Seventh Framework Program –Theme 3.6: Computing Systems

Grant Agreement n° 248 776

D1.4

"3D Memory Hierarchies – Modelling & Characterisation"

WP1: "From 3D Opportunities to 3D Manycore Architectures"

Editor: Martino RUGGIERO (UNIBO)

Date: Tuesday, 11 October 2011

1	CEA (coord.)	Commissariat à l'énergie atomique et aux énergies alternatives – Laboratoire des technologies de l'information.
2	UJF	Université Joseph Fourier, Grenoble I – VERIMAG
3	ETHZ	Eidgenassische Technische Hoschschule Zürich
4	UNIBO	Universtità di Bologna
5	STM	STmicroelectronics
6	EPFL	École polytechnique fédérale de Lausanne

Version: v3.2 – Status: Delivered

CEA Ref.: LETI/DACLE/11-0653

File: "`pro3d-d1.4-3d-memory-hierarchies-v3.2.doc`"

Versions of the Document

Version	Date	Author	Comment
v0.1	2011-09-21	Martino RUGGIERO	First draft.
v0.2	2011-09-23	Daniele BORTOLOTTI	L1 Section.
v2.0	2011-09-23	Martino RUGGIERO	Second draft.
v3.0	2011-09-29	Martino RUGGIERO	Third draft.
v3.1	2011-09-30	Martino RUGGIERO	Final draft.
v3.2	2011-10-10	Christian FABRE	CEA document tagging and minor adjustments.

Contributors

Martino RUGGIERO (UNIBO), Daniele BORTOLOTTI (UNIBO),
Andrea MARONGIU (UNIBO), Igor LOI (UNIBO), Luca BENINI (UNIBO).

Table of Contents

Table of Figures

1 Introduction

The main platform architecture developed in PRO3D project is depicted in Figure 1. It can be considered a derivation of the P2012 platform template, described in D7.1.

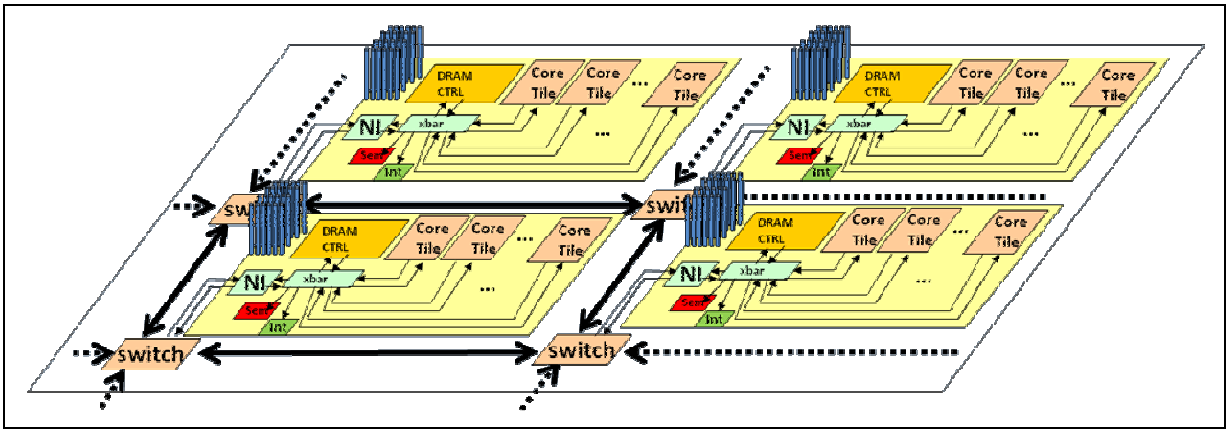

Figure 1: PRO3D Target Architecture

STMicroelectronics Platform 2012 (P2012) is high-performance architecture for computationally demanding image understanding and augmented reality applications [5]. P2012 architecture is composed by several processing clusters interconnected by a Network on Chip (NoC).

P2012 features a complex memory hierarchy organization which comprises several levels and memory types. L1 level is the closest to the processing element and can be accessed locally by its embedding processing element but may also be accessed by non local processing elements with an increased access time. L2 level is an intermediate level typically used for storing data which are shared by several processing elements of the same cluster. L3 is a large memory that can be accessed by all the clusters of the fabric. The possible implementation of level L3 is a central 3D stacked memory. Further details can be found in [34].

Clearly, the organization of memory hierarchy is one of the most important and critical phases in manycore architecture design. Dealing with massively parallel systems such as P2012, instruction caching, data memory and DRAM interfacing play a fundamental role since they must provide the required bandwidth to all cores and software tasks, complying with tight constraints in terms of size and complexity.

The present deliverable reports the work done in WP1, which is focused in deploying the modelling infrastructure to simulate thermal and functional features of the PRO3D stacks.

The following sections describe our modelling approaches and provide the technical details on the implementation work that has been conducted for developing and characterizing the memory hierarchy in the proposed PRO3D platform.

More in details, Section 2 focuses on the L1 memory subsystem. Section 3 discusses the DRAM subsystem for 3D integrated SoCs. Finally, Section 5 highlights the main conclusions.

2 L1 memory subsystem

To keep the pace of Moore's law, several Chip-Multiprocessors (CMP) platforms are embracing the manycore paradigm, where a large number of simple cores are integrated onto the same die. Current examples of many-cores include GP-GPUs such as NVIDIA Fermi [1], the HyperCore Architecture Line (HAL) [3] processors from Plurality, or **ST Microelectronics Platform 2012** [5].

All of the cited architectures share a few common traits: their fundamental computing tile is a tightly coupled cluster with a shared multibanked L1 memory for fast data access and a fairly large number of simple cores, with ≈ 1 Instruction Per Cycle (IPC) per core. Key to providing I-fetch bandwidth for a cluster is an effective instruction cache architecture design. Due to the lack of sophisticated hardware support to hide L2/L3 memory latency (e.g. prefetch buffers), the simple processors embedded in many-cores may indeed experience prolonged stalls on long-latency I-fetch.

The Fermi-based General Purpose Graphic Processing Units (GPGPU) comprises hundreds of Streaming Processors (SP) organized in groups of Streaming Multiprocessors (SM) [1]. The numbers of SMs and SPs per device vary by device. GPGPUs employ massively multi-threading in order to hide the latency of main memory. The GPU achieves indeed efficiency by splitting application workload into multiple groups of threads (called warps) and multiplexing many of them onto the same SM. When a warp that is scheduled attempts to execute an instruction whose operands are not ready (due to an incomplete memory load, for example), the SM switches context to another warp that is ready to execute, thereby hiding the latency of slow operations such as memory loads. All the SPs in an SM execute their threads in lock-step, according to the order of instructions issued by the per-SM instruction unit. SPs within the same SM share indeed one single instruction cache [2].

Plurality's HyperCore Architecture Line (HAL) family includes 16 to 256 32-bit RISC cores, shared memory architecture, and a hardware-based scheduler that supports a task-oriented programming model [3]. HAL cores are compact 32-bit RISC cores, which execute a subset of the SPARC v8 instruction set with extensions. The memory system is composed by a single shared memory which operates also as instruction cache. The shared memory holds indeed program, data, stack, and dynamically allocated memory. Each core has two memory ports: an instruction port that can only read from memory, and a data port that can either read or write to memory. Both ports can operate simultaneously, thus allowing an instruction fetch and a data access by each individual core at each clock cycle. The processors do not have any private cache or memory, avoiding coherency problems. However, conflicting accesses cannot be avoided causing latency increasing for not-served requests [4].

All of the cited platforms have adopted different instruction cache architectures, meaning that **there is still not a dominant paradigm for instruction caching in the many-core scenario**. Clearly, a detailed design space exploration and analysis are needed to evaluate how micro-architectural differences in L1 instruction cache architectures may affect the overall system behavior and IPC.

2.1 Shared L1 Cluster Architecture

This section provides description of the building blocks of both private and shared instruction cache architectures. To help system designers to compare different L1 instruction cache architectures, we have developed a flexible L1 instruction cache architecture system. The proposed templates, written in SystemC [8], can be used either in stand-alone mode or plugged into PRO3D virtual platform [7]. Our enhanced virtual platform is highly modular and capable of simulating at cycle-accurate level an entire shared L1 cluster including cores, L1 instruction caches, shared L1 tightly coupled data memory, external (L3) memories and system interconnections.

2.1.1 Processing Elements

Our shared L1 cluster consists of a configurable number of 32-bit ARMv6 processor (ARM11 family [10]). There are several ARMv6 instruction set simulators already available, Skyeye [11], SoClib [12] and SimSoc [9] are just a few representative examples. We chose the one in [9] as our base ISS. To obtain timing accuracy, after modifying its internal behavior to perform concurrent load/store and instruction fetch, we wrapped the ARMv6 ISS in a SystemC module.

2.1.2 L1 Instruction Cache module

The Instruction Cache Module has a core-side interface for instruction fetches and an external memory interface for refill. The inner structure consists of the actual memory (TAG + DATA) and the cache controller logic managing the requests. The module is configurable in its total size, associativity, line size and replacement policy (FIFO, LRU, random).

2.1.3 Logarithmic Interconnect

The logarithmic interconnect module has been modelled, from a behavioural standpoint, as a parametric, Mesh-of-Trees (MoT) interconnection network to support high-performance communication between processors and memories in L1-coupled processor clusters resembling the hardware module described in [13], shown in Figure 2.

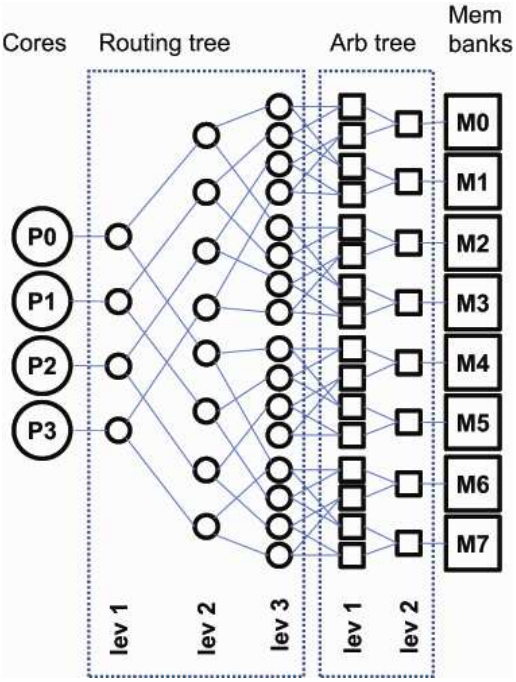

Figure 2: Mesh of trees 4x8: empty circles represent routing switches and empty squares represent arbitration switches (banking factor of 2)

The module is intended to connect processing elements to a multi-banked memory on both data and instruction side. Data routing is based on address decoding: a first-stage checks if the requested address falls within the intra-cluster memory address range or has to be directed off-cluster. To increase module flexibility this stage is optional, enabling explicit L3 data access on the data side while, on the instruction side, can be bypassed letting the cache controller take care of L3 memory accesses for lines refill. The interconnect provides fine-grained address interleaving on the memory banks to reduce banking conflicts in case of multiple accesses to logically contiguous data structures. The last *log2 (interleaving size)* bits of the address determine the destination. The crossing latency consists of one clock cycle. In case of multiple conflicting requests, for fair access to memory banks, a round-robin scheduler arbitrates access and a higher number of cycles is needed depending on the number of conflicting requests, with no latency in between. In case of no banking conflicts data routing is done in parallel for each core, thus enabling a sustainable full bandwidth for processors-memories communication. To reduce memory access time and increase shared memory throughput, read broadcast has been implemented and no extra cycles are needed when broadcast occurs.

2.1.4 L1 Tightly Coupled Data Memory

On the data side, a multi-ported, multi-banked, Tightly Coupled Data Memory (TCDM) is directly connected to the logarithmic interconnect. The number of memory ports is equal to the number of banks to have concurrent access to different memory locations. Once a read or write requests is brought to the memory interface, the data is available on the negative edge of

the same clock cycle, leading to two clock cycle latency for conflict-free TCDM access. As already mentioned above, if conflicts occur there is no extra latency between pending requests, once a given bank is active, it responds with no wait cycles. Banking factor (i.e. ratio between number of banks and cores) can be configured to explore how this affects banking conflicts.

2.1.5 Synchronization

To coordinate and synchronize cores execution, we modelled two different synchronization mechanisms. The first one consists of HW semaphores mapped in a small subset of the TCDM address range. They consist of a series of registers, accessible through the data logarithmic interconnect as a generic slave, associating a single register to a shared data structure in TCDM. By using a mechanism such as a hardware *test&set*, we are able to coordinate access: if reading returns '0', the resource is free and the semaphore automatically locks it, if it returns a different value, typically '1', access is not granted. This module enables both single and two-phase synchronization barriers, easily written at the software level. Theoretically all cores can be resumed at the same time (reading broadcast the value of the semaphore), but there is no guarantee that this happens because of execution misalignment. To get tight execution alignment, we developed two fast synchronization primitives based on a HW *Synchronization Handler Module* (SHM).

This device acts as an extra slave device of the logarithmic interconnect and has a number of hardware registers equal to the number of cores, where each register is mapped in a specific address range. When a write operation is issued to a given register, a synchronization signal is raised to the corresponding core suspending its execution after one cycle, when the synchronization signal is lowered the execution is resumed. The SHM is programmable in different ways from the software level via APIs. Writing to the OP_MODE register, different synchronization mechanisms can be enabled: if operating in SYNC_MODE, synchronization signals are lowered when all cores have executed the `sync()` API (writing to their respective register, increasing an HW counter inside the SHM), obtaining a cycle-accurate execution alignment. When operating in TWO_PH_MODE, a simple state machine inside the SHM distinguishes cores behaviour between master and slaves enabling a two-phases barrier. When the master reaches a `master_wait_barrier()` primitive, it is suspended until all slaves have reached the `slave_enter_barrier()`. After that, the master is awakened and is the only core executing until the `master_release_barrier()` primitive is reached, reactivating all slaves exactly in the same clock cycle. These APIs and the underlying HW mechanism offered by the SHM are fundamental for the OpenMP library described in the next section.

2.2 Private Instruction Cache Architecture

All the previously described architectural elements are combined together to form the private instruction cache architecture as shown in Figure 3. The cluster is made of 16 ARMv6 cores, each one has its own private instruction cache with separate line refill paths while the L1 data

memory is shared among them. An optional DMA engine can be used to carry out L3 to TCDM data transfers. Access to the off-cluster L3 memory is coordinated by the L3 BUS, requests are served in a round-robin fashion. On the data side all cores are able to perform access to TCDM, L3 memory and eventually to HW semaphores or SHM. The logarithmic interconnect is responsible of data routing based on address ranges as already described in the previous section. Default configuration for the private instruction cache architecture and relevant timings are reported in Table 1.

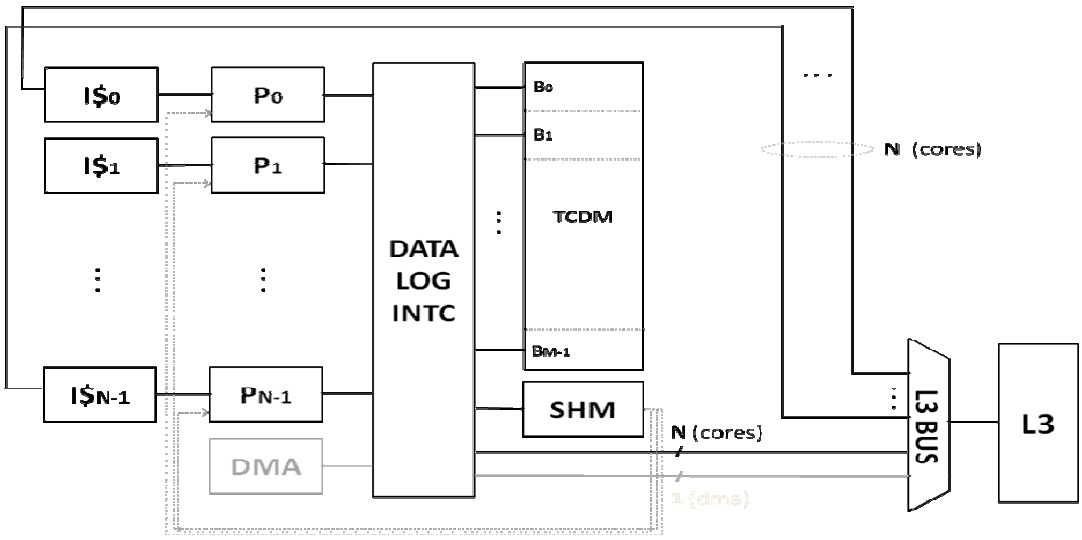

Figure 3: Cluster with private L1 instruction caches

Table 1: Default private cache architecture parameters and timings

PARAMETER	VALUE
ARM v6 cores	16
I$ size	1 KB
I$ line	4 words
t_{hit}	= 1 cycle
t_{miss}	≥ 59 cycles
TCDM banks	16
TCDM size	256 KB
L3 latency	50 cycles
L3 size	256 MB

2.3 Shared Instruction Cache Architecture

Shared instruction cache architecture is shown in Figure 4. From the data side there is no difference between the private architecture except for the reduced contention for data requests to L3 memory (line refill path is unique in this architecture). Shared cache inner structure is represented in Figure 5. A slightly modified version of the logarithmic interconnect described in the previous section (the first stage of address decoding is disabled) connects processors to the shared memory banks operating line interleaving (1 line consists of 4 words). A round robin scheduling guarantees fair access to the banks. In case of two or more processors requesting the same instruction, they are served in broadcast not affecting hit latency. In case of concurrent instruction miss from two or more banks, a simple MISS BUS handles line refills in round robin towards the L3 BUS.

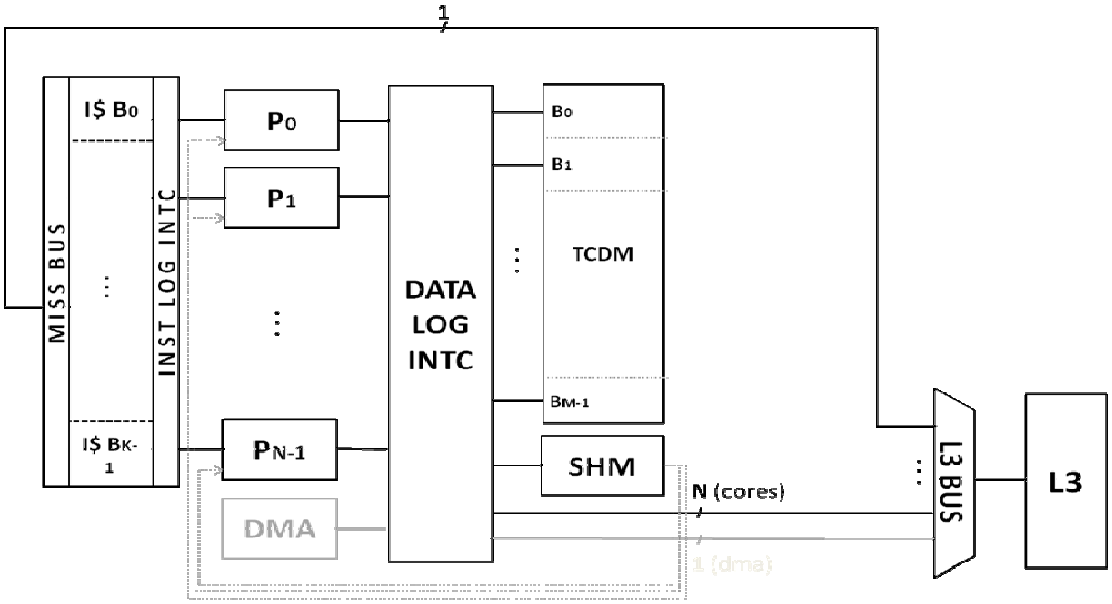

Figure 4: Cluster with shared L1 instruction cache

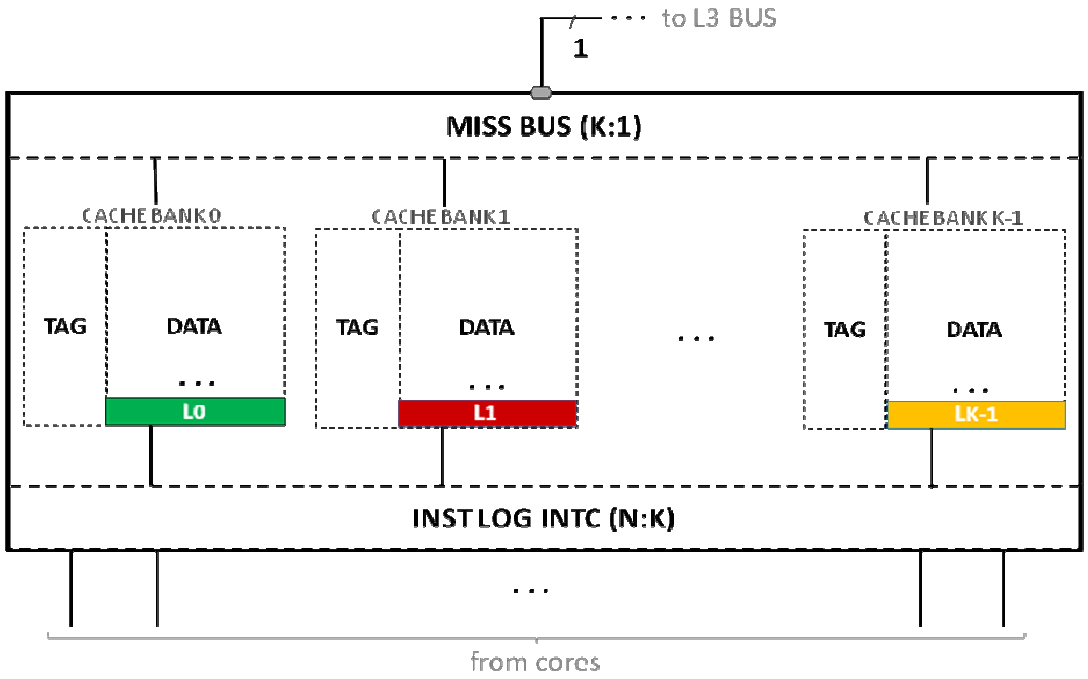

Figure 5: Shared instruction cache architecture

Table 2: Shared cache architecture parameters and timings

PARAMETER	VALUE
I$ size	16 KB
I$ line	4 words
t_{hit}	≥ 1 cycle
t_{miss}	*variable*

2.4 Software Infrastructure

In this section we briefly describe the software infrastructure: first compiler and linking strategies used to compile and allocate all the data needed for the execution of all benchmarks. In the second part we will introduce our custom implementation of the OpenMP library, developed to run on the proposed target architectures.

2.4.1 Compiler and Linker

Before describing compiling and linking strategies applied for our benchmarks, it is of primary importance to introduce the memory map seen by all processors in the architecture. Figure 6 shows the global memory map of one cluster, in which it is possible to distinguish

two memory regions: the L3 memory region (256 MB) and the TCDM memory region (256KB). The first is the off-chip memory used to store the executable of the applications, and data too big to be stored in the on-chip data memory. The TCDM region, mapping the shared data cache, is in turn divided in three sub-regions: *LOCAL_SHARED*, *STACK* and *HEAP*.

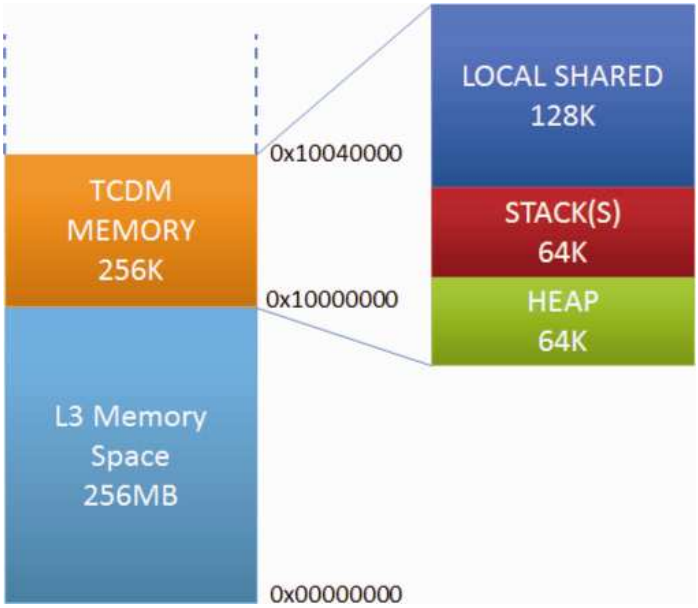

Figure 6: Cluster global memory map

The *LOCAL_SHARED* region is intended to maintain variables of static size (known at compile time) explicitly defined to be stored in the TCDM memory. To force the allocation of a variable in the on-chip data memory we combined the use of a linker script and gcc attributes. We defined a new section in the ARM binary, namely *.local_shared*, used to contain variables to be stored in this region of the memory map. The *STACK* region is defined to maintain the stack of all 16 processors, with 4K assigned to each of them. Each processor calculates its own stack top at simulation startup using a combination of linker script and an assembly boot routine. Finally, the *HEAP* region is used for dynamically allocated structures. The allocation is allowed through the `shmalloc` function, provided by the MPARM applications support (*appsupport*).

2.4.2 Custom OpenMP Library

To parallelize our benchmarks we used a custom implementation of the OpenMP APIs for parallel programming, adapted to run on our MPARM based architecture. The OpenMP programming paradigm is based on two different parallel constructs: *parallel for* and *parallel sections*. The first allow the exploitation of SIMD or SPMD parallelism, the iterations of the for cycle are divided in chunks and assigned to the available cores. The second describes task parallel sections of a program, each core can execute a different portion of code so a different task. To tailor these two constructs to the target architecture it is necessary to consider that our software infrastructure has no operating system. In our implementation all cores execute the same binary file as a single process running on each processor, and the work performed is

differentiated according to the processor id. The Master-Slave mechanism on which OpenMP is based is realized using the two-phase barriers described in Section 2.1.5. We also had to modify the compiler (GCC 4.3) to transform OpenMP annotations in a correct binary form for the MPARM architecture. The compiler has to create all the structures needed to run a certain application and to differentiate the work to be performed by all processors, using *appsupport*'s functions. Our OpenMP runtime has a thin software layer based on a set of shared structures used by the processors to synchronize, share data and control the different parallel regions of the applications. All these structures are stored in on-chip TCDM memory using both statically (*LOCAL_SHARED*) and dynamically allocated structures (`shmalloc`), some are also protected by a lock which is implemented via the hardware semaphores described in Section 2.1.5.

2.5 Architecture characterization

As already outlined in previous sections, we considered a cluster made of 16 ARMv6 cores connected through a low latency logarithmic interconnect to a multiported, multibanked 256 KB TCDM memory. On the instruction side, private and shared architectures differ in the cache architecture. An off-cluster (L3) 256 MB memory is accessible through the data logarithmic interconnect or through the line refill path. Our investigations focus on varying the total instruction cache size, and hereafter the L3 memory latency.

2.5.1 Microbenchmarks

In this section we present the results of three synthetic benchmarks intended to characterize both architectures and to highlight interesting behaviors. The synthetic benchmarks were written using Assembler language in order to have complete control of the software running on top of the architectural templates. They consist of a set of iterated ALU or MEMORY instructions performed to highlight a specific behavior. All the synthetic benchmarks share the common structure shown in Listing 1 below.

```
       mov r6, N_LOOP
       mov r5, #0
_loop: cmp r5, r6
       blt _body
       b _end
_body: ...
       add r5, r5, #1
       b _loop
_end: ...
```

Listing 1 : Synthetic benchmark structure

The performance metrics considered here are the *cluster IPC* (IPC_c, $0 < IPC_c \leq 16$) and its average value, calculated as the number of instructions executed by all the processors divided by the number of cluster execution cycles.

Cold misses: The body of this benchmark consists of only ALU operations (i.e. `mov r0, r0`) leading to a theoretical $IPC_c = 16$ (and average IPC = 1) for both architectures. The plot in Figure 7 shows on the Y-axis the cluster average IPC while X-axis reports how many times the loop is executed. Increasing *N_LOOP* both architectures tend to the theoretical value, but the private architecture starts from a lower IPC due to the heavy impact of cold misses serialization (16 cores contending for L3 access).

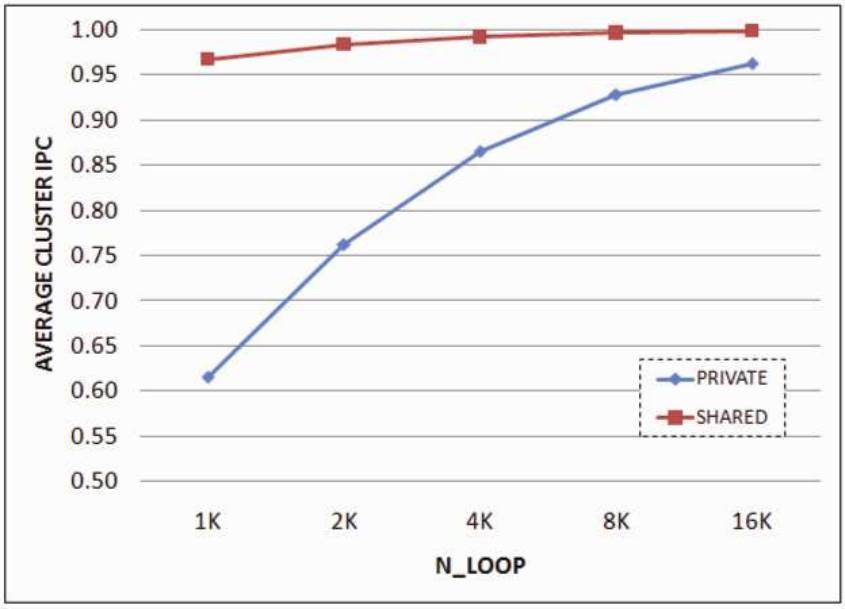

Figure 7: Private vs Shared architectures IPC with only ALU operations

Conlict free TCDM accesses: This benchmark adds the effect of TCDM access. As already mentioned before, in case of conflict free access, TCDM latency is two cycles leading to a single cycle stall between two consecutive instruction fetches. The loop is iterated a fixed number of times (4K in order to lower cold misses effect) and has a variable number of memory operations inside its body. We are considering a banking factor of 1, allowing every core to access a different bank without conflicts. The plot in Figure 8 shows on the Y-axis the average cluster IPC while on X-axis varies the percentage of memory instructions over the number of instructions the loop is made of. Both architectures are affected in the same way, with IPC tending to the asymptotic value value of 1/2[1] (and cluster IPC respectively to 8

[1] A program consisting of only ALU (1 cycle) or MEMORY (2 cycles for TCDM access) operations gives a per-core $IPC = (N_{alu} + N_{mem})/(1 \cdot N_{alu} + 2 \cdot N_{mem})$. Increasing N_{mem}/N_{alu} ratio, leads to an asymptotic value value of 1/2 . Cluster IPC, in this case of perfectly aligned execution, is $IPC_c = 16 \cdot IPC_i$ and its average is equal to the IPC of a single core.

because of any conflict leading to misalignement). Private architectures as an initial lower IPC due to the cold misses effect discussed in the previous paragraph.

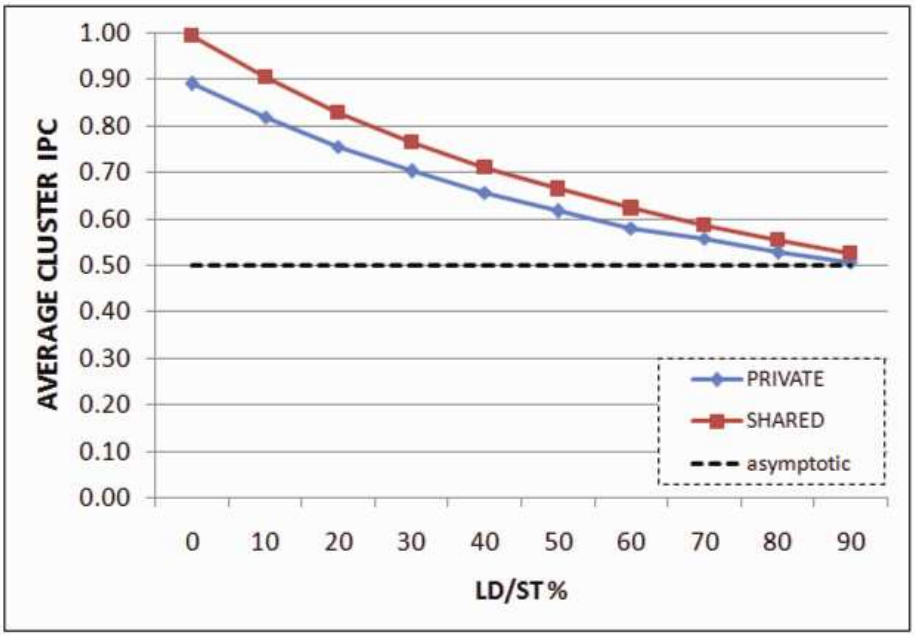

Figure 8: Private vs Shared architectures IPC with conflict free TCDM accesses

Conflicts on TCDM accesses: This benchmark adds another aspect of TCDM accesses: conflicts. Conflicting accesses to the same bank increase TCDM latency thus affecting IPC. In this scenario we considered a realistic ratio between memory and ALU operations of 20%. As before, the loop is iterated 4K times to reduce cold misses effect. The plot in Figure 9 shows on the Y-axis the cluster IPC while on the X-axis varies the percentage of memory accesses creating conflicts on the same bank. It is interesting to notice that, while there are no conflicts on TCDM, the shared architecture performs better the private one because of its intrinsic lower miss cost, in presence of TCDM conflicts the execution misalignment penalizes the shared cache architecture increasing the average hit time. It is important to underline that just a single conflict creates execution misalignment. To explain the sharp reduction of the IPC for the shared cache due to TCDM conflicts, let us consider a simple program consisting of 16 instructions. Before any TCDM conflict occurs, the execution is perfectly aligned leading to synchronous instruction fetching. The conflicting access in TCDM leads to a single-cycle misalignement among all cores in the next instruction fetch.

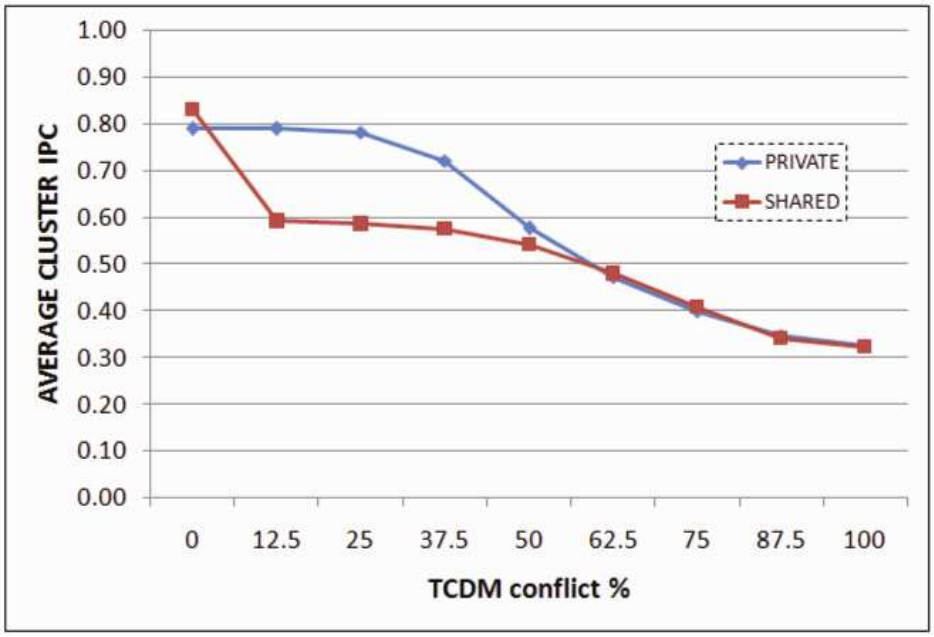

Figure 9: Private vs Shared architectures IPC with conflictingTCDM accesses

As shown in Figure 10, assuming a cache line is made of 4 32-bit words, there will be 4 groups of 4 processors accessing the same line (i.e. bank) but requesting instructions at different addresses. When this situation arises, the average hit time increases from 1 cycle (concurrent access) to 4 cycles (conflicting requests are served in a round-robin fashion). This particular case clearly shows how this architecture is sensitive to execution misalignment.

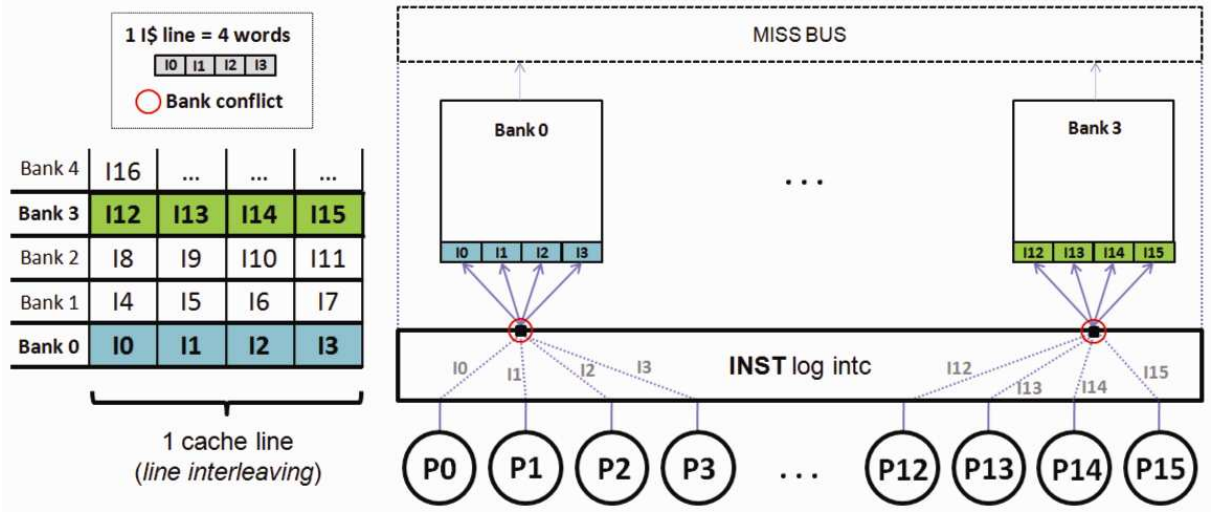

Figure 10: Misalignment in instruction fetching for shared cache

This phenomenon can stand out in an even worse case when the 4 blocks of instructions that are fetched by processors reside in the same bank (situation depicted in Figure 11). This leads to the worst-case for instruction fetch, increasing average hit time from 1 to 16 cycles.

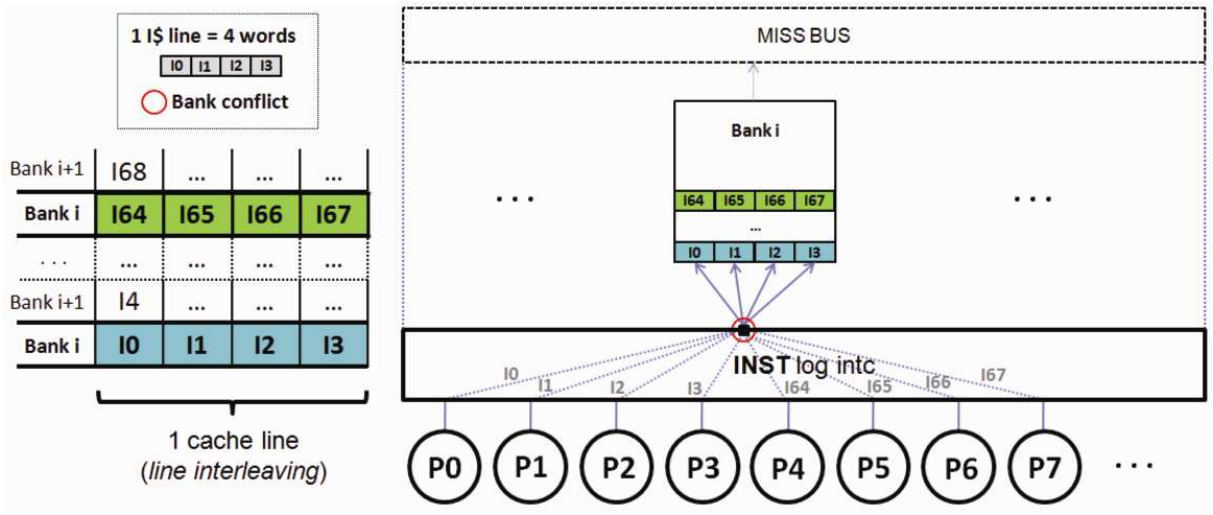

Figure 11: Worst case for instruction fetching in shared cache

2.5.2 Real Benchmarks

In this section we compare the performance of the private and shared I$ architectures by using two real applications, namely a JPEG decoder and a Scale Invariant Feature Transform (SIFT), a widely adopted algorithm in the domain of image recognition. In particular, our aim is to evaluate the behavior of the two target architectures when considering different types of parallelism at the application level. We thus parallelized our benchmarks with OpenMP, and considering three different examples. The first example expresses data-parallelism at the application level. Thus we focused on the two data-independent computational kernels in JPEG: Dequantization (DQTZ) and Inverse Discrete Cosine Transform (IDCT). With this parallelization scheme all processors execute the same instructions, but over different data sets. In the second example we adopt pipeline parallelism in the same JPEG application, where each of the four stages of JPEG – Huffman DC, Huffman AC, Dequantization, IDCT – is wrapped in an independent task assigned to a processor. To keep all the processors busy we execute four pipelines in parallel.

The third example considers three main kernels from SIFT: up-sampling, Gaussian convolution, and difference of Gaussians, all leveraging data-parallelism. In relation with the JPEG data parallel application, SIFT is composed of more complex computational steps that can stress the cache capacity causing more misses.

In what follows we carry out two main experiments. We evaluate the performance by (i) varying the cache size and (ii) the L3 latency. Figure 12 shows the results for the first experiment. Here we consider a fixed latency of 50 cycles for the L3 memory, and we vary the cache size. Focusing on the plots on the top part of the figure, for each of the three benchmarks and for the two architectures we show execution time, normalized to the slowest

value (i.e. the longest execution time for that benchmark). Looking at the data parallel variant of JPEG it is possible to see that the shared cache architecture performs worse than the private cache architecture. We would expect the SIMD parallelism exploited by this application to be favoured on the shared cache architecture, so this finding is seemingly counterintuitive. The reason for this loss of performance is to be found in an increased average hit cost, due to the banking conflicts in the instruction cache described in the previous section. So if we mathematically model the overall execution time of an application as

$$N_H \cdot C_H + N_M \cdot C_M \qquad (1)$$

where N_H and N_M represent the number of hits and misses, and C_H and C_M represent the average cost for a hit and for a miss, C_H may be higher than 1 for the shared cache. To confirm this assumption we report average cache hit ratio (left Y-axis, solid lines) and cost (right Y-axis, dashed lines) in the plots on the bottom part of Figure 12. It is possible to see that the average cache hit cost for the shared cache architecture is ≈ 2.4 cycles, while the number of misses is negligible (miss rate = 0.003%). As a consequence, the right-hand part of the formula above does not contribute to the overall execution time. To understand the absence of cold cache miss impact we analyzed the disassembled program code. The Dequantization kernel consists of a loop composed by a few tens of instructions, while the IDCT kernel loop contains roughly 200 instructions. Overall these results in a hundred misses, and no capacity misses are later experienced. Due to the SIMD parallelism all cores fetch the same instructions, thus only the first core executing the program incurs cold cache misses.

Instruction fetch from the remaining cores always results in a hit. In the private cache architecture, on the contrary, each core individually experiences 104 misses for cache sizes of 32 and 64 Kbytes, while around 400 for 16 Kbytes. These results in a cluster miss rate (total number of misses over total number of instructions) of 0.05% for the private cache and 0.003% for the shared cache.

For the SIFT application the difference in the number of misses between the shared and the private architecture is major, as we can see in Figure 12. In this case, due to the high average miss cost, the shared architecture provides best results despite the high average cost of an instruction hit (more than 2.25 cycles). Indeed, the average cost of a miss is around 800 cycles for the private cache (any size), while for the shared cache it is around 300 cycles. Again, this is due to the fact that for the private cache multiple refills from different cores are serialized on the L3.

For the JPEG pipelined application, the shared cache has a miss rate of 0.03% against 0.3% (64 Kbyte) of the private cache. Moreover in this case the shared cache has lower average costs for a hit (around 1.5 cycles) in respect with the other applications. The shared approach delivers 60% faster execution time for small cache sizes (16K), which is reduced to ≈ 10% for bigger caches. We must distinguish when an instruction is missed for the first time or not. In the first case we identify it as cold miss, while in the second case as a capacity miss. Tab. III shows the number of the capacity misses on the total number of misses in percentage for the private cache architecture across all the applications. The shared cache architecture can better exploit the total cache size, then it experiences no capacity miss. Figure 13 shows the results for the second experiment, where we keep a fixed cache size of 32KB and change the latency

of the L3 memory. The plots on the upper part show normalized execution time (to the slowest, as before), whereas the plots on the lower part show the average cost of a miss. Overall, it is possible to see that for L3 latency values beyond 100 cycles the shared cache architecture always performs better than the private cache architecture.

Considering Eq. 1 again, CM is the parameter which is mostly affected by the varying L3 latency. In particular, the term $N_M \times C_M$ linearly increases with the L3 latency as we can see on the lower part of Figure 12. In the data parallel applications (first JPEG variant and SIFT), the average cost for a miss in the private cache architecture sharply increases with the L3 latency, whereas the same curve for the shared cache has a much smaller slope. This is due to the fact that private caches generate much more traffic towards the L3 memory (16 line-refill requests against a single refill needed by the shared cache). Then, despite the very low number of misses for the JPEG data parallel application, their contribution accounts for 50% of the overall execution time in Eq.1. Regarding the pipelined JPEG application, different from the other examples the average miss cost is slightly higher for the shared cache. However, the miss rate for the shared cache is around 0.02%, while for the private cache it is around 0.2%, thus the shared architecture achieves slightly faster execution times.

Table 3: Number of capacity misses on total number of misses in percentage for the private cache architecture across all the applications and the cache sizes (Bytes)

	JPEG par	JPEG pipe	SIFT
Size 16K	73%	88%	86%
Size 32K	5%	41%	84%
Size 64K	5%	4%	52%

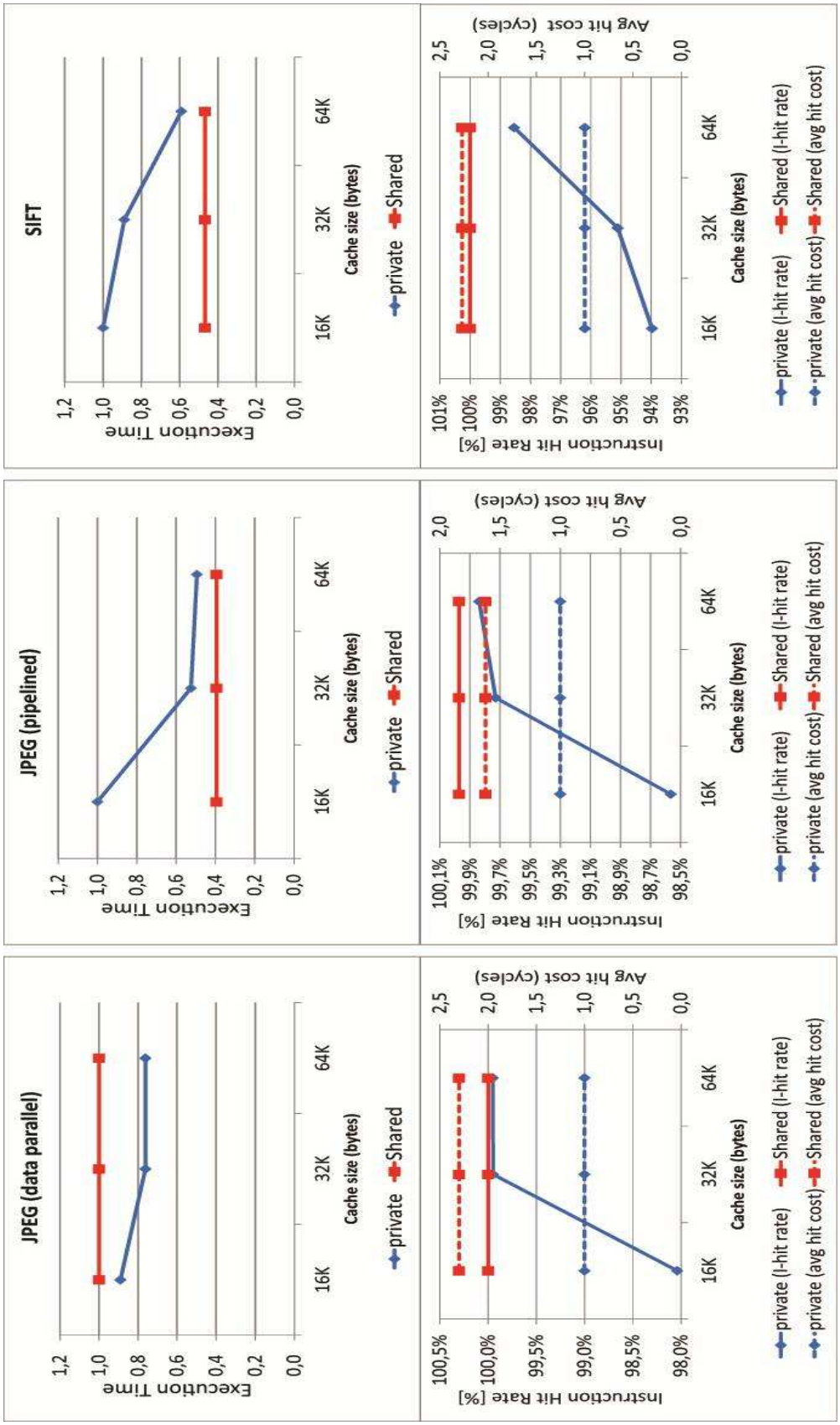

Figure 12: Impact of varying the cache size for different benchmarks

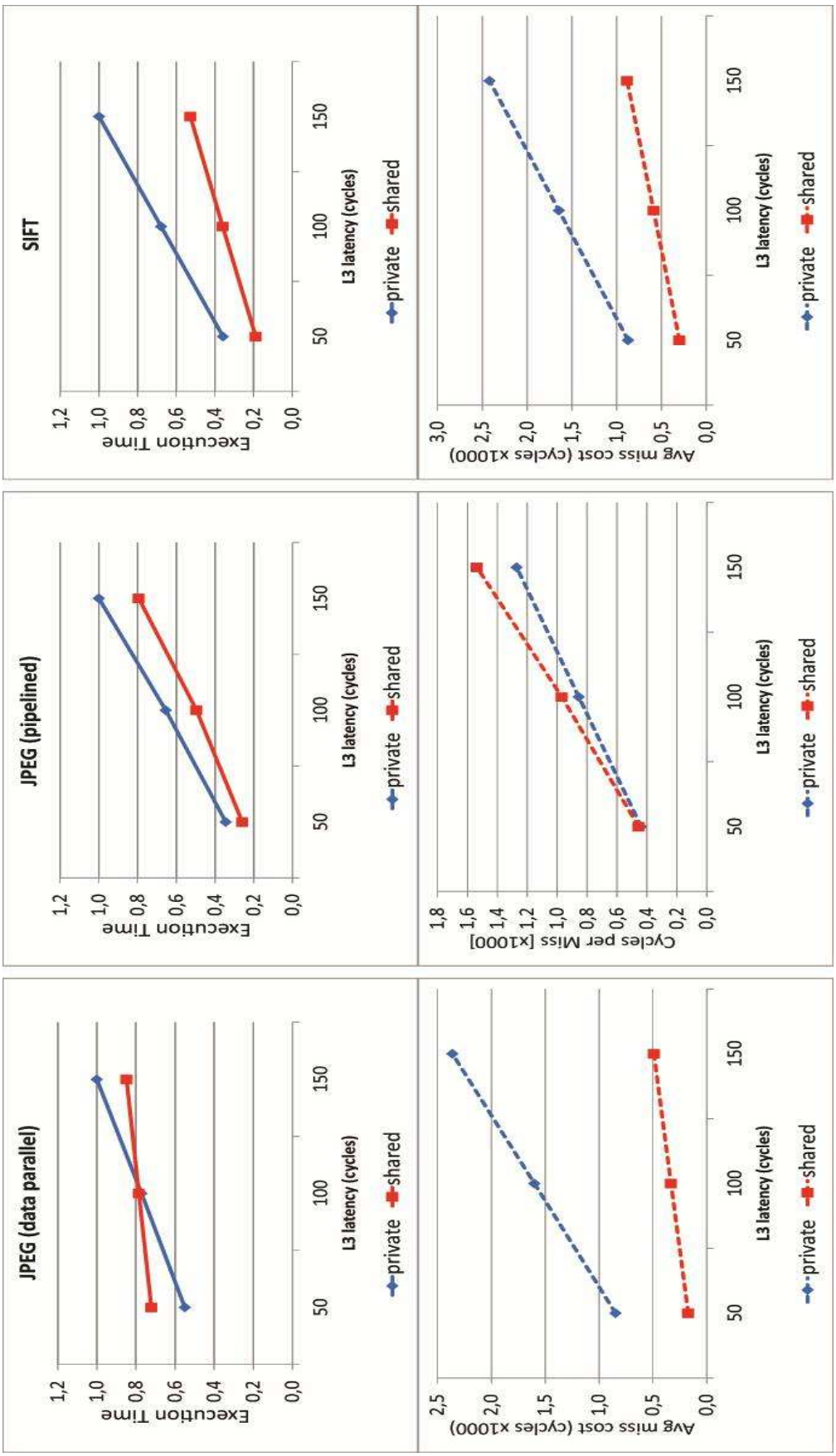

Figure 13: Impact of varying the latency of L3 memory for different benchmarks

2.5.3 Frequency comparison

As a last experiment we want to investigate how faster the private cache design should be clocked to deliver a similar performance to that achieved with the shared cache architecture. We considered as baseline configuration an L3 latency of 150 cycles and an Instruction cache of 32 KB. To carry out this comparison increasing the frequency of the clock within the cluster, we kept constant the L3 latency: our default T_{clk} is 10 ns leading to 1500 ns. The plot in Figure 14 shows on the Y-axis the ratio between shared and private execution time for the benchmarks described in Section V-B, while on the X-axis varies the percentage of frequency speedup.

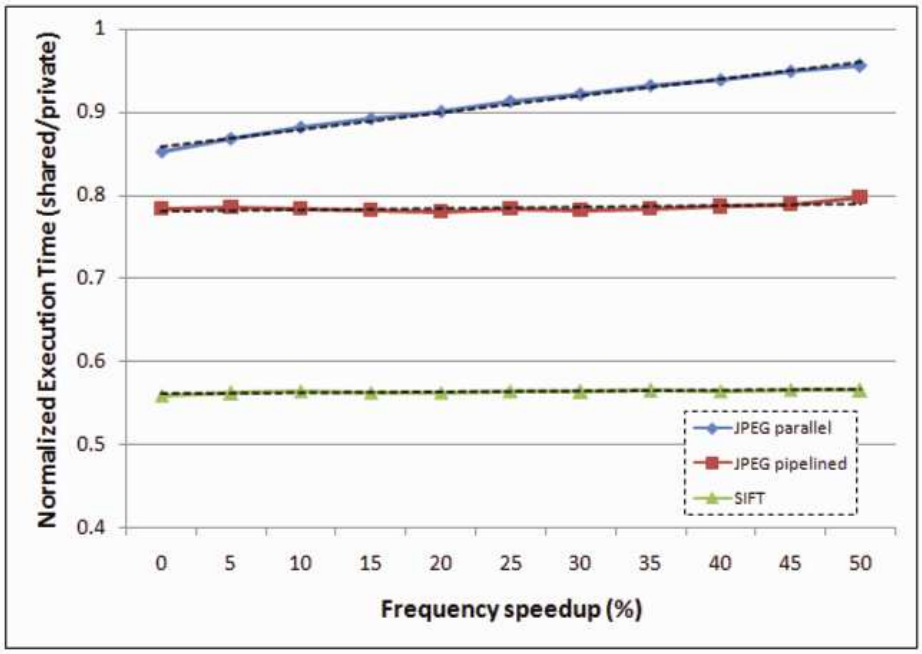

Figure 14: Frequency comparison of private and shared cache architectures

Increasing cluster clock frequency has significant effect only for JPEG parallel while private architecture is quite insensitive for both SIFT and JPEG pipeline benchmarks. To explain such behavior we have to look at the execution time breakdown. A faster clock inside the cluster affects hit time and TCDM latency but has negligible effect on miss latency (dominated by L3 latency) and L3 data accesses. Tab. 5 shows execution time breakdown for all benchmarks.

Tab 1 : Execution time breakdown for JPEG parallel, JPEG pipelined and SIFT benchmarks

	HIT & TCDM	MISS & L3 DATA
JPEG par	51.13%	48.87%
JPEG pipe	14.72%	85.28%
SIFT	0.96%	99.04%

A ≈ 51% of execution time affected by cluster clock frequency gives the performance improvement of the private architecture for JPEG parallel. The same is not true for JPEG pipelined and SIFT benchmarks. This behavior underlines cluster performance is not affected by clock frequency when the program running has the execution time dominated by L3 memory accesses.

3 DRAM Subsystem for 3D integrated SoCs

The performance of standard memory systems, such as the JEDEC DDR or LPDDR (low power DDR) standard, both in terms of access latencies and available bandwidth, has not kept up with the increases in on-chip computation rates, and this gap has generally been referred to as the memory wall [14]. By integrating multiple memory controllers with a large number of IOs for their channels and high-speed signaling the on-chip memory bandwidth can be increased. The number of package pins is however, even for high-end designs with very sophisticated packages, limited by packaging technology, as well as by the need to dedicate power and ground pins [15]. Also the energy per bit consumed for going off-chip is many times higher than the one required for on-chip accesses, as complex and power hungry IO transceivers are needed to deal with the electrical characteristics of interconnections between chips in conventional packages. So achieving peak bandwidths beyond 6.4 GB/s is very difficult using the current LPDDR or LPDDR2 architectures with cost effective packaging solutions such as Package-on-Package (PoP) [16].

To overcome the pin-limited performance growth [15], the power vs. bandwidth dilemma and the memory wall *3D integration* and *3D-stacked DRAM* have been proposed as a very promising solution. 3D-stacked DRAMs reduce the distance between CPU and external DRAM from centimeters to micrometers and improve the bandwidth and access latencies - but more importantly, they provide a major boost in energy efficiency in comparison to standard DRAM devices, such as DDR2, LPDDR2 or DDR3 [17]. In the last years 3D integration of ICs, especially of DRAMs, received tremendous attention [17]–[25]. We recognized that with the WIDE IO DRAM (4x 128 IOs) standard a much higher bandwidth (at 200MHz: 12.8 GB/s) is available for mobile SoCs. However if the full bandwidth is not needed during a read or write, a lot of data is wasted and also the energy for the transfer.

We focus on the energy optimization of the 3D-DRAM subsystem for future terminals. We propose a flexible bandwidth and burst length adaption for the 3D-DRAM and the controller. With this we are able to handle a large range of access sizes from fine-grained (32b) to coarse-grained (1Kb).

The main contributions of the work done in this WP are:

1) the co-optimization of controller and 3D-DRAM,
2) the fine-grained access to the 3D-DRAM which leads to power savings and an energy proportional access,
3) the implementation of the 3D-DRAM controller which covers all features needed for such flexible interface.

Recent products [26], demonstrators [18], announcements [17,27] and investigations [21]–[25] have shown the performance, energy and form factor advantages of 3D integration by using wide-IO buses and TSV interconnects. Facchini et al. [21] mainly focus on the optimization of the interface between processor and memory (e.g., DRAM). The internal structure of the memory subsystem is untouched and not optimized. Instead of focusing only

on the interface [21] we put emphasis on the complete 3D-DRAM subsystem. Therefore we re-architected and optimized the channel controller and 3D-DRAM together.

In [25] Loh uses a so-called "true-3D" configuration based on Tezzaron's 3D technology [28] for performance and power evaluations. However Tezzaron's 3D-stacked DRAM approach requires an additional chip layer in logic technology to speed up the interface and also to reduce the access time to the DRAM. Following the 3D-DRAM integration taxonomy proposed by [21], we focus on scenario 3/4 (CMOS IOs). To be conforming to the WIDE IO JEDEC standard, we use LVCMOS (low-voltage CMOS - 1.2V) signaling for the connection to the channel controller. Therefore we removed the complex IOs (SSTL) for area, performance and power calculations. Our analysis and estimations of the re-designed 3D-DRAMs are based on DRAM technology data from Inotera, Qimonda and Winbond. These data sets enable us to accurately predict power, performance and area for different 3D-DRAM architectures, as well as to optimize the internal structure and organization.

Memory controllers manage the architectural and circuit level interface between processors and DRAM. State-of-the-art DRAM controllers [29] are still designed for narrow off-chip interfaces. They are quite complex components and deploy many complex features to maximize the exploitable interface bandwidth. A DRAM controller can be coarsely split in two parts, front-end and back-end, also called channel controller. The front end includes a multi-port arbitration interface and IO queues with reordering capabilities to improve power consumption and access latency.

A similar approach has been presented by [30]. This WIDE IO multi-channel controller follows a straight forward approach, similar to [29], and does not investigate on real benefits of the WIDE IO (3D stacked technology) where the number of DRAM IOs is increased to 128-bit per channel [26].

Many published works describe advanced DRAM frontends, see [31] for a survey. However, we assume a state-of-the-art front-end which delivers memory transactions to the channel controller, and we focus on adaptation of the channel controller to wide 3D-DRAM interfaces.

3.1 Subsystem Architecture

This section gives an overview about the complete subsystem, shows details of the controller and 3D-DRAM, as well as describes the used 3D-DRAM configurations. The physical placement and organization of the 3D-DRAM subsystem with vertical DRAM channels is depicted in Figure 15.

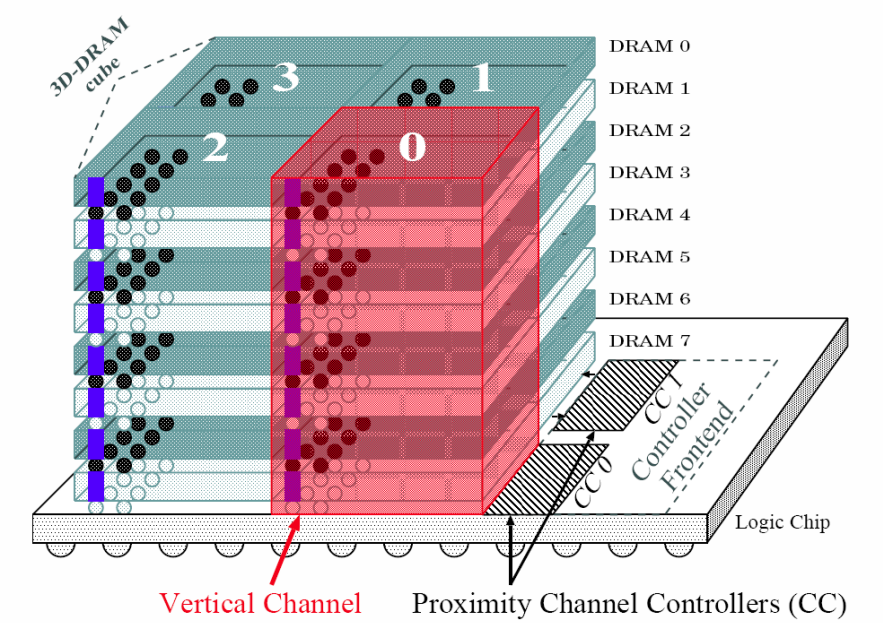

Figure 15: 3D-DRAM subsystem - vertical channel architecture, using up to 8 DRAM layers for a 3D-DRAM cube

The functional representation of the 3D-DRAM subsystem is shown in Figure 16, which consists of a 3D-DRAM memory controller and 3D-DRAM channels.

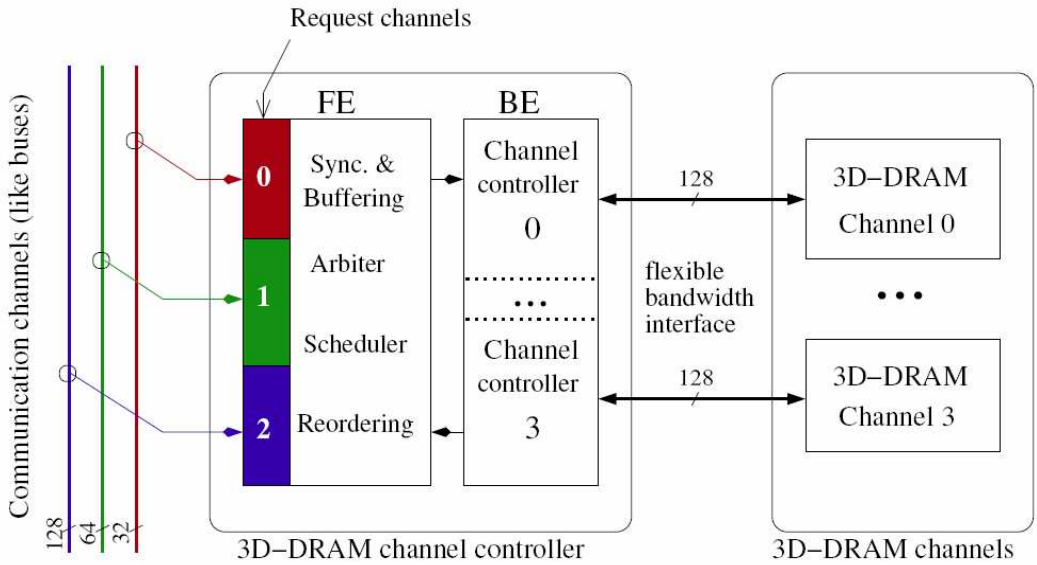

Figure 16: 3D-DRAM subsystem - functional overview (including front-end (FE) and back-end (BE) of the controller, the request channels (RC) and the channel controllers (CC)

The controller is divided into frontend (FE) and back-end (BE). The front-end is responsible for request handling, including FIFO-buffers for each request channel (RC), the arbitration and scheduling. The back-end contains the channel controllers (CC) for each 3D-DRAM

channel and is responsible for the DRAM command encoding and communication. Each CC is attached to the 3D-DRAM channels via a very flexible interface which provides up to 128 data IOs to each 3D-DRAM channel. The aim of this flexible interface is to adjust the bandwidth to/from the 3D-DRAM depending on the incoming request (RC 0 to 2). We have identified three typical request and bus sizes for integration of this subsystem into a heterogeneous SoC:

- 32-bit, AHB/AXI (ARM processor, RC 0)
- 64-bit, Graphics/Imaging, (mobile GPU, RC 1)
- 128-bit, Video Encoder, (HD processing core, RC 2)

3.1.1 3D-DRAM channel controller for SDR/DDR

The channel controller is the back-end part of the memory controller, and is in charge of managing incoming transactions (read and write requests) coming from the front end side (e.g. scheduler + FIFO) and directed to a specific memory channel. This IP has been designed to work seamless with the 3D-DRAM flexible interface and get the maximum benefits in terms of power efficiency and performance. This block is highly configurable to work with several memory types (SDR/DDR/LPDDR) and different flavor (number of banks, IO data width, etc). Additionally some features can be reconfigured at run-time writing in dedicated registers.

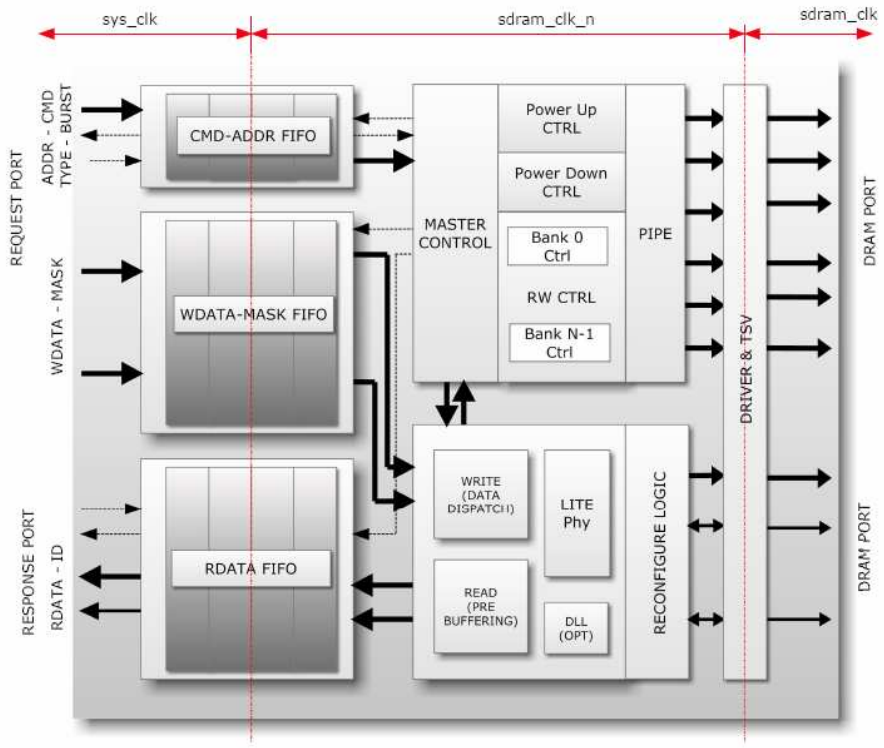

Figure 17: 3D-DRAM memory controller

The internal architecture of the channel controller is depicted in Figure 17 and consists of a first stage of buffering, a DRAM command encoder, a lite physical interface, a reconfigurable interface block and drivers for the DRAM ports.

The buffering stages are needed both to provide resources to temporarily store incoming requests, and to synchronize data transfer when crossing a clock domain boundary (front-end to back-end). The front-end usually is clocked at high speed to allow complex operations like scheduling and reordering, while the back-end is locked to the memory frequency.

The DRAM command encoder is in charge of decoding the incoming requests (post buffering) and generating the right command sequences to pre-charge a bank, to activate a row, perform a read or write, refresh periodically the memory, provide the startup configuration (LMR/EMR) and safe power up. These tasks are managed in several control units and scheduled by a single master control block which is in charge to synchronize these control units. For instance, each bank is managed by a dedicated control unit (bank controller) which tracks all the activity related to a specific bank (active row, type of activation, and timing counters) in order to maximize the performance (avoid bubbles) and minimize the power consumption. The flexible bandwidth adaptation is managed in each bank control unit and depending on the type of access, it provides for the memory the correct signaling needed to perform flexible data accesses (32-bit/64-bit).

Strobe and data strobe are managed in the lite physical interface, which is in charge of latching incoming data from the memory (read data) in case of load, or preparing and dispatching write data in case of store. In case of SDR, the DRAM clock can be used to safely latch read data, or delivery write date. In case of DDR, a DLL is needed to create a 2X faster DRAM clock to safely latch and dispatch data.

The reconfiguration logic block is used in 32/64-bit access modes, and the aim of this module is to selectively mask not driven data lines (during the LOAD), and compact the data in the pre-fetch buffer (normally pre-fetch buffer accommodate 128-bit words). Finally the last stage is composed by a pipe, to filter SDRAM signals from any kind of glitches, and providing more timing budget for inter-die traversal (from logic die to memory stack). The last stage is composed by dedicated IO drivers used to drive the DRAM ports through a stack of TSV and micro-bumps.

3.1.2 3D-DRAM architecture and flexible bandwidth interface

The investigated 3D-DRAMs are closely aligned to the new Wide IO DRAM JEDEC standard. Figure 18 depicts the used two different true 3D-DRAM (Wide IO) architectures. As in [32] explored an optimized 3D-DRAM with 8 banks consists of 8 layers (tiers) which corresponds to a layer per bank organization. For multiple banks per layer the optimal architecture must be revised and details are not shown here because of space reason.

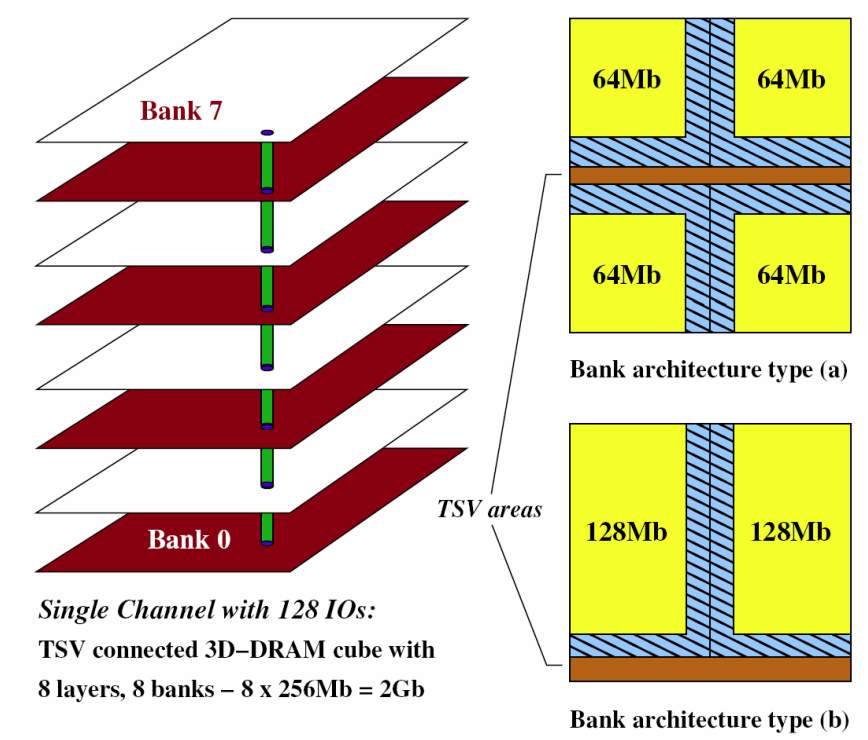

Figure 18: Architectures of a 2Gb 3D-DRAM (single channel)

The two architectures differ in the tile size used for a 3D-DRAM macro block to compose a bank. For bank architecture type (a) the tile size is 64Mb and for type (b) 128Mb. Option (b) has a higher area efficiency and therefore lower cost with the impact of lower performance (max. frequency). Architecture option (a) is very similar to the published 1Gb WIDE IO device of Samsung [26]. Samsung placed the TSV IO area at the bottom-right side for the most left channel which is given by the JEDEC standard. In Figure 19 the yellow areas with tile size numbers show the DRAM cell arrays, the light blue areas with stripes are occupied by column, row and control circuits as well as all other peripheral circuits and the brown areas, as indicated, are reserved for the vertical TSV connections.

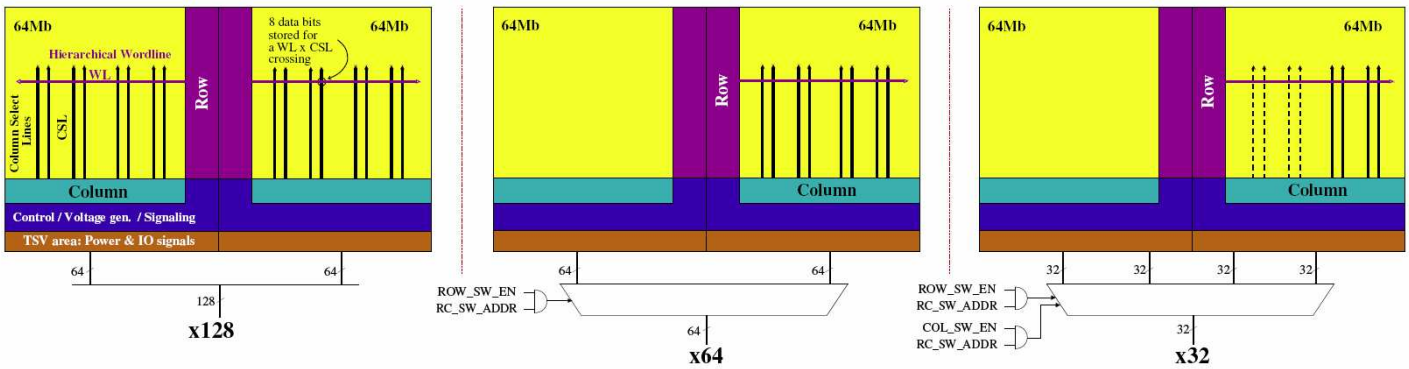

Figure 19: Flexible 3D-DRAM organization/bandwidth switching of a 128Mb bank in a 8 bank 1Gb 3D-DRAM, operating in x128, x64 or x32 mode

Our 3D-DRAM architecture is characterized by an implementation of a flexible bandwidth interface which enables internal organization switching for the 3D-DRAM. This is shown in Figure 20. The 3D-DRAM is able to operate in three different configurations:

- Native mode x128: All column select lines (16 CSLs) and all wordlines (2 WLs) are activated.
- Half bank mode x64: Only 8 CSLs and 1 WL are activated.
- Quarter bank mode x32: Only 4 CSLs and 1 WL are activated.

The multiplexer circuits for switching the data chunks are placed only once on the logic die (channel controller). This reduces the overhead for an 8-layer 3D-DRAM cube and improves performance and access latency.

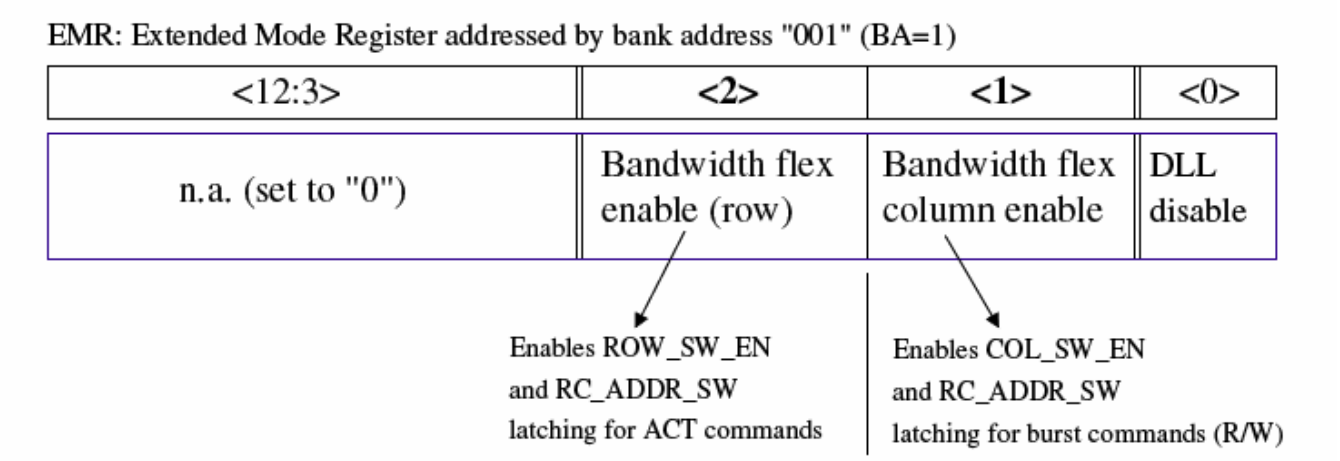

Figure 20: EMR settings for the flexible bandwidth interface of 3D-DRAM

To enable or disable these operating modes new Mode Register (MR) entries have to be defined. We have integrated them into the Extended Mode Register (EMR) as this usually done for new entries, see Figure 6. Additionally to the EMR settings new signals have to be defined for the 3D-DRAM interface in order to switch the bandwidth/organization on-the-fly. The functional details of those are:

- ROW SW EN - enables during a ACT command the organization switch.
- COL SW EN - enables during a RD/WR command the organization switch, depending on the value of ROW SW EN, which is stored for the each bank.
- RC SW ADDR - address which decides on the selection of the data chunk.

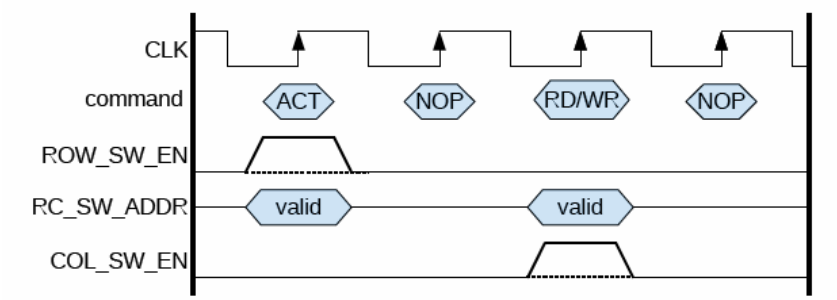

Figure 21: Timing diagram

Figure 21 shows the input timing behavior of the new interface signals related to a ACT and RD/WR command. If the new interface signals are enabled via the EMR and ROW SW EN or COL SW EN are asserted during a ACT or WR/RD command respectively, then a valid RC SW ADDR must be provided.

3.1.3 WIDE IO DRAM Configurations

3D-DRAM SINGLE CHANNEL CONFIGURATIONS

Dens. [Mb]	Arch. type	# of lay.	# of banks	Techn.	Cell size	A_{total} [mm^2]	Freq. [MHz]
			SDR x128				
**256	n.a.	1	4	58nm	$6F^2$	16	200
512	n.a.	2	4	58nm	$6F^2$	26	200
1024	(b)	8	8	46nm	$6F^2$	35	300
*2048	(b)	8	8	46nm	$6F^2$	60	167
4096	(a)	8	8	45nm	$4F^2$	97	200
			DDR x128				
256	n.a.	1	4	58nm	$6F^2$	22	200
512	n.a.	2	4	58nm	$6F^2$	32	200
1024	(a)	8	8	46nm	$6F^2$	44	300
*2048	(a)	8	8	46nm	$6F^2$	69	300
4096	(a)	8	8	45nm	$4F^2$	98	200

** *Density emulates the published Samsung 1Gb WIDE IO chip [13].*
* *This density is used to show the architecture options in figure 4.*

The Table above shows the summary of the generated 3D-DRAM SDR and DDR configurations. Typical performance, area and technology data are given here. In order to be JEDEC conform the supply voltage is set to VDD = 1:2V for all configurations. The footprint for the single channel can be calculated by $A_{footprint} = A_{total} / \#\ of\ layers$. In contrast to [21] we used a TSV diameter value of 8 µm and 16 µm pitch. This diameter and pitch is a good compromise between reported yield and density. The diameter value is very similar to the ones Samsung uses in [26] d=7.5 µm but with a maximum pitch of 50 µm given by JEDEC.

Our TSV capacitance evaluates to 94 fF and TSV resistance to 23m by using copper as filling material. Overall ten different 3D-DRAM configurations were created to strengthen our experiments.

3.1.4 Energy Model Correlation

To ensure the quality of our power modeling we compared a transaction-based power calculator (enhanced Micron version) [33] to our power estimation based on simulation traces and a cycle-accurate DRAM model running these workloads. By tuning our model to the IDD values of the Micron data sheets for a 2Gb LPDDRx32 and LPDDR2x32 part, we achieved less than 1% error compared to the power values evaluated by the transaction-based power calculator. So we verified our power estimation method, see also Figure 22 for detailed correlation results. We used three workloads based on a real-time application and cache transactions for this correlation (low-LL, medium-ML and high-HL).

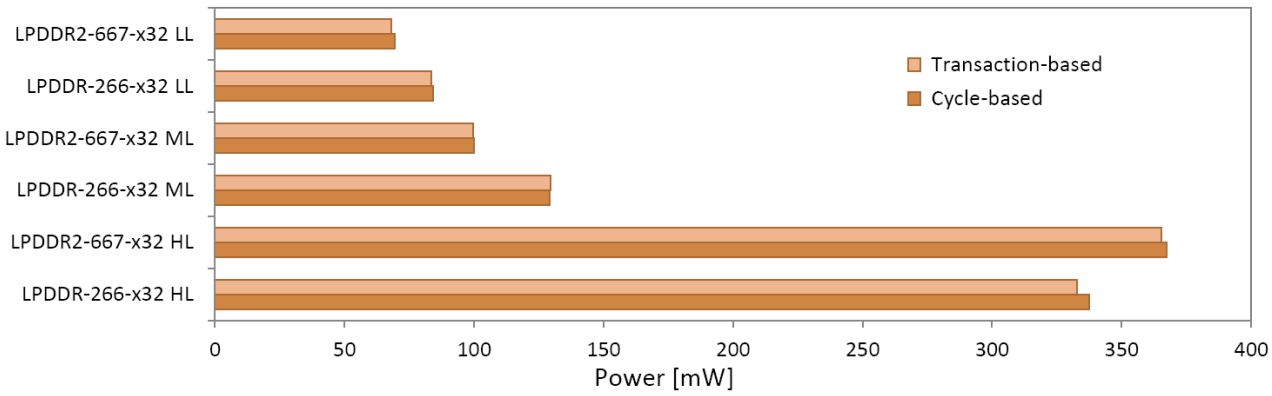

Figure 22: Power model correlation

3.2 Simulation and Traffic Generation

Following simulation setup and infrastructure was implemented to verify the functionality of the 3D-DRAM subsystem and to run our experiments. Three different traffic generators emulate the workloads produced by a typical SoC platform:

- Traffic A: 32-bit - Cache misses of an ARM - 0.1 GB/s
- Traffic B: 64-bit - DMA accesses in a SoC - 0.8 GB/s
- Traffic C: 128-bit - HD Video DMA accesses - 1.5 GB/s

We used 2x the traffic generator A - to simulate a Dual-Core environment. Additionally we enabled 1x traffic generator B and C. B emulates an imaging unit with DMA accesses and C an HD video core. Altogether we created a bandwidth request of 2.5 GB/s to the 3D-DRAMs. For the LPDDRx32-333 and LPDDR2x32-667 based on 2Gb Micron data sheets we reduced the bandwidth to 750 MB/s. These generators can be tuned to simulate different load behaviors and scenarios, see Table below.

AGGREGATED WORKLOADS

Name	Page Hit Ratio [%]	Read [%]	Write [%]	Throughput [GB/s]
IDLE	n.a.	0	0	0
PHR0	0	60	40	2.5
PHR50	50	60	40	2.5
PHR100	100	60	40	2.5

The page hit ratio (PHR) describes the relationship between bursts for which a page (activate a row) must be opened and bursts which are directed to an already opened page. If the PHR is 0% then for each burst an activate command to open a new page is required. If the PHR is 100% then all bursts are issued to an already opened page.

3.3 Experimental Results

In all experiments we present here we used a single channel of the 3D-DRAM cube connected to the flexible bandwidth and burst length adaption interface of the controller. We considered in our analysis only a single channel to put emphasis on the interface and the power savings per channel. Thus the results can be scaled when multiple channels or slices [17] are used. A single channel allows us also a fair and valid comparison to LPDDR/LPDDR2 devices. However, we had to scale down the applied bandwidth to 750 MB/s for LPDDRx32-333 or LPDDR2x32-667 devices as they support only peak bandwidths up to 1.33 or 2.66 GB/s respectively.

3.3.1 DRAM Power characterization

First we characterized the ten 3D-DRAM configurations and LPDDR/LPDDR2 devices by using the workload IDLE. During IDLE only AREF (Refresh) commands are sent to the DRAM. Figure 23 (a) shows the differences in Idle power for DDR and SDR mode. In DDR mode a DLL (Delay-Locked-Loop) is additionally enabled and it contributes significantly to the power consumption. We see also the effect that the leakage current is increased for higher densities.

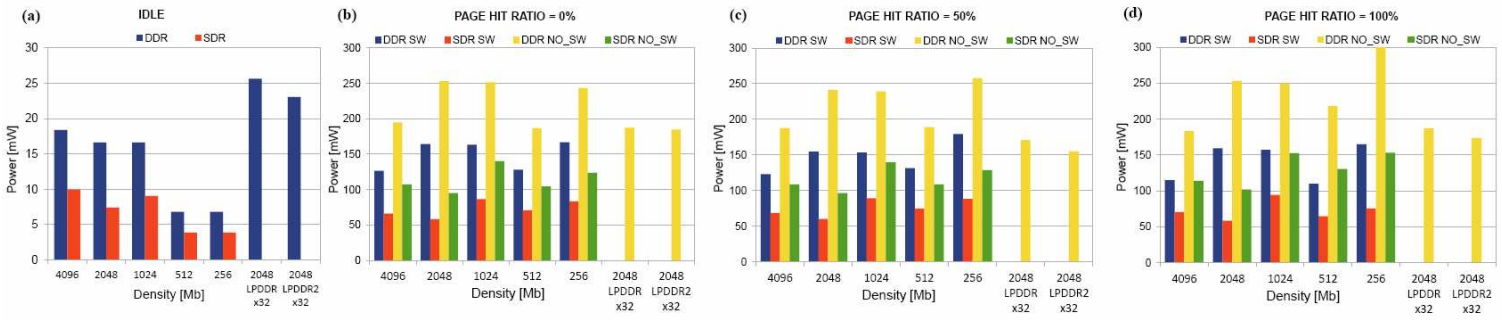

Figure 23: Power characterization of the 3D-DRAMs for DDR/SDR mode and comparison to LPDDR/LPDDR2

For the workload with a page hit ratio (PHR) of 0%, which is equivalent to 100% page miss, each data access to the 3DDRAM opens and closes a new row, this working mode is also called closed-page mode. The results for this mode are maximal 39% power savings between bandwidth switching enabled (SW) and disabled (NO SW) for the 2048Mb density in SDR mode. The average power savings of this mode for all densities and modes (DDR, SDR) are 34.9%, see also Figure 23 (b).

By running a second application-relevant workload with PHR = 50% the results shown in Figure 23 (c) are produced. The power savings for the bandwidth switching enabled mode (SW) are maximal 37.5% again for the 2048Mb density in SDR mode. The average power saving for this workload is 34.1%.

The third workload represents an extreme version of the open page mode, because all bursts are going to an already opened page. So the Read and Write currents are here dominating here. The results are given in Figure 23 (d). The maximal power savings of 50.8% are here in contrast to the workloads before for the 256Mb density. This is the smallest density with only 4 Banks. The saturation for this workload is very fast and because of the lowest power consumption ratio between row activation and burst ($P_{act + pre} = P_{burst}$) the savings become maximal. We achieved for the PHR100 workload an average power saving of 43.2%.

3.3.2 Synthesis Results of the SDR/DDR channel controller

In this section, we discuss the experimental results for the SDR/DDR channel controller in terms of power and area. To get these results, we synthesized the controller with the ST65nm technology library (Low Power process). The frontend flow (Multi V_{TH}) has been performed with Synopsys Design Compiler in topographical mode, while the backend with Cadence SoC Encounter. To run the synthesis flow, we fixed the frequency on both DRAM and front-end side choosing 500MHz and 333MHz respectively.

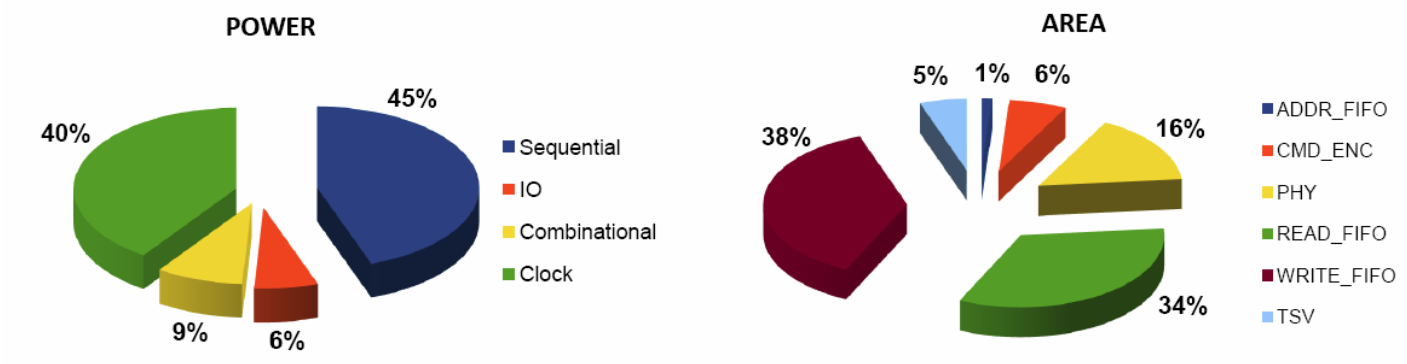

Figure 24: Power and area distribution of the 3D-DRAM channel controller implementation

As can be seen in Figure 24 the power cost (shown on the left) is dominated by clock distribution network (40%) and sequential elements (45% as part of buffering), while the combinational power impact is quite low (9%). The total power consumption (post-place and route) for this test case is 78mW, including the contribution of the IO drivers (6%). The total area cost (shown on the right) is 280K um2 and is dominated by the read and write FIFOs (72%) and Lite PHY (16%). Since these results are dominated by buffering resources (FIFOs), the impact of power and area can be reduced by decreasing the depth of these storage elements.

4 Conclusions

Key to providing I-fetch bandwidth for cluster-based CMP is an effective instruction cache architecture design. We analyzed and compared the two most promising architectures for instruction caching targeting tightly coupled CMP clusters, namely private instruction caches per core and shared instruction cache per cluster. Experimental results showed that private cache performance can be significantly affected by the higher miss cost; on the other hand the shared cache has better performance, with speedup up to ≈ 60%. However, it is very sensitive to execution misalignment, which can lead to cache access conflicts and high hit cost.

We presented also a new architecture for a highly energy efficient 3D-DRAM subsystem for 3D integrated SoCs. We investigated different DRAM families. 3D-DRAM densities from 256Mb to 4096Mb and also 2Gb LPDDR/LPDDR2 devices were characterized by four application-relevant workloads. We designed a 3D-DRAM channel controller which fits perfectly to the flexible bandwidth and burst length adaption interface of the 3D-DRAMs. By using this flexible interface the total 3D-DRAM subsystem consumes in the fastest configuration for 3D-DRAM (300MHz, 2Gb DDR) and controller (500MHz) less than 240mW, for a workload with PHR = 50%. With very low power settings of 3D-DRAM (167MHz, 2G SDR) and controller the subsystem power is decreased to less than 140mW. The experimental results show an overall average of 37% power savings by enabling the bandwidth and burst length flexibility for 3D-DRAMs.

5 References

[1] NVIDIA - Next Generation CUDA Compute Architecture: Fermi -WhitePaper

[2] H. Wong et al. - Demystifying gpu microarchitecture through microbenchmarking - In ISPASS, 2010.

[3] Plurality Ltd. - The HyperCore Processor www.plurality.com/hypercore.html.

[4] Plurality Ltd. - HyperCore Software Developer's Handbook www.plurality.co.il/software/documents/SDH-draft-1.5.pdf

[5] ST Microelectronics and CEA - Platform 2012: A Many-core programmable accelerator for Ultra-Efficient Embedded Computing in Nanometer Technology, 2010.

[6] M. M. S. Aly et al. - Performance and Energy Trade-offs Analysis of L2 on-Chip Cache Architectures for Embedded MPSoCs - In: GLSVLSI, 2010.

[7] The MPARM virtual platform - http://www-micrel.deis.unibo.it/sitonew/research/mparm.html

[8] SystemC Language - http://www.systemc.org/

[9] The SimSoc project - https://gforge.inria.fr/projects/simsoc/

[10] ARM 11 product page - http://www.arm.com/products/processors/classic/arm11

[11] SkyEye full system simulator - http://sourceforge.net/projects/skyeye/

[12] The SoClib platform - http://www.soclib.fr/

[13] A. Rahimi et al. - A fully-synthesizable single-cycle interconnection network for shared-L1 processor clusters - In: DATE, 2011.

[14] [1] W. A. Wulf and S. A. McKee, "Hitting the Memory Wall: Implications of the Obvious," Computer Architecture News, vol. 23(1), pp. 20–24, March 1995.

[15] [2] P. Stanley-Marbell, V. C. Cabezas, and R. P. Luijten, "Pinned to the walls — Impact of packaging and application properties on the memory and power walls," in Proc. Low Power Electronics and Design (ISLPED) 2011 Int. Symp, 2011, pp. 51–56.

[16] D. Oh et al., "Design and characterization of a 12.8GB/s low power differential memory system for mobile applications," in Proc. IEEE 18th Conf. Electrical Performance of Electronic Packaging and Systems EPEPS '09, 2009, pp. 33–36.

[17] Micron Techn., Inc., "Hybrid Memory Cube: Breakthrough DRAM Performance with a Fundamentally Re-Architected DRAM Subsystem," Hot Chips 23, Palo Alto, California, www.micron.com, Aug. 2011.

[18] Imec Belgium. (2011, July) Imec demonstrates 3D integrated DRAMon-logic for low-power mobile applications. Press release. [Online]. Available: http://www2.imec.be/be en/press/imec-news/

[19] D. Dutoit and A. Jerraya. (2010, July) 3D Integration Opportunities for Memory Interconnect in Mobile Computing Architectures. Future Fab Issue 34. CEA-Leti MINATEC. pp. 38-45. [Online]. Available: http://www.future-fab.com/

[20] R. Anigundi, H. Sun, J.-Q. Lu, K. Rose, and T. Zhang, "Architecture design exploration of three-dimensional (3D) integrated DRAM," in Proc. Quality Electronic Design Quality of Electronic Design ISQED 2009, 2009, pp. 86–90.

[21] M. Facchini et al., "System-level power/performance evaluation of 3D stacked DRAMs for mobile applications," in Proc. DATE '09. Design, Automation & Test in Europe Conf. & Exhibition, 2009, pp. 923–928.

[22] D. H. Woo et al., "An optimized 3D-stacked memory architecture by exploiting excessive, high-density TSV bandwidth," in Proc. IEEE 16th Int High Performance Computer Architecture Symp, 2010, pp. 1–12.

[23] EuroCloud Project. (2010, June) Energy-conscious 3D-Server-on-Chip for Green Cloud Services. Press Release. [Online]. Available: http://www.eurocloudserver.com/

[24] I. Loi and L. Benini, "An efficient distributed memory interface for many-core platform with 3D stacked DRAM," in Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE), 2010, pp. 99–104.

[25] G. H. Loh, "3D-Stacked Memory Architectures for Multi-core Processors," in Proc. 35th Int. Symp. Computer Architecture ISCA '08, 2008, pp. 453–464.

[26] J.-S. Kim et al., "A 1.2V 12.8GB/s 2Gb mobile Wide-I/O DRAM with 4x128 I/Os using TSV-based stacking," in Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International, Feb. 2011, pp. 496–498.

[27] K. Krewell. (2011, Aug) Apple A6 Delayed for TSMC 28nm. Processor Watch by The Linley Group. [Online]. Available: http: //www.linleygroup.com/newsletters/newsletter detail.php?num=4736

[28] Tezzaron Semiconductor Corp. (2010, May) Octopus 8-Port DRAM for Die-Stack Applications. Data sheet. [Online]. Available: http://www.tezzaron.com/memory/datasheets/

[29] Denali Software Inc, "Databahn dram memory controller ip," 2009. [Online]. Available: https://www.denali.com/en/products/databahndram.jsp

[30] Sonics, Inc. (2011, May) Using Multichannel DRAM Subsystems to Create Scalable Architecture for Video SOCs. Presentation. [Online]. Available: http://www.sonicsinc.com/uploads/pdfs/MultiCoreExpo-Sonics 3.09.pdf

[31] B. Jacob, with contributions by S. Srinivasan and D. T. Wang, The Memory System: You Can't Avoid It; You Can't Ignore It; You Can't Fake It. Morgan & Claypool Publishers, Jun. 2009, ISBN 978-1598295870.

[32] C. Weis, N. Wehn, L. Igor, and L. Benini, "Design Space Exploration for 3D-stacked DRAMs," in Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE), 2011, pp. 1–6.

[33] K. Chandrasekar et al., “Improved Power Modeling of DDR SDRAMs,” in Proc. of DSD, 2011, 2011.
[34] Platform 2012 cluster functional specifications v0.1 in P2012 project on https://minalogic.net