"The MPCAM Based Multi-core Processor Architecture: A Contention Free Architecture"

ALLAM ABUMWAIS, Department of Computer Engineering, Near East University, LEFKOSA, CYPRUS

E-mail: Allam.Abumwais@aauj.edu

ABDULKARIM AYYAD, Department of Computer Engineering, Al-Quds University, EAST JERUSALEM, PALESTINE
E-mail: akayyad@eng.alquds.edu

Abstract: A symmetric multi-core processor is a chip which integrates a number of processors (cores). Each core has its own local memory which is accessible by its core only. The cores share and equally access a shared memory. The multi-core processors suffer from the delay caused by the contention among the cores to access the shared memory. Also, a bigger delay results from the cache coherence operations, where each core must update other cores on any change it makes on a shared variable. This is accomplished by broadcasting the change to the rest of the cores. In 2014 the authors completed, tested and verified an organization of a multi-port content addressable memory (MPCAM) which, if used as a shared memory, it allows all the cores of the processor to access it simultaneously without the need for queuing and arbitration. The access time is the same as that of accessing the core's private memory. The organization of this memory guarantees the cache coherence automatically and eliminates the need for cache coherence operations. This is an unprecedented result. This architecture represents a whole solution to the long standing problem of latency due to contention and cache coherence operations in multi-core (formerly multiprocessor) system.

Key-words: Multi-core, shared cache, contention, cache coherence, dual port CAM, multi-port content addressable memory (MPCAM).

1 Introduction

In symmetric multi-core processor systems, the processor includes a number of cores and a shared memory. Each core includes a pipelined processor and two levels of private cache, cache level 1 (L1) and cache level 2 (L2). L1 includes instruction cache and data cache whereas cache L2 is a data cache. The shared memory is considered as a third level data cache (L3). Each core can exclusively access its private L1, and L2 caches, whereas it shares cache L3 with other cores of the system. All cores access the shared memory through an interconnection network. The memory management unit (MMU) of the system loads the data caches with the primary data.

Accessing the shared cache causes contention among the cores which means increasing the shared cache access time and hence slowing the core operations. The contention and the resulting extra latency are there even if the best network (NXN crossbar switch) is used between the cores and the modules of the shared cache [1], [2], and [3].

What add to the latency are the cache coherence protocols which are used to guarantee that the core is accessing the right version of data. Also when the core updates a shared variable, it has to broadcast the new version of the variable to all other cores. Only one core can broadcast its updated variable at a time. This results in queuing and long latencies [4], [5].

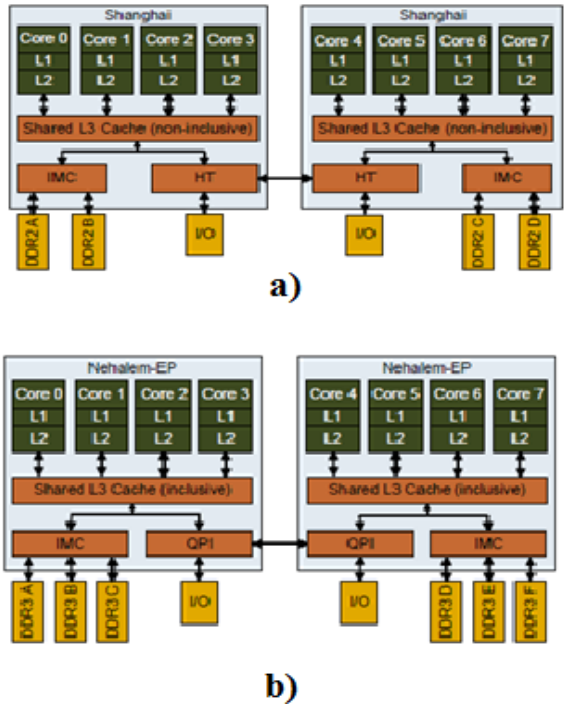

Figure 1: The AMD multi-core architecture (a) and the Intel multi-core architecture (b) [6]

To dilute this problem, modern multi-core processor systems employ a fast serial point to point link among the processors and chipsets of the system. Examples are the hyper transport link (HT) of the AMD Shanghai and the Quick path (QP) of Intel Nehalem depicted in figure 1 [6] [7].

In the above examples, the AMD processor includes 64 KB L1 cache/core, 512KB cache L2 cache/core, and shared 6MB cache L3 whereas Intel processor includes 32 KB L1 cache/core, 256 KB L2 cache/core, and shared 8 MB L3 cache. Cache L2 looks redundant. Rather than writing to L2 we can write to L1 where we can get faster access. In fact, we can compromise the size of L1 and L2 so that we can have a new L1 so that L1 < "newL1"<(L1+L2) in size with "newL1" = L1 in access time. The modules of L3 can be redesigned and reorganized in the crossbar so that we can get an access time equal to that of L1. The new L3 can be considered as a shared L2 cache. The design presented in this paper replaces the L3 shared cache with what is called Multiport Content Addressable Memory (MPCAM). The MPCAM is a very fast fully associative memory. It allows simultaneous access for read and write operations for all cores of the system. No queuing and no arbitration are needed. It also automatically guarantees the cache coherence for all variables without the need for cache coherence protocol.

2 The MPCAM-based Architecture

The multi-core organization presented in this paper includes only two levels of cache. L1 is the private (local) cache and L2 is the shared cache. Figure 2 depicts this architecture.

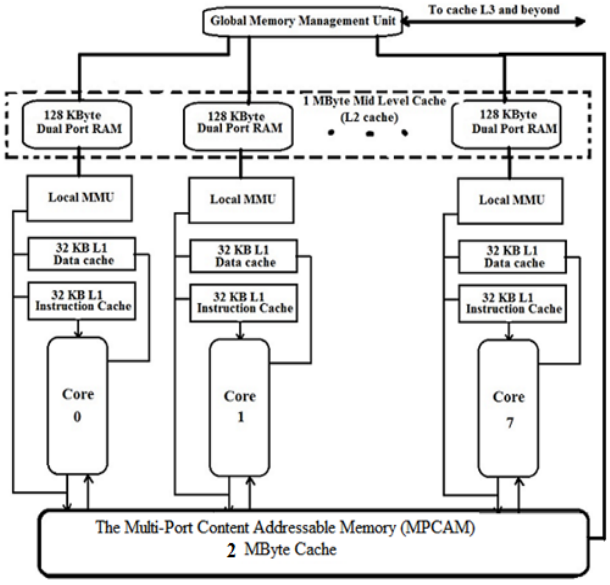

Figure 2: The proposed Multi-core processor architecture

Cache L1 includes 32 KB/core instruction cache and 32 KB/core data cache. Cache L2 can range from 2 MB to 6 MB of data cache without exceeding the allowed silicone limits. It is also equally and simultaneously accessible by all cores of the system. In the following paragraphs, we are going to show the architecture of the MPCAM and explain how it works.

The MPCAM is built of an array of dual port CAMs (DPCAMs) embedded on the cross points of a crossbar network. The dual port CAM (DPCAM) is a content addressable memory with two specialized ports; through the first, we write the data and the tag to the first available memory line, and through the second we apply the tag to all memory lines simultaneously to search for and retrieve the data if found. Simultaneous write and read operation are allowed unless the

same line is addressed. In this case, the priority is given to the writing port [8]. This is different from the dual port CAM presented in [9]. Figure 3 depicts the MPCAM organization.

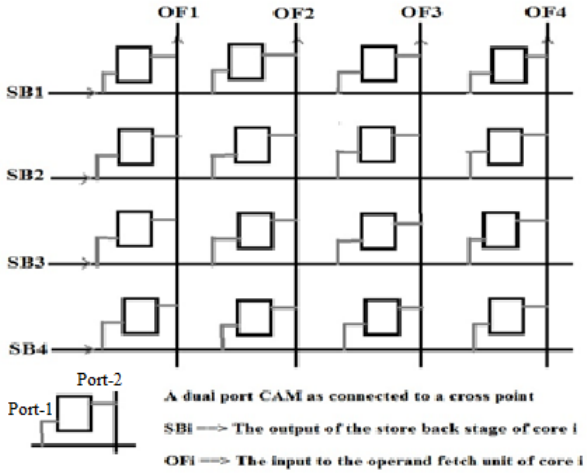

Figure 3: A 4X4 MPCAM organization

In this organization, all port-1s of the embedded DPCAMs are connected to the horizontal buses (input buses) of the crossbar, and all port-2s are connected to the vertical busses (output buses) of the crossbar. With reference to figure 2 and 3, if this MPCAM is implemented as a shared cache, the store back (SB) units of the pipeline are connected to the horizontal busses of the crossbar, and the operand fetch (OF) units are connected to the vertical busses. All the processor cores can write (broadcast) their updated shared variables to all the DPCAM modules in their row simultaneously. The tag accompanying the data includes an address and the version number of the variable. The write enable pointer of each DPCAM points to the least recently written to the line of the memory. If the DPCAM has, say 2 K lines, the current line won't be written to again until 2 K write operations have elapsed.

The operand fetch unit of each core can search for the required data by applying the tag to all lines of all modules in its column. Because the data is broadcast to all modules in the row, a copy of it exists in each column. So, all the cores can search for the same or different data in their columns simultaneously. One can see that there is no contention among the cores in the write or read operations to the shared cache. Also, as all versions of the variable exist in the shared cache, there is no need for executing cache coherence protocols or processor to processor communication to broadcast the latest version of the variable. For the primary shared data, we can add a row of DPCAMs to the crossbar with its input bus is connected to MMU of the system. The MMU loads the primary shared data to that row. Alternatively, the MMU can distribute the primary data to all rows evenly, where the cores can access them through the columns simultaneously.

The versions of the shared variable can be near-reaching (consumed by near coming processes) or far-reaching (consumed long after being written). The far-reaching has the possibility of being overwritten before used. In this case, the compiler can produce a near "read and store" operation so that the concerned cores read them from the shared memory and store them in their local memories. Alternatively, we can implement two DPCAMs at each cross point of the crossbar as shown in figure 4, one for the near-reaching variables and the other for the far-reaching ones. The modules of the far-reaching variables are going to be smaller than those of the near-reaching ones because they are less frequently used.

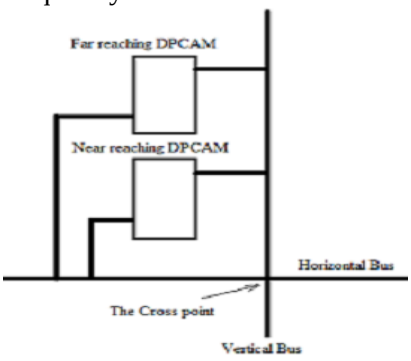

Figure 4: The near-reaching and far-reaching DPCAM implementation

3 Implementation and Verification

Using Quartuss II design and simulation package from Altera [10], the authors have designed, simulated, and verified a 3X3 MPCAM as targeted to Stratix-3 Field programmable gates array (FPGA) from Altera [11], [12].

We could not design a larger MPCAM because of the "number of pin limitation". However, it is

possible to design much larger MPCAM if we design it as integrated with a multi-core system where we don't need any external pins. The access time will be the same because regardless of the number of DPCAM in the row and the column, they will be accessed in parallel simultaneously.

3.1 Functional Verification

Figure 5 shows a sample of 3X3 MPCAM operations. Note that in interval 1 (0 to 10 ns), cores 1, 2 and 3 are writing (broadcasting) data and tags to the modules in their rows. Immediately after write (WR) signal (form 151) goes low, the presented data in forms (1, 34, and67) appear on the output of the flip/flops of the written to locations (forms 152, 185, and 218), which means that the data has been stored in the targeted memory locations. In the second interval (20 to 30 ns), all the cores present the tags of the data written by core-2 (forms 252,269, and 286) with a read (RD) signal (form 251) to read this data. Immediately after the RD signal goes low, the data appears on the column buses of the three cores (forms 303, 336 and 369), i.e., the three cores read the same data simultaneously. Interval 3 shows core 1 and core 2 reading the same data simultaneously, while core 3 is writing (broadcasting data).

3.2 Timing Verification

Figure 6 shows the timing of the read and write operations of the MPCAM as targeted to Stratix-3 FPGA from Altera. Figure 6 shows that in the interval (60-30ns), core 1 presents the data, the tag, and a WR signal (form151) to the modules in it row. Three nanoseconds after the WR signal goes low (at 87 ns point), the data appears on the output of targeted modules (at 84 ns point) the scale reads from right to left.

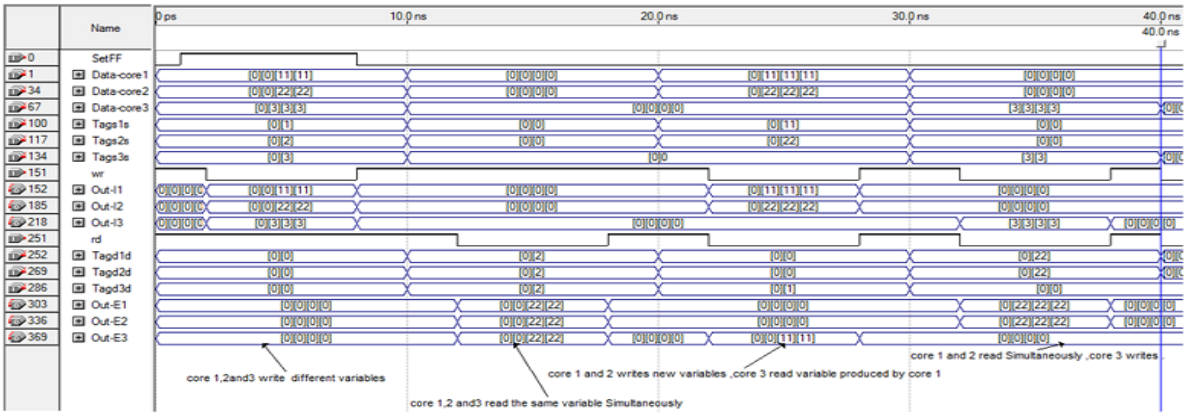

Figure 5:a sample 3X3 MPCAM operation

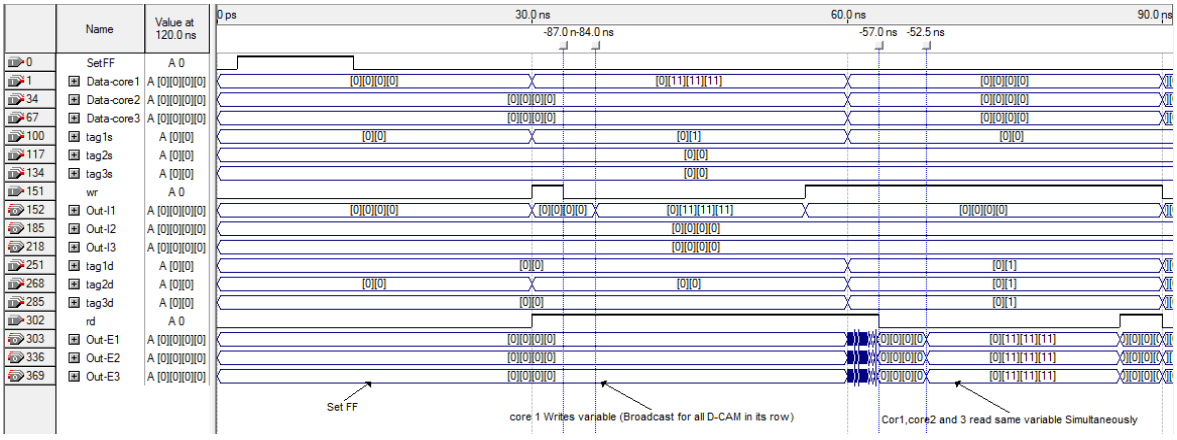

Figure 6: The timing of the read and write operations of the MPCAM as targeted to Stratix-3 FPGA from Altera.

To prove that the data has been written to all modules, we made all the cores access the data in their columns by presenting the same tag (forms 251, 268, and285) and a RD signal (form 302). After 4.5 ns the RD signal goes low, the same data appears on three different columns and read by three different cores, simultaneously.
From the timing diagram, we can see that the write and read operations to the shared cache wouldn't take more than 5 to 6 ns as compared to 35-40 clock cycles (70-80 ns at 500 MHz) average shared cache access time in Nehalem processor [13].

4 Simulating The MPCAM-based Multi-core Architecture

In this section, we are going to simulate the performance of the multi-core architecture which includes the MPCAM as a shared memory. The simulation package Valgrind is used as a simulation tool [14], [15]. We found that the Valgrind was used to simulate a modified version of an AMD architecture which has a private cache L1 and a shared L2 cache. The cores access L2 through a crossbar switch network, where L2 includes a number of modules equal to the number of cores in the system. In our proposed architecture, the same size of private L1 cache and an NXN MPCAM as an L2 shared cash were used. N is the number of cores in the system.

The simulation process was run for a number of cores ranging from 1 to 8 cores for the AMD and our architecture. The Valgrind provides three simulation functions; the "date" and 'df" functions, the performance "PP" function, and the dependency function. The first functions "date" and "df" return the date and the amount of space used on all mounted volumes. The performance "PP" and the dependency programs return the execution time and the number of cache misses during the program execution. The execution time is the elapsed CPU real time between invocation and the termination of the program.

The "date" and the "df" functions are run to prove that the simulation is running correctly. They were run a number of times for a single core for both architectures. They returned the same number of cache misses, a thing which proved that the simulation process for proposed architecture was going correctly.

4.1 The Execution Time

The multithreading performance program "PP" was run for both architectures to measure the execution time of two parallel loops and to count the cache misses for 1, 2, 3, 4, and 8 cores respectively. Table 1 shows the results of the execution time.

Table 1:The execution time of the two architecture

Number of core	AMD multi-core execution time in seconds	The MPCAM based multi-core execution time in seconds
one	16.2	16.2
two	10.1	9.8
three	8.43	6.8
four	7.16	6.1
eight	4.98	3.13

The results shows that the MPCAM based architecture has a better performance than the crossbar based AMD architecture.

4.2 Cache L2 (Shared Variable) Misses

The "PP" program was run to find the shared variable misses in for the AMD and the MPCAM based architectures. The results are shown in table 2 and figure 7. The miss ratios in the shared cache of MPCAM architecture are negligible as compared to those of the AMD shared cache architecture. As there is no contention problem in the MPCAM based architecture, a cache miss in the MPCAM based architecture implies that the consuming core has presented its read request to the shared cache before the variable is written to the shared cache. The onus here is on the scheduling policy to reduce the cache missed through observing relaxed dependency timing between the production and the consumption of the shared variable. We used another dependency multithreaded benchmark program to compare

the execution time between our model and AMD model when the threads of cores are dependent on each other, so we used a benchmark program that multiplies two matrices and calculates the determinant for the result matrix. Figure 8 shows the results of execution time when we run this program in the two simulators.

Table 2: The shared caches misses and miss ratios for the AMD and the MPCAM based architectures.

Num. of cores	AMD architecture		MPCAM based architecture	
	Number of misses	Miss ratio	Number of misses	Miss ratio
2	1,006,004	1.6%	1800	0.00228%
3	1,257,560	2.0%	1665	0.00211%
4	1,506,206	2.1%	2,298	0.00316%
8	1800,000	2.4%	3100	0.00413%

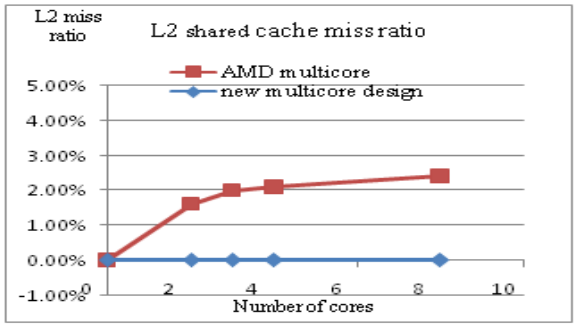

Figure 7: The shared cache miss ratios for the AMD and the MPCAM based architectures

4.3 Running the Dependency Function

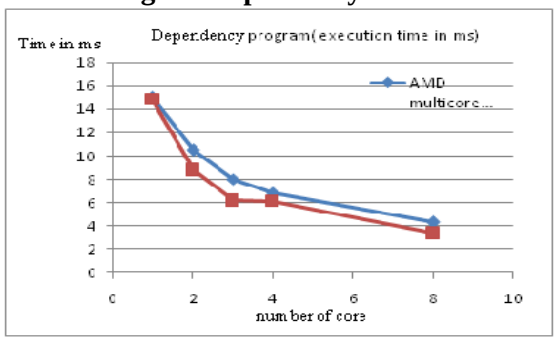

Figure 8: Dependency Program Execution Time in Multi-core

5 Conclusion and Discussion

In this paper, we presented an unprecedented shared memory organization. By providing a numbers of input and output ports equal to number of the processor cores, it totally eliminates the contention among the cores to access the shared memory. There is no network contention and there is no memory interference. As each core of the system writes (broadcasts) the variable to all DPCAM modules in its row, a copy of the variable will be available in each column of the MPCAM, where it can be searched for and read by each core independently and simultaneously. The DPCAM modules of the MPCAM work as a scratch book where the core can write all the version of the shared data without the risk of being overwritten until the subsequent (K-1) memory lines of the module are written to. K is the number of memory lines in the module. So, as all the versions of each variable is available, each has its unique tag, the core can access the version it needs at any time and in any order, as far as the version is available. A 32-bit tag provides a unique value for four Giga versions of variables. Therefore, the need for cache coherence is eliminated a thing which saves the non-compute time spent in executing the cache coherence protocols.

As the MPCAM is used as a big scratch pad with a very short access time, there would be no need for the global register. In this organization the (OF) and the (SB) of the core pipeline can be used to access the data in the local and the shared cache. The instruction fetch (IF) unit accesses the local instruction cache to fetch the instructions. The memory write (MW) and the memory read (MR) units can be dedicated to access the cache level L3 and the main memory outside of the processor. By this, we make sure that there is no competition among the pipeline units to access any part of the system.

As all tags of the DPCAM modules in the column of the MPCAM are compared with the applied tag simultaneously and as each line of the memory has its own comparator, the memory access time will be the same regardless of the module size and the number of modules in the rows and the columns of the MPCAM. This means that regardless of the N value, the access time of an (NXN) MPCAM, where N is the number of MPCAM in each row or column, will be the same as far as the semiconductor technology can accommodate it.

The organization presented in this paper proved to be expandable to many-core processor system

with a shared cache access time equivalent to twice the local cache access time. This organization will be presented in our future work.

References

[1] John L. Hennessy and David A. Patterson, "Computer Architecture A Quantitative Approach", Fourth Edition Standford University, 2007.

[2] A. Ayyad, I. Exman, M. Land, L. Rudolph, "An Experimental Cross-Bar Switch For Support Of Collective Communications In Parallel Processing", Proceedings of the Nineteenth Convention of IEEE in Israel, November 5-6, 1996.

[3] Barry Wilkinson, "Computer Architecture; Design and Performance", 2nd edition, Prentice Hall Europe, 1996.

[4] J. Archibald and J.L. Baer, "Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model", ACM transaction on Computer System, Vol 4, No.4, PP 273-298, 1986.

[5] B. Baterman, C. Freeman nad E. Resse, "A 450MHz 512KB second-Level cache with a 3,9GB/s data bandwidth ", 1998.

[6] Daniel Hackenberg Daniel Molka Wolfgang E. "Nagel Comparing Cache Architectures and Coherency Protocols on x86-64 Multicore SMP Systems", Center for Information Services and High Performance Computing (ZIH), MICRO'09, December 12–16, 2009, New York, NY, USA.

[7] "An Introduction to the Intel QuickPath Interconnect", Intel Corporation, January 30, 2009.

[8] Abdulkarim Ayyad, "The Design of a Special Purpose Dual Port Content Addressable Memory", Computer Architecture Lab Course Project Report, Al-Quds University, Palestine, 2011.

[9] Raymong Leong, Gary Green, "Dual-port content addressable memory", Assignees: Cypress Semiconductor Corporation, Patent number: US6122706, Application number: 08/172,575, Filing date: Dec 22, 1993, Issue date: Sep 19, 2000.

[10] The homepage of Altera. http://www.altera.com/

[11] Altera, "Stratix III Development Kit", Document Version: 1.1, San Jose, CA 95134,Agest 2008 (www.altera.com)

[12] Altera ,"Stratix III 3SL150 Development Board ", Refernce Manual , San Jose, CA 95134,May 2013.

[13] Michael E. Thomadakis, Ph.D.,"The Architecture of the Nehalem Processor And Nehalem-EP SMP Platforms", Supercomputing Facility, Research Report Texas A&M University, March, 17, 2011. miket@tamu.edu 2014.

[14] Robert Franz and Josef Weidendorfer, "Development of a Multicore Cache Simulator for Performance Analysis", bachelor thesis, Technical University Munich , july 2008.

[15] The homepage of Valgrind. http://www.valgrind.org