Simulation of L2 Cache Separation Impact in CPU Performance

Erion Çano

Department of Computer Engineering,
Polytechnic University of Tirana, Albania
erion.cano8@gmail.com

Abstract. Cache memory performance is very important in the overall performance of modern CPUs. One of the many techniques used to improve it is the split of on-chip cache memory in two separate Instruction and Data caches. The current CPU organizations usually have per core separate L1 caches and unified L2 caches. This paper presents the results of simulating different CPU organizations with unified and separate L2 Instruction and Data caches using Marss-x86, a Cycle-Accurate full system simulator. The results indicate that separating the L2 cache memory provides higher overall CPU IPC. The highest improvement is 3% and is achieved in a quad-core CPU model with shared L3 cache. Analyzing the hardware costs and complications of separating L2 cache might be an interesting future work direction.

Keywords: Cache Organization, CPU Performance, L2 Cache Models, CPU IPC.

1 Introduction

The continuous improvements offered by silicon technology make it possible to place more transistors on a single chip. There are currently many commodity PCs equipped with quad-core CPUs which have considerably large L3 cache memories integrated inside. As the performance gap between CPU and main memory has been getting larger, the importance of the fast on-chip cache memories continues to rise. The first on-chip caches were small, unified and one-level only. The current chips contain large L3 caches with different organizations.

One of the simplest techniques used to improve caching and CPU overall performance is the usage of dedicated and separate cache memories for Instructions and Data. Separate L1 I and L1 D caches were a reality since more than a decade ago (since Intel® Pentium® processor models). The most important reason of using separate L1 caches is the possibility to access them in parallel, thus attaining a higher bandwidth. The drawback of having two caches is the higher miss rate in each of them as their size is lower. Other different pros and cons of the two cache organizations are discussed in Section 2.

In this article I simulate and evaluate the overall CPU performance improvements of different possible separate Instruction and Data L2 cache organizations. To have realistic results I consider only bandwidth (number of cache connections) and miss rate (cache size) and keep the other cache specifications identical. To model and simulate the different cache organization CPUs, Marss-x86, a Cycle-Accurate full system simulator presented in [1] is used. Besides providing basic modules of processing cores, caches and memory, Marss-x86 is also extensible and permits quick integration of other simulation modules. The simulation results show that the IPC increase of splitting L2 cache into two equally sized instruction and data caches, no matter how significant, is always positive and ranges from 0.4 to 3 %.

The rest of the paper is organized as follows: Section 2 discusses pros and cons of having unified and separated Instruction and Data caches. Also it summarizes related work about size impact on cache access time and power consumption. Section 3 shortly presents Marss-x86 simulator, its structure and the features it offers. The simulations' specifications and the different machine models I have used are described in Section 4. Section 5 presents the simulation results while Section 6 concludes and shows some possible future work directions.

2 Unified vs. Split Cache Memory Organizations

Besides the traditional unified caches, the other very common design is having separate caches for instructions and data. There are certainly different pros and cons of splitting a cache memory into two smaller caches. The most important advantage of splitting the cache memory is the increase in bandwidth that results. Modern processors can read data from the instruction cache and the data cache simultaneously in a single cache memory cycle. Having two separate Instruction and Data caches and accessing them simultaneously offers the possibility to potentially double the bandwidth [2]. Also the Instruction cache does not need to manage a processor store. Having it separate from the Data cache makes possible to simplify its design. Replacement policy may also result more effective. One can be direct-mapped and the other can be highly associative.

This architecture, also known as Harvard architecture, has the drawback that it needs two complete sets of address and data lines, one to each of the caches. There are also many hardware complications and costs of having to address and index two separate caches instead of one. The most significant drawback of splitting a cache in two smaller caches is the increase in miss rate that will result in each of them as a consequence of the lower (half in case of symmetric split) capacity that will result. An important advantage of a unified cache is the balancing it offers in terms of instruction/data words. If the ratio of data to instruction words changes during the runtime it is adapted to by the replacement policy. This is also rare as the capacity is higher than in case of separate caches. In case of two separate Instruction and Data caches, having too many Instruction or Data words to cache will result in filling up one of the caches. No adaption is possible [3].

In general accessing large memories of any kind takes more time than accessing smaller memories. The circuit level higher access times of larger caches have been also analyzed at [4]. The authors report a proportional increase in access time as a function of cache size for both direct mapped and set associative caches. The authors attribute the higher delays to the circuit comparisons and tag matching. In [5] the authors show that in general it takes less time to access direct-mapped caches than set associative caches. What is more important is the quasi-linear rise of access time with the increase of cache size. They also conclude that there is a considerable decrease in energy consumption if the cache is partitioned into several banks.

Other articles like [6] and [7] evaluate the performance of different Temporal/Spatial split of cache models. They report performance improvements due to the better exploitation of the locality patterns in the code of different applications. In [8] the authors propose an algorithm to reduce cache interference among different simultaneous processes by dynamically (and of course logically) partitioning the last level cache among those competing processes. They report significant cache performance improvements with minimal hardware overhead for the modifications. Another advantage of having smaller separate caches is the reduction in power consumption. In [9] the author presents a scheme named cooperative partitioning to logically divide the LLC ways between the competing cores of the CMP. This scheme uses a shadow tag to monitor the cache requirements of each application and makes a proportional partition of the cache blocks between the cores running the applications. He reports a reduction of 67% and 25 % reduction in dynamic and static energy consumption for dual-core systems.

The purpose of this work is to assess any overall IPC improvement gained from the split of L2 cache into two equally sized instruction and data caches. One large L2 cache provides low bandwidth because it cannot be accessed in parallel by Instructions L1 and Data L1. However it has low miss rate as it is large and can hold many blocks. Two smaller L2 caches (Instructions L2 and Data L2) provide higher bandwidth as they are accessed in parallel. However the miss rate is higher as they have smaller capacities. Being the dominant factors that influence cache performance, bandwidth and size are the two parameters I consider. The others cache characteristics are kept identical in every set of comparative simulations.

3 Brief Introduction of Marss-x86

Marss-x86 is an open source simulator for Cycle-Accurate simulations of multicore CPU configurations. From the different functionalities and features it provides the following are the most important:

- It makes use of different Cycle-Accurate simulation models for out-of-order and in-order single core and multicore CPUs implementing the x86 ISA.
- It supports switching between the Cycle-Accurate simulation mode and the x86 emulation mode of QEMU, an emerging emulator.
- Being based on QEMU, Marss-x86 can boot and execute unmodified operating systems, applications (i.e. benchmarks) and library binaries.
- It includes models of memory hierarchies for single-core and multicore chips and realizes 200 – 400 kilo instructions per second simulations in Cycle-Accurate simulation mode.

Marss-x86 reuses many components of QEMU like emulated IO devices, user interfaces etc [10]. It also provides a MIMO based interface which allows communications between the VM’s software and the simulated/emulated hardware components.

Using this interface programs running in VM can send control signals to the simulator. Marss-x86 allows users to simulate different hybrid CPU configurations consisting of in-order and out-of-order cores of the same chip. It also implements Cycle-Accurate simulation for superscalar pipelines. Marss-x86 framework is extensible and permits quick integration of other simulation modules like DRAMsim, DiskSim and FlashSim. This provides the possibility for accurate and overall system simulations.

The configuration files are written in YAML, a human friendly data serialization language [11]. A typical YAML configuration (modeling) file contains three types of modules: Cores, caches and memory controllers. For each type of module Marss-x86 provides at least one basic module and gives the possibility to create custom modules based on the desired specifications. The YAML configuration file of a dual-core machine with L3 cache is given below:

```
machine:
  dual_l3_1:
    description: Dual Core CPU with L3 cache - configuration 1
    min_contexts: 2
    max_contexts: 2
    cores:
      - type: ooo
        name_prefix: ooo_
        option:
            threads: 1
    caches:
      - type: l1_mesi_32K
        name_prefix: L1_I_
        insts: $NUMCORES
        option:
            private: true
      - type: l1_mesi_32K
        name_prefix: L1_D_
        insts: $NUMCORES
        option:
            private: true
      - type: l2_mesi_256K
        name_prefix: L2_
        option:
            private: true
            last_private: true
        insts: $NUMCORES
      - type: l3_wb_4M
        name_prefix: L3_
        insts: 1
    memory:
      - type: dram_cont
        name_prefix: MEM_
        insts: 1 # Single DRAM controller
        option:
            latency: 90 # In nano seconds
    interconnects:
      - type: p2p
        connections:
            - core_$: I
              L1_I_$: UPPER
            - core_$: D
              L1_D_$: UPPER
            - L1_I_$: LOWER
              L2_$: UPPER
            - L1_D_$: LOWER
              L2_$: UPPER2
            - L3_0: LOWER
              MEM_0: UPPER
      - type: split_bus
        connections:
            - L2_*: LOWER
              L3_0: UPPER
```

This machine model is described in Section 4.3. After executing a simulation Marss-x86 gives basic results such as IPC. Many other specific simulation results can be obtained by running different statistic collection Python scripts that are provided.

4 Simulation Models

To have a reliable assessment of L2 cache memory organization impact in the overall IPC of the machine I used different CPU models such as single core with L1 and L2 caches, dual-core with L1 and L2 caches, dual-core with L1, L2 and L3 caches, quad-core with L1 and L2 caches and quad-core with L1, L2 and L3 caches. Marss-x86 reads the configuration files of these models which are written in YAML format. It generates and compiles the corresponding C++ code which is than executed. The following subsections present the simulation environment, specifications and details of the CPU models that are used.

4.1 Simulation Environment and Specifications

I used a quad-core 2.79 GHz Intel® Xeon® E5-1603 CPU equipped physical machine running Ubuntu 13.04 with kernel version 3.8.0-35-generic. I installed Marss-x86 which uses QEMU to boot and run a disk image over the simulated machines. The emulated system and application consists of Linux kernel 2.6.31.4 and Radix Sort C-implemented algorithm which is used to exercise the CPU models by sorting millions of randomly generated integer numbers. In every simulation model I used different number (1, 2 or 4) of the default Marss-x86 Out Of Order CPU cores each of which has the following specifications:

Table 1. Emulated CPU Parameters

Property	Value
freq:	2793000000
threads:	1
phys_reg_files:	4
phys_reg_file_int_size:	256
phys_reg_file_fp_size:	256
dispatch_width:	4
issue_width:	4
writeback_width:	4
commit_width:	4

Cache memory specifications depend on cache level and size. For L1 I used write-back caches for the single core model and MESI coherent caches for the multicore models. For L2 I used write-back caches if there is not a L3 cache and MESI coherent L2 if there is a shared L3 cache. L3 caches are shared and write-back in every model.

Table 2. Memory Specifications

Property	**Value**
RAM_size	2147483648 B (2 GB)
number_of_banks	64
latency	260 cycles
latency_ns	90 ns

The common cache specifications in every model are 64 B for line size, 2 read ports and 2 write ports. The rest of cache specifications are given in the following sections. I also used the following memory specifications in every simulation:

4.2 Models of Single-Core CPUs

In this simulation two organizations of a single core CPU with 2 levels of on-chip cache memory are compared. The first model is a very common single-core organization with shared L2 cache (i.e. Intel® Pentium M® 740 family has the same basic structure). The second model is uncommon having separate L2 I and L2 D second level caches. In the first model there are two p2p connections between the cache memories, specifically L1 I $\longleftrightarrow$ L2 and L1 D $\longleftrightarrow$ L2, and a single p2p connection between L2 and the main memory. In the second model the cache memory connections are L1 I $\longleftrightarrow$ L2 I and L1 D $\longleftrightarrow$ L2 D. There are also 2 p2p connections between the 2 L2 caches and the main memory. The goal is to assess the IPC of splitting the L2 cache in two halves, L2 I and L2 D and having the possibility to make parallel accesses between level 1 and level 2 caches. Level 1 cache memories are all identical in both models having 32 KB size, 64 sets, 8-way associativity and 2 cycles latency. Level 2 cache memories differ only in size having 2 MB vs. 1 MB L2 I + 1 MB L2 D, 4096 sets vs. 2048 + 2048 sets, 8-way associativity and 18 cycles latency. Cache memories are all write back.

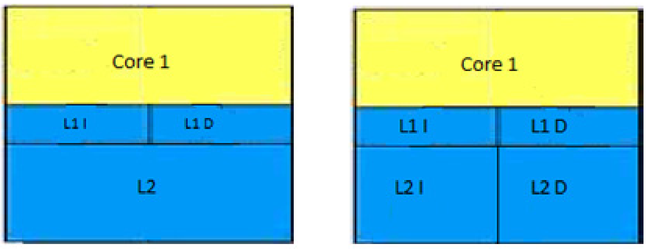

Fig. 1. Single unified L2 vs. Single separate L2

4.3 Models of Dual-Core CPUs

In this simulation I compare two organizations of a dual-core CPU with 2 levels of on-chip cache memory. The first model is a very common dual-core organization with shared L2 cache (i.e. Intel® Core Duo® L2500 family has the same basic structure).

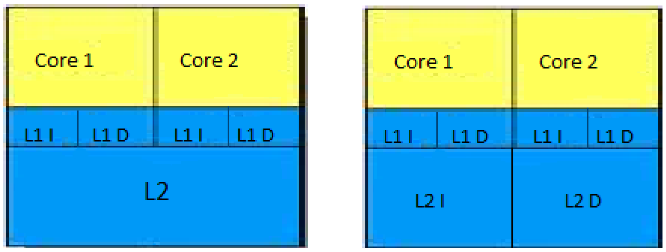

Fig. 2. Dual unified L2 vs. Dual separate L2

The second model is uncommon having shared (not per core) separate L2 I and L2 D second level caches. In the first model there are 4 split bus connections between the four L1 cache memories and L2. There is also and a p2p connection between L2 and the main memory. In the second model there are 4 split bus connections, specifically Core 1 L1 I $\longleftrightarrow$ L2 I, Core 2 L1 $\longleftrightarrow$ L2 I, Core 1 L1 D $\longleftrightarrow$ L2 D and Core 2 L1 D $\longleftrightarrow$ L2 D. There are also 2 p2p connections between the two L2 caches and the main memory. The goal is the same, the evaluation of the IPC improvement of splitting the L2 cache in two halves, L2 I and L2 D. L1 cache memories are MESI coherent and have identical specifications in both models (and in the models that follow). They have 32 KB size, 64 sets, 8-way associativity and 2 cycles latency. Level 2 cache memories differ only in size having 2 MB vs. 1 MB L2 I + 1 MB L2 D, 4096 sets vs. 2048 + 2048 sets, 8-way associativity and 18 cycles latency.

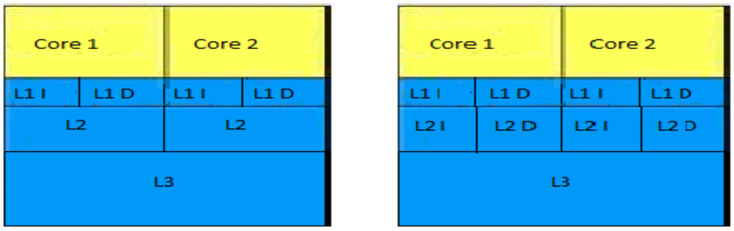

Fig. 3. Dual unified L2 with L3 vs. Dual separate L2 with L3

In this simulation I compare two organizations of a dual-core CPU with 3 levels of on-chip cache memory. The first model is a very common dual-core organization with per core L2 caches and shared L3 cache (i.e. Intel® Xeon® W3505 family). The second model is uncommon with separate per core L2 I and L2 D caches and shared L3. In the first model there are 4 p2p connections between the 4 L1 caches and the two L2 caches (Core 1 L1 I $\longleftrightarrow$ Core 1 L2, Core 1 L1 D $\longleftrightarrow$ Core 1 L2 , Core 2 L1 I $\longleftrightarrow$ Core 2 L2, Core 2 L1 D $\longleftrightarrow$ Core 2 L2). There are also 2 split bus connections, Core 1 L2 $\longleftrightarrow$ L3 and Core 2 L2 $\longleftrightarrow$ L3 and the p2p connection between L3 cache and the main memory. In the second model there are 4 p2p connections between the corresponding L1 and L2 caches. There are also 4 split bus connections between all level 2 caches and the level 3 cache and of course the p2p connection between L3 and the main memory. The YAML configuration file of this machine was presented in section 3. Level 2 cache memories differ only in size having 256 KB L2 vs. 128 KB L2 I + 128 KB L2 D, 512 vs. 256 + 256 sets, 8-way associativity and 10 cycles latency. The shared L3 caches are identical in both models having 4 MB size, 4096 sets, 16-way associativity and 32 cycles latency.

4.4 Models of Quad-Core CPUs

In this simulation two organizations of a quad-core CPU with two levels of on-chip cache memory are compared. The first model is a common quad-core organization with shared L2 cache (i.e. Intel® Core 2 Extreme® family). The second model is uncommon having shared (not per core) separate L2 I and L2 D second level caches. In the first model there are 8 split bus connections between the level 1 cache memories and the shared L2. There is also and a p2p connection between L2 and the main memory. In the second model there are 4 split bus connections between level 1 instruction caches of the different cores and L2 I. There are also 4 split bus connections between level 1 data caches of the different cores and L2 D. There are of course 2 other p2p connections between the two L2 caches and the main memory. Level 2 cache memories differ only in size having 12 MB vs. 6 MB L2 I + 6 MB L2 D and 12288 sets vs. 6144 + 6144 sets, 16-way associativity and 22 cycles latency.

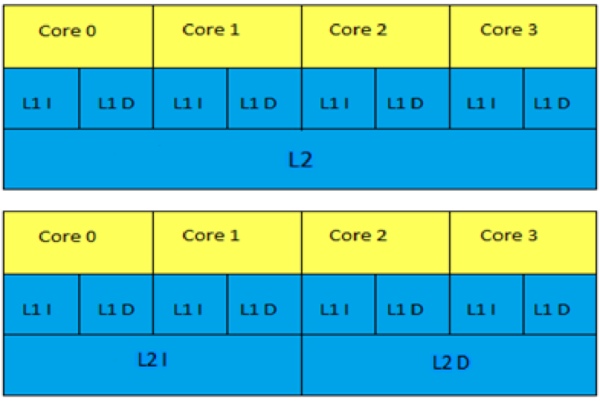

Fig. 4. Quad unified L2 vs. Quad separate L2

In this last simulation I compare 2 organizations of a quad-core CPU with 3 levels of on-chip cache memory. The first model is a quad-core organization with per core L2 caches and shared L3 cache (i.e. Intel® Core® i5 760). The second model has separate per core L2 I and L2 D and shared L3 cache. In the first model there are 8 p2p connections between the L1 caches of the different cores and the 4 L2 caches. There are also 4 split bus connections between the L2 caches and the shared L3. There is also the p2p connection between L3 and the main memory. In the second model there are 8 p2p connections between the corresponding L1 and L2 caches. There are also 8 split bus connections between all L2 caches and the L3 cache and the

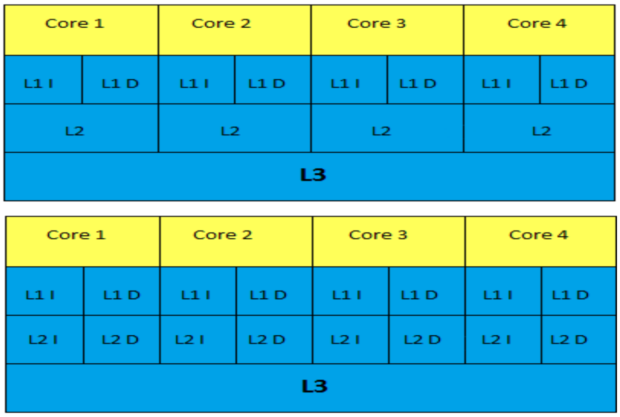

Fig. 5. Quad unified L2 with L3 vs. Quad separate L2 with L3

p2p connection between L3 and the main memory. L2 cache memories differ only in size having 256 KB L2 vs. 128 KB L2 I + 128 KB L2 D, 512 vs. 256 + 256 sets, 8-way associativity and 10 cycles latency. The shared L3 caches are identical in both models having 8 MB size, 8192 sets, 16-way associativity and 34 cycles latency.

5 Evaluation of Results

I ran each simulation (model) 5 times for 30 million cycles each. The average IPCs reported by Marss-x86 were computed and compared. Table 3 presents the IPC difference in % between the average IPCs of the two compared models in every simulation set. First thing to note is the fact that this difference is always positive. There is a slight improvement of 0.4 % from the first Single core model. Dual-Core L2 model also reveals a low improvement of 0.5 %. Splitting L2 cache when it is the last level cache doesn't seem to be beneficial, probably because of the higher miss rates of the smaller L2 I and L2 D. As there is no L3 to serve these requests, they have to go to the main memory which is much slower. The 4th simulation of the quad-core L2 model gives an improvement of 1.2 % which is higher than the first two L2 cache simulations. In this simulations the L2 caches are larger (lower miss rates) and the aggregate bandwidth demand of the 4 cores is higher. Apparently higher bandwidth prevails over higher miss rates.

Table 3. IPC difference between the compared CPU models

Compared Models	**IPC Increase (%)**
Single unified L2 vs. Single separate L2	0.4
Dual unified L2 vs. Dual separate L2	0.5
Dual unified L2 with L3 vs. Dual separate L2 with L3	2.4
Quad unified L2 vs. Quad separate L2	1.2
Quad unified L2 with L3 vs. Quad separate L2 with L3	3

The 3rd and 5th simulations give encouraging results. Higher miss rate delays of splitting L2 cache are minimized by the larger L3 cache. The higher bandwidth of the parallel p2p connections between the corresponding L1 and L2 caches of each core yields a considerable IPC improvement. The improvement difference between 1st, 2nd and 4th simulation against 3rd and 5th sets suggest that splitting the L2 cache doesn't pay off when there is no L3 cache to compensate the higher L2 miss rates. However it gives considerable improvement when the L3 cache is present. The improvement differences between 2nd and 4th simulation sets and also between 3rd and 5th simulation sets suggests the IPC increase is higher in CMPs with higher number of cores as there is higher bandwidth demand from L2 caches. Being aware of the many hardware costs and complications that the separation of L2 cache implies (which need to be analyzed), having separate per core Instruction and Data L2 caches may be a reality in the imminent many-core CPUs with L4 off-chip caches.

6 Conclusions and Future Work

In this paper I presented a set of simulations for different CPU L2 cache organizations comparing the overall IPC of unified vs. separate Instruction and Data L2 caches. The results indicate that splitting L2 cache into two equally sized instruction and data caches provides a not always considerable but higher IPC. This improvement is merely 0.4 % in a single core L2 organization. It is 0.5 and 1.2 in dual-core and quad-core organizations. The highest improvements are attained in dual-core and quad-core CPUs with a shared L3, respectively 2.4 % and 3 %. The results suggest that a L2 cache split in L2 Instruction cache and L2 Data cache makes sense in many core (i.e. at least four cores) CPUs with at least a L3 cache present on-chip.

Even though the results of the last simulation may seem encouraging there are different hardware costs and complications of having per core separate L2 caches. In [12] the author proposes a logical split of L1 data cache based on run-time data locality analysis. He presents an interesting evaluation of the hardware cost this of organization and concludes that the major problem is the extra space required for storing the extra tags of the two caches. A similar analysis of the extra complications and hardware costs of having separate L2 per core Instruction and Data caches is a tough undertaking and a possible future direction.

References

1. Patel, A., Afram, F., Chen, S., Ghose, K.: MARSS: A Full System Simulator for Multicore x86 CPUs. In: Design Automation Conference (2011)
2. Handy, J.: The Cache Memory Book, p. 63
3. Flynn, J.M.: Computer Architecture: Pipelined and Parallel Processor Design, p. 294
4. Wilton, J.E.S., Jouppi, P.N.: CACTI: An Enhanced Cache Access and Cycle Time Model. IEEE Journal of Solid-state Circuits 31(5), 677–688 (1996)

5. Su, C.L., Despain, M.A.: Cache Design Trade-offs for Power and Performance Optimization: A Case Study. In: ISLPED 1995 Proceedings of the 1995 International Symposium on Low Power Design, pp. 63–68 (1995)
6. Prvulovic, M., Marinov, D., Dimitrijevic, Z., Milutinovic, V.: Split Temporal/Spatial Cache: A Survey and Reevaluation of Performance. IEEE TCCA Newsletters (1999)
7. Naz, A., Rezaei, M., Kavi, K., Sweany, P.: Improving Data Cache Performance with Integrated Use of Split Cache, Victim Cache and Stream Buffers. SIGARCH Computer Architecture News 33(3), 41–48 (2005)
8. Suh, E., Rudolph, L., Devadas, S.: Dynamic Partitioning of Shared Cache Memory. Journal of Supercomputing Architecture (2002)
9. Sundararajan, T.K.: Energy Efficient Cache Architectures for Single, Multi and Many Core Processors. PhD Dissertation (2013)
10. Patel, A., Afram, F., Ghose, K.: MARSS-x86: A QEMU-Based Micro-Architectural and Systems Simulator for x86 Multicore Processors (2011)
11. Machine configuration, `http://marss86.org/~marss86/index.php/`
12. Samdani, G.Q.: A Split Data Cache Organization Based on Run-Time Data Locality Estimation. PhD Dissertation (2000)