Does shared-memory, highly multi-threaded, single-application scale on many-cores?

Ghassan Almaless, Franck Wajsburt
LIP6 - UPMC Sorbonne Universités
4, place Jussieu – Paris, France
firstname.lastname@lip6.fr

Abstract

Nowadays, single-chip cache-coherent multi-cores up to 100 cores are a reality. Many-cores of hundreds of cores are planned in the near future. Due to the large number of cores and for power efficiency reasons (performance per watt), cores become simpler with small caches. To get efficient use of parallelism offered by these architectures, applications must be multi-threads. The POSIX Threads (PThreads) standard is the most portable way to use threads across operating systems. It is also used as a low-level layer to support other portable, shared-memory, parallel environments like OpenMP. In this paper, we propose to verify experimentally the scalability of shared-memory, PThreads based, applications, on Cycle-Accurate-Bit-Accurate (CABA) simulated, 512-cores. Using two unmodified highly multi-threads applications, SPLASH-2 FFT, and EPFilter (medical images noise-filtering application provided by Phillips) our study shows a scalability limitation beyond 64 cores for FFT and 256 cores for EPFilter. Based on hardware events counters, our analysis shows: (i) the detected scalability limitation is a conceptual problem related to the notion of thread and process; and (ii) the small per-core caches found in many-cores exacerbates the problem. Finally, we present our solution in principle and future work.

1 Introduction

Since ten years ago, the processors manufacturers reached physical limits concerning clock frequency and heat dissipation [4]. They turn today to multi-cores. Nowadays, multi-cores are common [15, 13] and single-chip cache-coherent multi-cores up to 100 are a reality [3, 8]. Many-cores with hundreds to thousands of cores are planned in the near future [10]. In the case of single-chip many-core, the bus interconnect, which becomes a bottleneck, is replaced by Network-on-Chip (NoC) interconnect like a bidirectional ring in Intel MIC architecture [1] or a 2D-mesh in Tilera architecture [8]. Both of these two industrial architectures are cache-coherent and they have backwards compatible Instruction Set Architecture (ISA) which makes their adoption easier by the software industry.

As the number of cores becomes large and for power efficiency reasons (performance per watt), cores become simpler each with its own CPU, FPU, TLB and L1/L2 caches. These per-core caches become relatively small with low associativity. This emphasizes the cache (TLB-I, TLB-D, L1-I, L1-D, L2) locality problem caused by a potentially higher miss rate. The traffic of the cache miss from one level might cause an interconnect and several cache-levels traversal with potential cache-coherence traffics. This impacts both the performance (more latency) and the power consumption (energy by moved bit). To get efficient use of parallelism offered by many-cores, applications must be multi-threads. Due to per-core small caches, a thread's working set has to be small and this can be achieved by decomposing a big working set among several threads, ideally, until it meets per-core cache size. As a result, performance driven applications should be significantly multi-threads. The POSIX Threads (PThreads) is the most portable way to use threads across operating systems. It is also used as a low-level layer to support other portable, shared-memory, parallel environments like OpenMP.

In this paper, we propose to verify experimentally the scalability of shared-memory PThreads-based applications. In particular, we investigate the effect of small caches on the scalability of single, multi-threads, shared-memory, applications to resolve a given problem using the biggest set of core allocated by an operating system. To this end, we did our experimentation using a full-system Cycle-Accurate-Bit-Accurate (CABA) simulation. In this accurate full-system simulation we used: (i) TSAR[1] a clustrized, cc-NUMA, many-core ar-

[1]TSAR (Tera-Scale ARchitecture) is a 4-years, MEDEA+, European funded project started in 2008, grant #2A718.

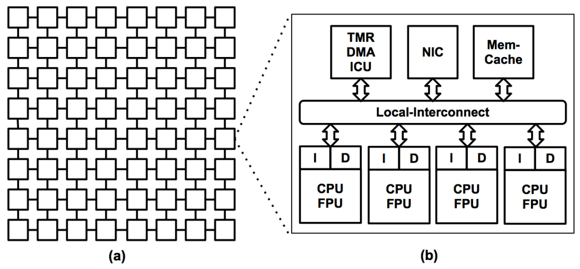

Figure 1: (a) TSAR clustered architecture with 2D-Mesh NoC; (b) A cluster of TSAR which contains: a local-interconnect, up to 4-cores, Network-Interface (NIC) to the NoC, Memory-Cache, multi-timers, 4-channels DMA and an Interrupt Control Unit (ICU).

chitecture configured to 512-cores; (ii) ALMOS (Advanced Locality Management Operating System), a new research operating system which we develop targeting cc-NUMA many-cores; and (iii) two unmodified shared-memory, highly multi-threads applications: SPLASH-2 FFT [5] and EPFilter (a medical images noise-filtering application provided by Phillips). Our study shows a scalability limitation beyond 64 cores for FFT and 256 cores for EPFilter. Based on hardware events counters, related to per-core L1 and TLB caches (instructions and data), our analysis shows: (i) the detected scalability limitation is a conceptual problem related to the notion of thread and process; and (ii) the small per-core caches found in many-cores exacerbates the problem.

The remainder of this paper is organized as follows. Section 2 introduces the experimental testbed and workloads, while Section 3 presents: (i) the experimental results including the speedup measurement; (ii) the detected scalability limitation and its analysis; and (iii) the description of the conceptual problem behind it. Section 4 describes our solution in principle, while Section 4 reviews related work. The conclusions and future work are presented in Section 6.

2 Testbed and Workloads

Our experimental evaluations are done using a cycle-accurate full-system simulation. In this section, we first describe the TSAR many-core and its CABA simulator. Then, we present the ALMOS operating system executed by the TSAR simulator. Finally, we present two evaluation workloads used in this experiments.

2.1 TSAR (Tera Scale ARchitecture)

TSAR [2] is an homogeneous, cc-NUMA (cache-coherent Non Uniform Memory Access) many-core architecture. It consists of up to 1024-clusters interconnected by DSPIN (Distributed, Scalable, Predictable, Integrated Network) a 2D-mesh NoC [18]. This homogeneous many-core has some common properties with a recent industrial many-core [8] such as small L1 cache size, distributed L2 caches, the choice of 32-bit cores, and the usage of 2D-mesh NoC with X-First wormhole packet-routing. Figure 1 illustrates TSAR clustered architecture.

A cluster of TSAR contains up to 4-cores, each of which has its own CPU, FPU and L1 separated (instruction and data) physical cache with MMU. The shared physical address space is 1 TB (40 bits physical address). It is distributed among clusters each of which is a home-cluster of its corresponding segment. The physical memory segment, homed by a given cluster, is accessed and cached by a specific per-cluster controller named Memory-Cache. The Memory-Cache can be seen as L2-cache with a coherence-directory and a memory-controller to access the cluster's memory segment via a separate and dedicated NoC. Each L1 can read and write to any cache-line of physical memory. If the requested physical address belongs to the core's cluster, it is a local request and the local Memory-Cache handles it. Otherwise, the request is a remote one and it is routed via the NoC to the target cluster. The target cluster is determined by decoding the MSB bits of the requested physical address.

In TSAR, cores can be of any simple RISC type. That is, a single-issue, short pipeline without any branch predictor nor out-of-order execution. TSAR memory subsystem is independent of cores type and there is a defined interface between the TSAR L1-cache and the used core. Each L1 has its own MMU with separated (instructions and data) TLBs. In order to be independent from core choice, the TLB MISS are handled by a hardware table-walk. TSAR page tables have two-levels where two page sizes are supported (2 MB and 4 KB). The coherence of L1 caches and their TLBs is guaranteed by a distributed directory based cache-coherence protocol named DHCCP (Distributed Hybrid Cache Coherence Protocol). If a given L1 writes to address X, the write is propagated (write-through strategy) to the home Memory-Cache of X cache-line. If the cache-line is not shared (references counter equal to 1) then the write is done. If the cache-line is shared with N L1-caches, then the write is blocked by the Memory-Cache until it takes the appropriate action. It sends a multicast-update command to L1-caches if $N < \lambda$, othwise it sends a broadcast-invalidate command to all L1-caches. The size of the cache-line is 64 bytes. An L1 can issue one to four words of 32bits in one write request. A cache miss issued by an L1 is always one cache line size.

TSAR architecture is prototyped with a Cycle-Accurate-Bit-Accurate (CABA) SystemC [12] based simulator. This simulator is able to do accurate full-system simulation starting from reset interrupt. The drawback of such an accuracy is the simulation time (2000 simulated cycles per second). At the command line, the simulator takes the X and Y widths and the number of cores per cluster. It supports up to 256 clusters (16x16) and provides up to 12 MB of per-cluster physical memory.

2.2 ALMOS

ALMOS stands for Advanced Locality Management Operating System. It is a new research operating system that we develop targeting cc-NUMA many-core with hundreds of cores. The locality of memory access impacts directly both the scalability and the power consumption. The main challenge is to enforce the locality of memory access made by threads of parallel applications. Although the locality enforcing needs a fine management of hardware resources (mainly cores and physical memory), ALMOS aims to hide the hardware topology and its resources management to applications. This allows POSIX shared-memory, parallel applications as well as legacy applications to benefit from performances offered by many-cores. ALMOS is a UNIX-like, POSIX compatible operating system. It currently has a fairly complete C library, a math library, a fairly complete PThreads library and the GNU OpenMP run-time. The kernel of ALMOS has the primordial subsystems related to tasks, virtual memory and files management. The PThreads implementation has a native support from the kernel and it has 1:1 threading model as in Linux. In this experimental study we use ALMOS for three reasons: (i) it is optimized and naturally available for TSAR; (ii) we have a fine control on its kernel and its behavior is fully predictable; and (iii) it has a similar threading model and implementation as a more complete and mature shared-memory operating system like Linux.

2.3 Evaluated Workloads

In order to evaluate the scalability of shared-memory, highly multi-threads single-application on 512-cores we selected two HPC-class applications: SPLASH-2 FFT and EPFilter. The choice of these applications is motivated by: (i) the need for scalable parallel applications allowing us to avoid the applications quality question in case of an abnormal results; (ii) the need for All-to-All inter-threads communication scheme that stress the NoC and various caches including TLB-I, TLB-D, L1-I, L1-D and Memory-Caches (L2); and (iii) feasibility - we need applications with relatively small execution time as we do an accurate full-system simulation.

Both applications are written in C and use PThreads. The FFT program is a complex, one-dimensional version of the ”Six-Step” FFT described by Bailey et al. [5]. The

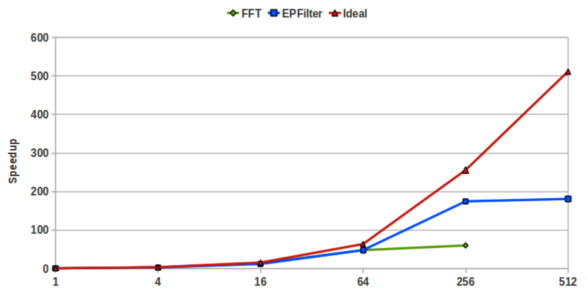

Figure 2: FFT and EPFilter Speedup, the number of cores is on x-axe.

EPFilter is an industrial medical image noise-filtering application provided by Philips. It consists of applying a convolution filter of 201x35 pixels on an image 1024x1024 pixels of 2-bytes. The application uses 4 memory-mapped regions. One of them is used to map the image file, while the other three regions are used as intermediate buffers where a pixel is extended on 4-bytes. FFT process 262144 (M=18) complex doubles.

3 Experimental results and analysis

Our TSAR configuration are as follow: (i) core type is MIPS32; (ii) L1-I and L1-D are each of 16 Kb, 4-ways; (iii) TLB-I and TLB-D are each of 16 entries, 4-ways; (iv) Memory-Cache of 256 Kb, 16-ways; and (v) Mesh of 8x16 = 128 clusters. This configuration is the same for all discussed experiments. Figure 2 shows the speedup measurement for FFT and EPfilter applications during their parallel phase. As we can see, the FFT is facing a speedup limitation beyond 64 cores while EPfilter speedup limitation comes beyond 256. It is important to recall the communication scheme of these two applications before further scalability investigation. FFT has an All-to-All communication scheme in which each thread reads intermediate results of all other threads before starting a new processing phase. EPFilter has the same scheme of communication but each thread writes each filtered pixel to other threads. Another point is how threads and physical memory placement is done in ALMOS. Threads are pinned to cores with one thread by core. When a thread faults on a virtual page, ALMOS allocates the requested physical page from the thread's cluster with a granularity of small page size (4 KB). Following this policy, all writes done by a core will be local for FFT, while in EPFilter, all reads will be local (except for some global variables). Finally, both of the evaluated applications are data-parallel where the parallel phase consists of several processing phases which alternate with global synchronization phases. The slower thread will determine the duration of the parallel phase.

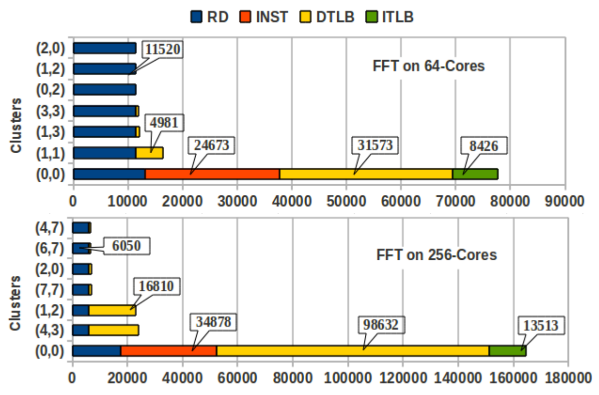

Figure 3: FFT histogram of remote cache-related requests received by clusters. Clusters as labeled by their coordination in the mesh. The requests number is shown on the x-axe. Requests are sorted in descending order.

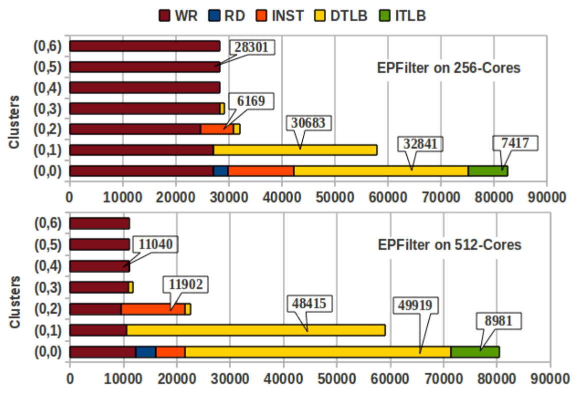

Figure 4: EPFilter histogram of remote cache-related requests received by clusters. Clusters as labeled by their coordination in the mesh. The requests number is shown on the x-axe. Requests are sorted in descending order.

3.1 Events Counters Measurement

In order to investigate the reason behind the detected speedup limitation we used TSAR hardware events counters related to its various cache requests. The inspected events are: L1 data miss (RD), L1 instruction miss (INST), L1 writes (WR), TLB data miss (DTLB) and TLB Instruction miss (ITLB). As described in section 2.1, the physical address space is distributed among clusters. The Memory-Cache of a given cluster can be seen as the only server of cache-lines homed by its cluster. Thus, in order to verify if the scalability limitation is caused by a physical pages allocation/placement mismatch, we focused on remote memory related requests issued by remote cores to each Memory-Cache.

Figures 3 and 4 show histograms of cache-related remote requests received by each cluster. The requests are sorted in descending order. All absent clusters in each histogram, have the same requests details as the last one listed. Each figure shows two requests histograms corresponding to the last two points in the scalability diagram of FFT and EPFilter shown in Figure 2. The first histogram of each figure corresponding to the last-scalability-point, that is, the cores number beyond which the speedup reaches its limit (64-cores for FFT and 256-cores for EPFilter).

3.2 Scalability limitation analysis

We classify the five instrumented requests in two categories: payload-requests and overhead-requests. Payload-requests are RD and WR, while overhead-requests are INST, DTLB, ITLB. By payload-requests, we designate requests issued by a core upon the processing algorithm executed by its thread (RD and WR). The overhead-requests are needed only to enable a thread to progress and continue processing its data.

Figure 3 shows a reduction of per-cluster payload-requests by a factor of 1.9 while the number of cores is multiplied by 4. In the other hand, the overhead-requests increased by a factor of 2.55. The treatment of the most of overhead-requests is done by the Memory-Cache (L2) of the cluster (0,0). The result is a serialization of overhead-requests treatment and therefore a speedup limitation. Figure 4 shows a reduction of per-cluster payload-requests by a factor of 2.57, while the number of cores is multiplied by 2. Although the overhead-requests increased by a factor of 1.38, their treatment still serialized, mainly on clusters (0,0) and (0,1). Figures 3 and 4 show that the communication scheme of FFT application (remote RDs, local WRs) and EPFilter (remote WRs, local RDs) is as expected, which indicates that the physical pages mapping various data used by both applications are well placed by ALMOS. The speedup limitation in both applications is due to the fact that the overhead-requests are centred on one or two clusters which become a bottleneck.

3.3 Scalability Drawback of Threads

The overhead-requests are mandatory and cannot be eliminated. DTLB and ITLB are related to the notion of paginated virtual address space. Moreover, their number is proportional to cores number. More threads implies more cores (true parallelism) thus more overhead-requests. The small size of caches in current and future many-cores exacerbates the problem because the rate of requests is inversely proportional to cache size. A virtual address space is related to the notion of a process. All threads of a given process share the same virtual address space. In a multi-threaded program, the process (representing an image of a program in execution) is just a resources container. The execution is defined by another entity which is a thread. In order to reduce the latency of DTLB and ITLB requests, the page tables have to be replicated on several memory banks, except that these page tables are unique for each process. Therefore, we cannot replicate them. Distributing them on different memory banks will reduce the latency of related miss but it will not resolve the problem. If there are multiple threads sharing the same segment of virtual address space, say 2 MB, all TLBs miss of all cores running these threads will request the same last-level page table and so it will be a serialization point. However, the first-level page table is looked up by all TLBs miss requests and It is also another serialization point. We have the same problem with physical pages mapping the program's instructions (INST miss requests). They can be replicated only in per-process bases, while they are shared by all threads of the same process.

4 Our solution in principle

We seek a software solution that resolves the detected scalability limitation and provides a cc-NUMA optimized implementation of PThreads. By its conforming to the PThreads interface, it will be compatible with current shared-memory programming paradigms and it can be reused in other shared-memory operating systems. Our solution in principle consists of introducing the notion of a task as a hybrid notion between thread and process. A task is a thread with its own virtual address space. The virtual address space of a task is partitioned mainly in three regions: (i) per-cluster shared region; (ii) global shared region; and (iii) per-task private region. The existence of per-task virtual address space enables the kernel to replicate the page tables and program instructions per-cluster bases. By mapping the global region at the same virtual address between all the process's tasks, the kernel gives the illusion of a single, shared virtual address space in a transparent manner to the application. All inter-tasks shared data like .bss, .data program sections and dynamically allocated memory (malloc) can be shared as usual. A task can do a private dynamic allocations in its private region without causing any contention with other tasks. The kernel can use the per-task private region to provide a dynamic stack. ALMOS replicates its kernel in each cluster and as the kernel code is mapped in per-process bases, threads cannot benefit truly from this replication because of their share of process page tables. Thus, when a thread executes a kernel code, all instruction miss will be redirected to the unique cluster containing a kernel replica mapped in the process's virtual address space. This will be not the case with tasks thanks to per-task virtual address space in which the kernel page tables can set to refer to local kernel replica in task's local cluster.

5 Related work

As the best of our knowledge, we are the first to establish a speedup-limitation causality between the effect of small caches found in current and future single-chip many-cores and the notion of threads sharing the same virtual address space as their process. However, prior work exists that attempts to improves performance of application based on the cache misses information [20, 22, 21]. TLBs are critical to processor performance [6, 14, 16] and hardware solutions exists [6, 9] to reduce miss traffics. These solutions were proposed to current multi-cores with large transistor budget per-core where in the current and future many-cores caches will be small. Big physical pages can be used to reduce DTLB and ITLB miss requests [19]. This can reduce the number of TLB entries used for a memory-mapped region. In the other hand it can cause a physical memory waste and it is less flexible for an operating system to manage the memory protection. Using big physical pages is also less flexible in placement strategies as a big page is located in one location, while small pages can be interleaved on more locations. Also, on demand page migration is more expensive for a big pages compared to small ones. They can cost I/O performance drawback [19]. Using big physical pages do not reduce the instructions miss requests. In Corey [11], the authors argues in favor of breaking of the UNIX/POSIX abstraction by (among other things) exposing the management of process's virtual address space to programmers of user applications. This is done by allowing programmers to explicitly control a kernel data-structure, named address range, so they can control the partitioning of the global virtual address space of a process. The goal of this approach is to improve the performance of page-fault handling and page tables modifications (TLBs shootdown). This technique has been used in Barrelfish [7] to construct a shared virtual address space between distributed entities named *dispatchers*. A dispatcher is an execution unite in user-mode and it is managed by cpu upcalls which is used in Psyche [17]. Both of these systems do not present their techniques in relation with TLBs and instruction misses latency in a many-core. As it is mentioned in the section 2.1, the TLBs coherence is insured by the hardware cache-coherency protocol so there is no need to any TLBs shootdown mechanism. Our analysis is based on cache (data, instruction and TLBs) miss traffics and our evaluated HPC-oriented applications are not sensitive to pages-faults performance related drawbacks. The performance drawback caused by pages-faults and TLBs shootdown are not relevant for a large number of HPC-oriented applications where each thread touches its data in the initialization phase and all threads are synchronized before starting the parallel processing phase.

Although our study argues also in the favor of reconsidering the process's virtual address space, we arrived to this conclusion from a completely another way by analyzing the cache misses traffics and their relation to the notion of threads and a process. Our solution introduces a third sharing level (by cluster) and addresses the problem of remote instruction miss traffics by replicating the code of applications by cluster basis. Our approach does not break the UNIX/POSIX interface as the management of a task's address space is done by the kernel in a transparent manner and the user-land (programmers, libraries and run-times) has no knowledge or control on it. This is a very interesting and suitable property as it insures a PThreads compatibility and do not requires any modification for the existent applications.

6 Conclusions and future work

In this paper, we have presented an experimental evaluation of the scalability of two shared-memory, highly multi-threads, applications, on 512-cores based many-core. We have identified a scalability limitation beyond 64 cores for the SPLASH-2 FFT and 256 cores for EP-Filter. Based on hardware cache-related events, our analysis has shown that: (i) the detected scalability limitation is due to the treatment serialization of TLBs and instructions miss requests and not to data physical pages miss-placement; (ii) the traffics of these miss requests are proportional to threads number; and (iii) the problem behind the scalability limitation is related to the notion of thread and process where a virtual address space is unique and defined per-process bases and it is shared between all process's threads. Moreover, the many-cores tends to use small per-core caches which exacerbates the problem as the traffics of the cache miss requests are inversely proportional to its size. In order to reduce the latency of TLBs and instruction related miss requests, we have proposed a solution in principle to resolve the detected scalability limitation in software. This solution aims to provide a NUMA-optimized implementation of PThreads interface for many-cores and it is articulated around the notion of the task as a new kernel hybrid entity between thread and process. We are currently implementing this solution in ALMOS and our preliminary evaluations are promising and indicate the pertinence of this solution.

References

[1] Intel many integrated core architecture. http://www.many-core.group.cam.ac.uk/ukgpucc2/talks/Elgar.pdf.

[2] Tera-scale architecture. https://www-asim.lip6.fr/trac/tsar/wiki.

[3] Tile-gx processors family. http://www.tilera.com/products/TILE-Gx.php.

[4] AGARWAL, V., AND AL. Clock rate versus ipc: the end of the road for conventional microarchitectures. In *Proceedings of the 27th annual international symposium on Computer architecture* (New York, USA, Jun 2000), ISCA '00, ACM, pp. 248–259.

[5] BAILEY, D. H. Ffts in external or hierarchical memory. *J. Supercomput. 4* (March 1990), 23–35.

[6] BARR, T. W., COX, A. L., AND RIXNER, S. Translation caching: skip, don't walk (the page table). *SIGARCH Comput. Archit. News 38*, 3 (June 2010), 48–59.

[7] BAUMANN, A., AND AL. The multikernel: a new os architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (Big Sky, Montana, USA, Oct 2009).

[8] BELL, S., AND AL. Tile64 - processor: A 64-core soc with mesh interconnect. In *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International* (feb. 2008), pp. 88 –598.

[9] BHATTACHARJEE, A., AND AL. Shared last-level tlbs for chip multiprocessors. In *HPCA* (2011), IEEE Computer Society, pp. 62–63.

[10] BORKAR, S. Thousand core chips: a technology perspective. In *Proceedings of the 44th annual conference on Design automation* (San Diego, California, USA, Jun 2007).

[11] BOYD-WICKIZER, S., AND AL. Corey: An operating system for many cores. In *In Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation* (2008).

[12] BUCHMANN, R., AND GREINER, A. A fully static scheduling approach for fast cycle accurate systemc simulation of mpsocs. In *Microelectronics, 2007. ICM 2007. Internatonal Conference on* (dec. 2007), pp. 101–104.

[13] CHARLES, J., JASSI, P., ANANTH, N. S., SADAT, A., AND FEDOROVA, A. Evaluation of the intel core i7 turbo boost feature. In *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)* (Washington, DC, USA, 2009), IISWC '09, IEEE Computer Society, pp. 188–197.

[14] CHEN, J. B., BORG, A., AND JOUPPI, N. P. A simulation based study of tlb performance. In *ISCA'92* (1992), pp. 114–123.

[15] CONWAY, AND AL. Cache hierarchy and memory subsystem of the amd opteron processor. *IEEE Micro 30* (March 2010), 16–29.

[16] KANDIRAJU, G. B., AND SIVASUBRAMANIAM, A. Going the distance for tlb prefetching: an application-driven study. *SIGARCH Comput. Archit. News 30* (May 2002), 195–206.

[17] MARSH, AND AL. First-class user-level threads. *SIGOPS Oper. Syst. Rev. 25* (September 1991), 110–121.

[18] MIRO-PANADES, I., GREINER, A., AND SHEIBANYRAD, A. A low cost network-on-chip with guaranteed service well suited to the gals approach. In *IEEE 1st Internationnal Conference on Nano-Networks* (2006).

[19] NAVARRO, J., IYER, S., AND DRUSCHEL, P. Practical, transparent operating system support for superpages. In *SIGOPS Oper. Syst. Rev* (2002), pp. 89–104.

[20] WEISSMAN, B. Performance counters and state sharing annotations: a unified approach to thread locality. *SIGPLAN Not. 33* (October 1998), 127–138.

[21] WOO, S. C., AND AL. The splash-2 programs: characterization and methodological considerations. *SIGARCH Comput. Archit. News 23*, 2 (May 1995), 24–36.

[22] ZHANG, X., DWARKADAS, S., AND SHEN, K. Towards practical page coloring-based multicore cache management. In *Proceedings of the 4th ACM European conference on Computer systems* (New York, NY, USA, 2009), EuroSys '09, ACM, pp. 89–102.