable to supply the requested bandwidth (in case of multicore traffic) up to 80% [3] of the interconnect aggregated capacity; after this point there is a degradation that leads to an average throughput of 89% under peak load (see Fig 7b). 100% throughput is only achievable under synthetic traffic generating no collisions in the initiator STBus node.
Fig 7c illustrates the average throughput under peak load for cache accesses with a variable miss ratio. We swept the miss ratio from 0% to 100% (for Load 64 to load 8 byte) in two corners: with fast (5 cycles) and slow L3 (100 cycles). Bars represent the maximum available bandwidth and in each configuration we can identify a breaking point between 20% and 30% (sixth and seventh bars) that leads to a drastic bandwidth reduction.
The second type of traffic are traces. GEM5 [12] simulation environment has been used to generate L2 memory accesses, (back-annotated on text file and then fed to our TGENs). GEM5 runs a full-system simulation of Alpha CPUs (we simulate 16 processors with 32KB instruction and data caches) with Linux 2.6.27 kernel executing PARSEC V2.1 benchmark suite [13] on medium sized inputs, and a fixed number of traces are gathered only for the parallel part of the benchmarks when they enter their Region of Interest (ROI). We programmed our testbench to execute in parallel all the traces, allocated in 4 clusters (4 traces per cluster). Fig 8 shows for each benchmark both an average bandwidth requirement and the miss ratio for a 2MB L2 cache. Most of the accesses are load 64 byte, and miss ratio average is 20% (ranges from 9% to 40%). For these benchmarks the demanded bandwidth is far from the maximum channel capacity, this implies that the cache can easily support realistic application traffic and that it could also be used for much more demanding traffic generators than general-purpose cores (e.g. multiple DMA engines or GPU-like cores).

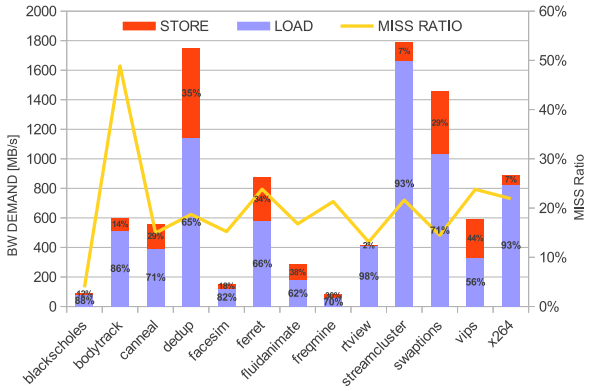

Fig. 8. Performance and average bandwidth requirement for several benchmarks

VIII. Physical Implementation

This section presents the post place&route results and the comparison of our L2 memory with and without the cache mechanism (called “L2 RAM”). Our static L2 is featured (similar to the cache) by 4 initiator ports, the multi-bank mechanism, and implements the same protocol (STBus) and data-with of the cache. Our design flow is based on the STm FDSoI CMOS-28nm technology library, with hierarchical synthesis methodology with Synopsys Design Compiler Graphical (2013.03), and place and route with Cadence Encounter Digital Implementation (11.1). We implemented 4 different configurations for both memory types: 2MB, 1MB, 512KB and 256KB. Cache is composed by 8 banks while the “L2 RAM” scales from 256 to 32 banks. Fig 9 illustrates power, area and performance metrics. The vertical bars represent the area cost, and as can seen, “L2 RAM” is mainly composed by the memory hard macros (Single Port). Memory density for such implementations ranges from 93% to 88% since these configurations are simply scratchpads. Caches add significant overhead both with the AUX-TAG and with the logic for routing, buffering and for reconfiguration. Density for this

[3] with maximum efficiency

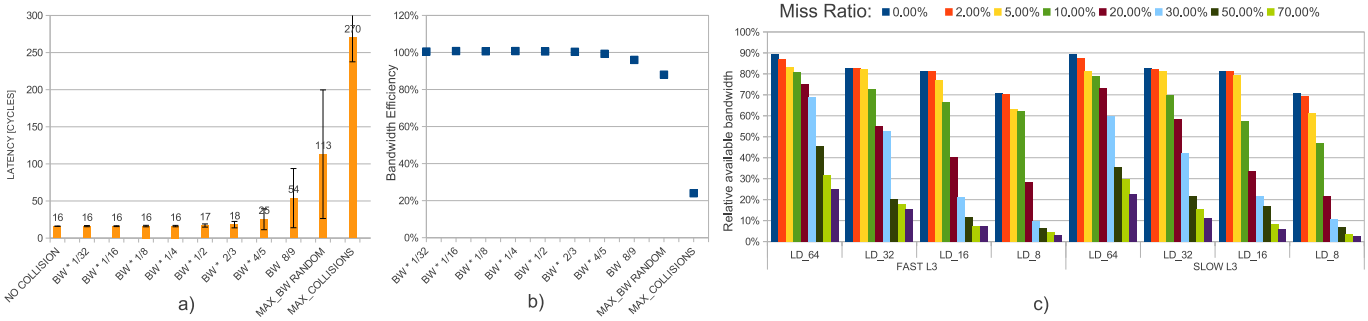

Fig. 7. Latency and bandwidth distribution: in a) it is plotted the average cache hit latency (load 64 byte) and standard deviation for different bandwidth demands corner. b) is linked with a) experiment and shows the bandwidth efficiency for cache hits. Efficiency is represented as ratio between demanded and available bandwidth. In c) we kept constant the bandwidth demand (at its maximum for Load 64 to load 8 byte) and we swept the miss ratio from 0% to 100% in two corners: with fast (5 cycles) and slow L3 (100 cycles). Bars represent the maximum available bandwidth

category ranges from 75% to 61%. A cross memory density comparison (between cache and scratchpad) gives a density loss that ranges from 20% to 30%. Fig 9 also depicts the power trend for a switching activity with 5% of toggle rate. The main power overhead in this case is due to the additional memory macros for AUX and TAG (that increases its bit-width when scaling down the cache size), and the buffering resources. The 2MB cache shows $2\times$ power consumption (at the maximum frequency) with respect to the scratchpad, while the 256KB version gives a penalty of $7.5\times$. Finally Fig 9 reports the maximum frequency for each configuration. The cache critical path is bound by the access and setup time of the memories in case of Hit. In that case, the tag is read, compared and in case of hit the data is prepared to perform the read or write in the data RAM, therefore the sum of access time of tag RAM plus the setup of the data RAM are the main limiting factor for the maximum speed.
Fig 10 shows the physical implementation for the 2MB L2 cache configuration. On the left side the top view of the multi-bank cache is reported, while on the rigth side the single cache bank is shown. Physical sizes of top and block depend on the target utilization, and in our case we chose 70% to reduce wire congestion. Data RAMs are grouped in four different ways (associativity), while TAG and AUX are kept close because their relative position is critical for timing.

IX. Conclusion

In this paper, we presents a synthesizable multi-bank, multi-port and non blocking shared L2 Cache IP which could be attached to a cluster-based multi-core platform through its STBus ports offering high-bandwidth memory access with a low latency (9-16 cycles). Both benchmark and synthetic simulation results demonstrated that our cache can offer a maximum aggregate bandwidth of 28GB/s (89% of the maximum channel capacity) for 100% hit traffic with random banking conflicts, as realistic case. Physical implementation results showed that our cache can operate at up to 1GHz in STMicroelectronics FDSoI 28nm technology (bound by memory access time) with a memory density loss that ranges from 20% to 30%, and with a power consumption that ranges from 112mW to 80 mW (respectively for the 2MB and 256KB version).

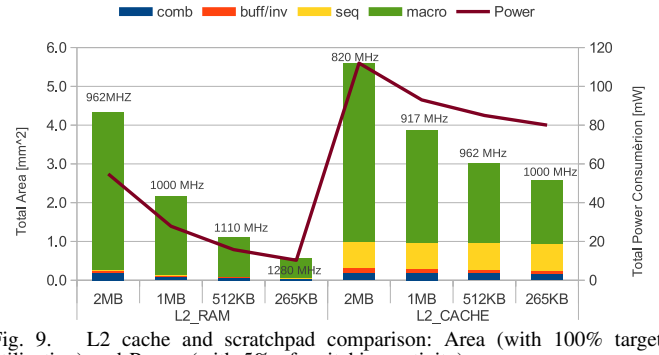

Fig. 9. L2 cache and scratchpad comparison: Area (with 100% target utilization) and Power (with 5% of switching activity).

X. Acknowledgments

This work was supported by ERC-AdG MultiTherman (291125), funded by the European Community.

References

[1] Milo M. K. Martin et al., “Why on-chip cache coherence is here to stay”, in Communications of the ACM, vol 55, no. 7, pp. 78-89, July 2012
[2] The Khronos OpenCL Working Group, OpenCL - The open standard for parallel programming of heterogeneous systems, http://www.khronos.org/opencl/, Feb. 2011.
[3] Cell Broadband Engine, “https://www-01.ibm.com/chips/techlib/techlib.nsf/products/Cell_Broadband_Engine”
[4] D. Melpignano et al., “Platform 2012, a many-core computing accelerator for embedded SoCs: performance evaluation of visual analytics applications” in DAC ’12, pp. 1137-1142
[5] Adapteva Parallella, “http://www.adapteva.com/”
[6] Alberto Ros et al., “VIPS: Simple Directory-Less Broadcast-Less Cache Coherence Protocol”, 2011
[7] Stefanos Kaxiras et al., “EFFICIENT, SNOOPLESS, SYSTEM-ON-CHIP COHERENCE”, in proceedings of SOCC 2012, pp. 230-235, Sept. 2012
[8] OCTEON II CN68XX Multi-Core MIPS64 Processors. “http://www.cavium.com/OCTEON-II_CN68XX.html”
[9] R.A. Haring et al., “The IBM Blue Gene/Q Compute Chip”, Published in IEEE Micro, vol. 32, no. 2, pp 48-60, April 2012
[10] Exynos 5 Octa “www.samsung.com/exynos/”
[11] A. Vega et al., “Comparing Last-level Cache Designs for CMP Architectures”, in Proceedings of the Second IFMT 10
[12] N. Binkert et al., The gem5 simulator, SIGARCH Comput. Archit. News, vol. 39, no. 2, pp. 17, Aug. 2011.
[13] C. Bienia and K. Li, Parsec 2.0: A new benchmark suite for chip-multiprocessors, in Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation, June 2009.

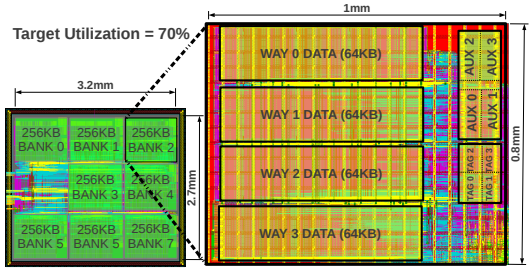

Fig. 10. Multibank and single bank cache layout in 28nm FDSoI technology