FELI: HW/SW Support for On-Chip Distributed Shared Memory in Multicores

Carlos Villavieja[1,2], Yoav Etsion[2], Alex Ramirez[1,2], and Nacho Navarro[1,2]

[1] Universitat Politecnica de Catalunya, Barcelona, Spain
[2] Barcelona Supercomputing Center, Barcelona, Spain
{first.last}@bsc.es

Abstract. Modern Chip Multiprocessors (CMPs) composed of accelerators and on-chip scratchpad memories are currently emerging as power-efficient architectures. However, these architectures are hard to program because they require efficient data allocation. In addition, when running legacy applications on these architectures, unless their code is adapted to utilize the distributed memory architecture, applications cannot benefit from their high computational power.

In this paper, we propose FELI, a set of operating system mechanisms that allocate application data to on-chip memories without any user intervention. FELI[1], automatically maps data to on-chip memories using the address translation mechanism. It relies on a set of TLB counters, and dynamical migration of pages from off-chip memory to on-chip memory. We also introduce virtually tagged L0 caches to alleviate the address translation overhead. Moreover, we make a comparison in performance and power consumption versus a homogeneous cache-based CMP design.

Our evaluation shows a 50% average improvement in power consumption with the scratchpad-based CMP compared to a cache-based CMP. And a 10% in average memory access time even accounting for the cost of page migrations and TLB invalidations. FELI can automatically allocate on-chip memory to an average of 90% of the applications working set.

Keywords: Chip MultiProcessors, Scratchpad on-chip memories, page migration.

1 Introduction

In recent years, the power wall has led to the emergence of Chip Multiprocessors (CMPs). However, the memory wall still remains a problem on CMPs, as more cores have to be fed with data. Cache-based architectures have mainly been used to alleviate high memory latencies. However, cache memories have unpredictable access time and do not scale well as the number of cores increases in scenarios with a high degree of data sharing. Cache coherency protocols become an issue.

[1] FELI is an acronym for *Fitting Everything Local In*, which is the philosophy of the proposed migration mechanism.

E. Jeannot, R. Namyst, and J. Roman (Eds.): Euro-Par 2011, LNCS 6852, Part I, pp. 280–292, 2011.

Scratchpad-based architectures are becoming a promising alternative [2]. They are software managed, have a predictable access time, and they are not as power hungry as cache memories. Digital Signal Processors (DSPs), the Cell/BE or GPU computing platforms already integrate on-chip scratchpad memories like Local Storage (LS) or shared memory areas. Applications use scratchpad memories to fetch specific application data through direct memory access (DMA) and then execute a kernel with all necessary data already present on-chip. After execution, results are usually transferred back to off-chip memory. This allows applications to benefit from lower latencies and predictable access time. Moreover, this memory architecture allows to overlap DMA data transfers with computation which can completely hide memories latencies. In addition, scratchpad memories do not require any coherence protocol and, thus are highly scalable.

Even the emergence of newer parallel programming models, there is still a large number of parallel legacy applications that do not benefit from these architectures. Without code modifications, all application data is located off-chip and all accesses pay the full main memory access latency. These applications were neither designed nor programmed for this kind of physically distributed memory. Therefore, in order to run these applications on scratchpad-based architectures efficiently, novel solutions need to be applied. Memory management in runtime libraries and/or at the Operating System (OS) level are crucial in order to transparently take advantage of these software-managed on-chip memories.

In this paper, we introduce the concept of Local Partition (LP) as a scratchpad memory attached to each core memory management unit(MMU). Under our shared memory physically distributed design, all cores can reference and access any LP inside the CMP and off-chip. The system builds a single global(physical) address space that includes all on-chip and off-chip memories. We show that allocating data through local and remote LP on a chip is beneficial for application performance. In this direction, we introduce FELI, a set of OS mechanisms to transparently perform effective memory allocation and remapping. The OS automatically moves data structures to those scratchpad memories that provide the best memory access times. We describe and evaluate a simple page migration mechanism based on page access counters stored in all entries of the cores Translation Lookaside Buffers (TLBs). Through the memory translation mechanisms data can be mapped to any on-chip/off-chip location. This allows the execution of legacy code in these scratchpad-based CMP architectures with a reasonable average memory access latency. All memory operations require the TLB for address translation and at the same time data mappings to any physical location. To alleviate the power consumption and overhead of the MMU that is introduced by this mechanism, we also introduce a virtually tagged level 0 data cache to avoid most of address translation requests. Finally, to evaluate our proposal, we have also performed a comparison with a traditional cache-based CMP architecture.

Contributions: To the best of our knowledge, this is the first paper that proposes a system that allows to run legacy applications unmodified on a CMP

architecture with scratchpad memories using a single global address space (on-chip Distributed Shared Memory). We make the following contributions:

- We show that legacy (shared-memory) applications can run unmodified in an on-chip Distributed Shared Memory (DSM) architecture without performance penalties.
- We show that a simple page migration policy can move most of the application working set in on-chip memory.
- We introduce a virtually tagged L0 exploring its size and invalidation mechanisms. This avoids TLB lookup and address translation for 70-80% of all memory operations.
- We discuss the characteristics of the OS mechanisms and the hardware (TLB) modifications proposed. These mechanisms enable the implementation of high efficient memory allocation policies in FELI.
- We evaluate FELI on a scratchpad-based CMP and compare the performance and power consumption with an homogeneous cache-based CMP.

2 On-Chip Distributed Shared Memory

2.1 Chip MultiProcessor Architectures

Figures 1 and 2 illustrate the two baseline CMP architectures used in this paper. Both are composed of multiple cores and several on-chip memories connected through an interconnection network to off-chip main memory. Figure 1 shows a cache-based architecture with a private L1 and a partitioned shared L2 cache per core. Figure 2 shows a scratchpad-based architecture with an addressable on-chip memory per core. In this architecture, we propose using a single global physical address space to map together all on-chip and off-chip memories. The on-chip scratchpad memories attached to each core from now on will be called Local Partitions (LPs). Since there is a single global physical address space, any core can access any LP in the CMP with a single load/store instruction or through DMA operations. All cores incorporate a Memory Management Unit (MMU) that includes a Translation Lookaside Buffer (TLB) for address translation, a Network interface Controller (NiC) for packet routing, and a programmable DMA engine to transfer data between local and remote memory (either the LP of another core or the off-chip memory). Indeed, DMA is a key element that allows the application or the Operating System(OS) to program data transfers with the benefit of overlapping program computation and communication. No coherence protocol is implemented for the LPs, so data replication or data migration need to be explicitly performed by software, through DMA or remote read/write operations.

2.2 Single Global Address Space On-Chip

Using a single global address space is essential to transparently manage data transfers. It makes the architecture fully coherent and only one copy of each

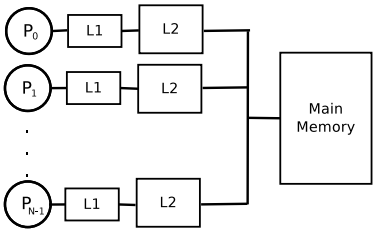


Fig. 1. Cache-based CMP architecture

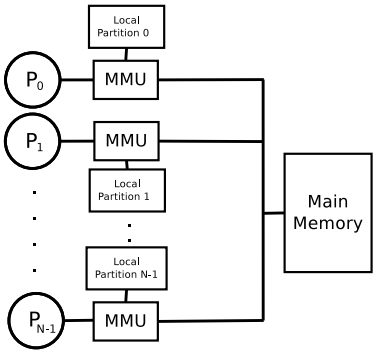


Fig. 2. Scratchpad-based CMP architecture

data is stored. Using address translation, the OS can map any virtual address to any physical location(on-chip or off-chip). In Figure 3, both address spaces are shown. Virtual address space in the top part is split in several parts or areas. Each area named after the letters A to E, represent a virtually contiguous set of pages. The physical address space in the bottom part is split in the physical memory locations. The virtual address areas are mapped as follows: Area A is fully mapped to the off-chip Main Memory. B and C areas are both mapped to the same LP 0. D and E areas are mapped to other LPs. Since we rely on the address translation mechanism to map virtual addresses to physical, the minimal area size is the page size.

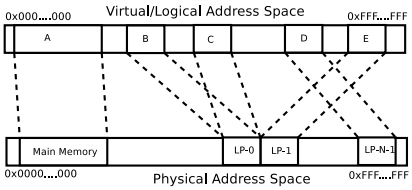


Fig. 3. Example of virtual to physical mapping

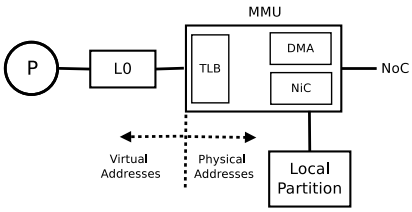


Fig. 4. L0-MMU-core architecture

2.3 FELI - Operating System Support for Locality Management

We have built a set of OS mechanisms to run legacy applications transparently and efficiently on top of the hardware described in Section 2.1. These mechanisms are called FELI.

In a scratchpad-based architecture, an application starts running with all its data loaded at the off-chip memory. A core that needs any data requires an access to the off-chip memory paying the highest access latency. No access to the on-chip memory or LP is done. FELI sets a *page migration* mechanism to alleviate this problem.

On a first data-page access, the requester core's TLB misses for the address translation, and a page-table request is sent to main memory. Next, the page-table request returns the address translation for the page, and the core TLB stores the translation and an access counter. Using the physical address a request is sent to main memory to obtain the data. From this moment on, each subsequent access to that page increases the counter in the TLB entry. The counter is required to monitor when the page is being actively used.

Once the access counter reaches a certain threshold (MMU threshold register), a page migration process is triggered. The threshold value is based on different costs (migration penalty and off-chip main memory accesses) commented in Section 2.5. The page migration allows to have a *zero copy* policy of all data. Each data has a unique location. Using this mechanism, FELI moves data from off-chip memory to the on-chip LP and vice-versa. This way, FELI tries to keep the most used data by each core in its LP. We do not consider page migrations between Local Partitions.

The migration mechanism must ensure memory coherence, therefore the following transaction needs to be performed in order:

1. **Invalidate the TLB entry:** Once the threshold is reached, and we start the page migration, the TLB page entry of the requester core is invalidated.
2. **Invalidate page-table entry:** In order to prevent other cores from requesting the page translation, the page-table entry is invalidated and locked. Translation requests from other cores for that page are stalled.
3. **Invalidate all TLBs:** Once the request to the page table (2) is acknowledged, the core holds the token for the page, and sends a broadcast message to invalidate all TLBs within that entry. This is commonly known as a TLB shootdown process. It includes waiting for all cores acknowledgments.
4. **Victim selection:** We choose a victim page from the LP, and perform a TLB shootdown for the victim page.
5. **Page Transfer:** Once both pages are selected and invalidated, we program the DMA controller to transfer both pages. The victim page from LP to main memory, and the new page from main memory to LP.
6. **DMA End:** Once we receive the DMA finalization signal, we update the page translation entry in the page table and unlock the entry. The next access to the page will miss in the TLB and it will be updated through a page-table request.

2.4 L0 Cache

Using LPs and FELI, we reduce the average memory access time by allocating most of the application's working set on-chip. However, all memory operations pay for the overheads in time and power usage [5] of address translation and request routing at the MMU. Address translation depending on the TLB configuration may take between 2-4 cycles [17]. To minimize the MMU overhead, we have introduced a small virtually tagged cache (L0) attached to each core. Figure 4 shows the L0 cache architecture connected to a core MMU. This L0 cache

only stores data located at the core LP. Therefore, we skip cache coherency with other cores. On a load miss in the L0, the data is requested to the MMU. The data response to the load missed in the L0 is only cached in the L0 if its address is hold at the core's LP. On a store operation, if it hits in the L0, the store is write-through to the LP.

Coherency at L0-LP level is only required for all stores to a LP from a remote core, and for all page migrations. On any of these two operations, the affected address is physical, however the L0 cache is virtually tagged. Since we do not have a reverse translation mechanism because of its high cost, invalidation operations must be performed. For this purpose, we have studied three different invalidation policies:

1. Total L0 invalidation: On an invalidation operation, the entire L0 cache is invalidated.
2. Perfect L0 invalidation: On an invalidation operation, using a reverse mapping we invalidate only the cache lines affected. This technique is easily implemented in our simulator, however, we do not consider it for a viable hardware option due to its real cost.
3. Bit filter L0 invalidation: The MMU holds a bitmap register to monitor all LP pages with some data cached in the L0. Each bit represents a LP page. On a local load request to an address stored in the LP, the bit field for that page is marked. On an invalidation request, we only invalidate pages marked in the bitmap. On a page migration the register is reset. This mechanism minimizes the performance impact of invalidations.

The MMU/TLB monitors requests to the LP to skip synonyms being allocated in the L0 cache. This way L0 can not hold two different virtual addresses pointing to the same data and therefore avoid conflicts. Only data from one page of the possible synonyms can be allocated at the L0. This leaves our L0 cache free of synonym conflicts.

2.5 Discussion on DSM Architecture Parameters

In all system architectures where page migration or page replacement occur, it is necessary to use a page replacement algorithm. In FELI, we have studied three well-known page replacement algorithms: *Random*, *FIFO* and *LRU*. As previously studied [19], the victim selection algorithm becomes irrelevant on configurations with enough space for the application working set. Our baseline 256KB LP can easily allocate application's working set on-chip, while it is a common L2 cache size. We use *random* because it requires less hardware complexity. Previous work from Etsion et al. [8] also confirm using *random* selection is efficient.

We have also quantified the overhead of adding a two bit counter to all TLB entries using CACTI [12]. The total area increase of TLB size is negligible (below 0.1%). In addition, the two bit counter does not affect the TLB access latency.

Another parameter to take into account is the page size used for our evaluation. Initially we chose 4K as the usual size for pages. Several experiments show

that the evaluated benchmarks suffer from internal fragmentation with larger page sizes. Moreover, when using larger sizes, we increase the number of accesses to remote LP increasing the overall memory access time. This translates to a performance degradation as a result of having more accesses to remote LP than LP.

We performed an exploration of the page-counter threshold value used to trigger page migration. Our experiments show that it is necessary to have a small threshold value to detect active pages in the working set of the benchmarks evaluated. For the results shown Section 4, we use a threshold value of two. A small value of the threshold ensures that pages in the LP are the most used ones from the benchmark. This parameter might vary if larger memory page sizes are used.

3 Methodology

Workloads: We use ten benchmarks (blackscholes, bodytrack, dedup, ferret, fluidanimate, rtview, streamluster and swaptions) from PARSEC suite [3] to evaluate the hardware and OS policies presented in this paper. We have selected a set of the most representative benchmarks based on the degree of data sharing between all threads. We used simlarge input set for PARSEC. We have used PIN [11] to obtain all applications traces. All benchmarks are evaluated with configurations of 32 threads. Using a methodology proposed by Casas et al. [4], and validated against SimPoint [9], we selected the most representative part of all benchmarks. Each benchmark trace contains 5×10^9 instructions.

Table 1. Table of the simulation configuration parameters

CMP Size	**32 cores**
TLB	**128 - 4-way - 1 cyc**
L0	**4KB - 4-way - 1 cyc**
LP	**256KB - 3 cyc**
Remote LP	**4 + NoC**
L1	**32KB - 4-way - 2 cyc**
L2	**256KB - 8-way - 7 cyc**
Remote L2	**7 + NoC cyc**
Main Memory	**250 cycs**
NoC	**crossBar**
NoC	**25 cyc - 25.6GB/s**

Table 2. Estimated breakdown of all operation costs involved in a page migration

Operation	**Cost (cyc)**
TLB inv	**50**
PageTable inv	**250-400**
CMP TLB inv	**200-300**
Program DMA	**10**
Transfer	**512**
Migration	**2000**

Simulator: For the evaluation of the memory architecture, we have used TaskSim [15], a trace-driven multicore simulator. TaskSim targets the simulation of parallel applications. It allows performing detailed simulations of all components in the CMP architectures shown in Figures 1 and 2. All memory accesses behave the CMP architectures defined in Section 2.1. TaskSim simulates all latencies for all components in the architecture, including memory controllers and

the interconnection network. All memory configurations and latencies in Table 1 have been obtained using CACTI [12]. Table 2 shows an approximated cost for each operation concerning the implemented page migration mechanism. Based on real measurements we have estimated the total cost of a page migration to 2000 cycles.

Metrics: To evaluate the CMP system performance, we measure the average access time per memory location(AMAT). We have also evaluated the power consumption of both CMP architectures. We account for static and dynamic power of all memories on both CMPs architectures.

4 Experimental Evaluation

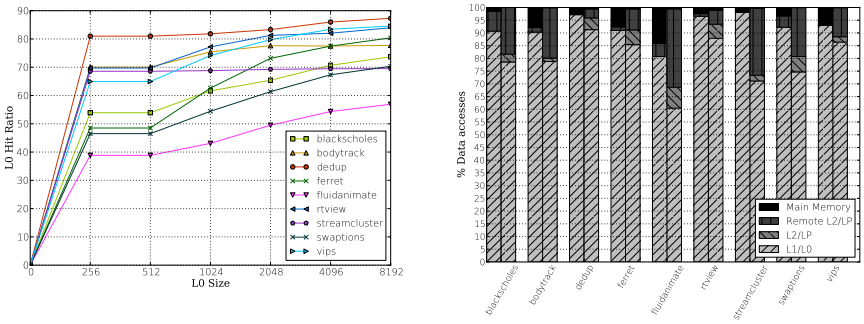
Both of the CMP architectures evaluated are similar in terms of on-chip memory capacity, however, memory access latency and power consumption are quite different as we can observe in this Section.

All the scratchpad-based CMP configurations use a page migration threshold of two. Discussion on choosing this value can be found in Section 2.5. For the L0 coherency, we use the Bit filter invalidation policy described in Section 2.4 because it is high efficient at a reasonable performance cost compared to perfect L0 invalidation which requires a reverse-mapping mechanism, or invalidating the whole L0 cache which reduces the L0 performance dramatically.

First, we analyze the performance of the new L0 cache introduced to overcome the overhead on address translation. Figure 5(a) shows the L0 cache hit ratio as we increase its size. The X-axis represents different L0 cache sizes, and the Y-axis shows the hit ratio. Each line of the figure represents a single benchmark. As it can be observed, at size 4KB, the L0 cache size hit ratio stabilizes around 70%. We use 4KB size and not larger to maintain a small 1 cycle latency. In addition, using a small L0 cache allows to avoid huge performance degradation and power consumption when invalidating the L0 because of page migrations or remote stores.

Figure 5(b) shows the percentage of memory accesses (Y-axis) that hit in the different types of memories in each CMP architecture. For all benchmarks shown in the X-axis, the left bar represents the cache-based and the right bar represents the scratchpad-based results. It can be observed that the L0 hit ratio for almost all benchmarks is above 75%. Although the L0 cache size is much smaller (4KB to 32KB) than the L1 cache, the L0 hit ratio is in average just a 10% lower. Moreover, for most applications the hit ratio for remote LP is higher than L2 cache. Page migration on other cores allows to allocate most of the application working set on-chip minimizing accesses to off-chip main memory. These results demonstrate that combining the L0 cache with the page migration mechanism to LP is a very efficient allocation policy. However, cache line replacement outperforms on high irregular memory access pattern applications (p.e: streamcluster).

In order to compare the two CMP architectures, we have evaluated the average memory access time (AMAT) and the power consumption for the cache-based



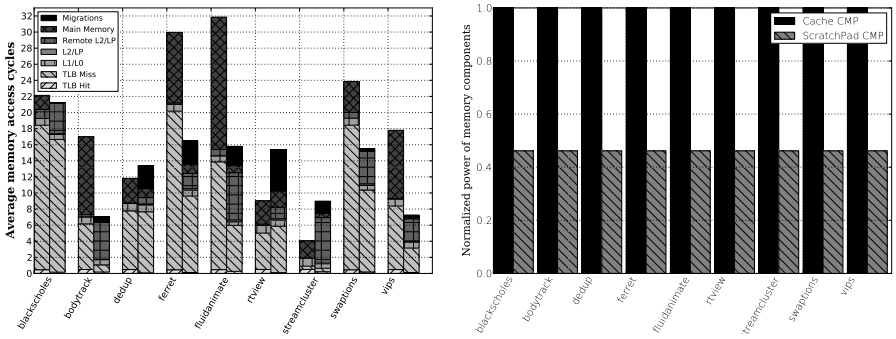
(a) Evaluation of the L0 Hit Ratio/Size for the LP CMP architecture.

(b) Applications Data Layout comparing cache/scratchpad-based CMP.

Fig. 5. In the left graph, we observe an evaluation of the L0 Hit Ratio/Size for the LP CMP architecture. In the right graph, we observe the Applications Data Layout comparing cache/scratchpad-based CMP.

and scratchpad-based CMP. Figure 6(a) shows the AMAT which represents in terms of memory latency, which memories take most of accesses time over the application execution. The X-axis of the graph shows the benchmarks. For all applications two bars represent the cache and the scratchpad-based respectively. The Y-axis shows the AMAT in cycles. The average access time for scratchpad-based is around 13 cycles compared to 14 for the cache-based CMP. It can be observed, in the bottom part of each bar, for scratchpad-based CMP the percentage of time for TLB Hits is negligible because of the high hit ratio in L0 cache. Cache-based CMP benefit by the performance of a higher L1 but with a higher cache latency and the addition of the address translation latency (1 cycle). Moreover, applications with less regular access pattern (bodytrack, fluidanimate, ferret, vips) have a lower L0 cache hit ratio. However, in these applications the remote LP hit ratio is more effective than the L2 cache. This is because using page migration more not yet used data is prefetched than on a L2 miss. For the cache-based case, we observe a high percentage of accesses to the off-chip memory.

Figure 6(b) shows an estimated power consumption comparison of both architectures. We used CACTI [12] to obtain the dynamic and static power consumption of all memory components of both architectures. The X-axis of the graph shows the benchmarks evaluated. As previous graphs, cache and scratchpad-based CMP are shown. The Y-axis shows the power consumption normalized to the architecture that consumes most. For all applications we can observe the power consumption of the scratchpad-based architectures is around 50-55% more efficient for all PARSEC benchmarks. The first reason why scratchpad-based architecture is much more efficient is the high hit ratio of the L0 cache. This allows to skip many lookups in the TLB, and since TLBs are power-hungry caches, the overall power consumption is highly reduced. Moreover, even if the L0 has a similar consumption to the L1 cache, the power consumption of the LP is much



(a) Average Memory Access Time (b) Power Consumption Comparison

Fig. 6. Average Memory Access Time graph and Power Consumption of all benchmarks for cache and scratchpad-based CMP architectures

better than that of an L2 cache. All benchmarks evaluated allocate most of their working set on-chip, and therefore the main source of dynamic power consumption comes from LP in the scratchpad-based CMP, while in the cache-based the main source for the dynamic power consumption are the L1 and TLB caches.

5 Related Work

Distributed shared memory (DSM) machines have largely been used in the past. However, as far as we know, FELI is the first to evaluate DSM in CMPs. DSM related work mainly explores page migration techniques. In [10] several techniques are described deeply dealing with memory management in local/remote memory nodes. Holliday et al. [10] refers to Aging as an effective page migration technique. Corbalan et al. [7] demonstrate than page migration is relevant to achieve high performance ratios in OpenMP parallel applications. However, they also demonstrate that process scheduling and affinity are key when page migration is considered. Since we are not considering OS context switch neither scheduling threads, thread affinity does not affect our results. Introducing thread scheduling would require remote LP migration to transfer remote pages to a new core LP when threads migrate. Dimitrios et al. [14] also uses page reference information to support page migration. They achieve a performance speedup of 264% for OpenMP parallel applications. As Corbalan et al. [7] they combine page migration with scheduling. In contrast to our solution, they consider compiler support to identify hot memory areas. In our initial experiments, we evaluated first touch and profile based technique to feed the OS with profiling information, however, our results using FELI obtained finer granularity over hot memory areas without the use of profiling. FELI is based on the TLB trigger and hence when page access reaches the threshold value, it obtains the application's hot memory areas dynamically at reasonable cost. Similarly Scheurich et al. [16] and Jeun et al. [20] present the pivot mechanism as an alternative

access reference based page migration algorithm. They use access information from each processor as a threshold and as the direction to migrate memory pages to other processors. These previous work has been done for Symmetric Multi-Processors(SMPs). Our work is concentrated in Chip MultiProcessors (CMPs) where latency and a high bandwidth is provided because of on-chip integration. Our work is still not comparable since we do not migrate pages between LPs.

Chaundri [6] presents PageNUCA, a set of OS-assisted locality management policies for large CMP NUCAS. It applies page-migration mechanisms for the last level cache of a CMP achieving a 12% of performance and energy improvement. This solution is closest to our scheme but it is designed for a cache-based architecture and it is only applied for shared L2 cache. We evaluated FELI over all on-chip memory.

Several work in scratchpad-based architectures are focused on compiler support to efficiently allocate data. Avissar et al. [1] and Nguyen at al. [13] present an optimal scheme to allocate data on scratchpad-based architectures in embedded applications. They use a compiler strategy that automatically partition application data among the memory units. Other solutions to improve memory allocation in scratchpad-based architectures can be found based on compile time techniques [18].

6 Conclusions

In this paper, we propose FELI, a set of Hardware/Software mechanisms to run legacy applications in a scratchpad-based CMP architecture. FELI uses a page migration mechanism to dynamically allocate data into on-chip addressable memories or Local Partitions (LPs). This is the first paper where DSM is used for scratch-pad based CMPs. Our mechanisms use address translation to map any virtual address to any physical address. In FELI, the OS manages the LPs as caches of 4KB lines, exploiting the predictable access time and the power efficiency of scratchpad memories. In addition, we have added a virtually tagged L0 cache to improve the overall memory latency. FELI automatically allocates around 90% of application data on-chip. On a performance and power consumption comparison with a cache-based CMP, FELI achieves an average reduction of 10% in memory access time and a reduction of 50% in power consumption.

Acknowledgments. This research is supported by the Consolider program (contract No. TIN2007-60625) from the Ministry of Science and Innovation of Spain, the TERAFLUX project (ICT-FP7-248647), and the European Network of Excellence HIPEAC-2 (ICT-FP7-249013). Y. Etsion is supported by a Juan de la Cierva Fellowship from Ministry of Science and Innovation of Spain. Special thanks to the members of the Heterogeneous Architecture group at BSC and the anonymous reviewers for their comments and suggestions.

References

1. Avissar, O., Barua, R., Stewart, D.: An optimal memory allocation scheme for scratch-pad-based embedded systems. ACM Trans. Embed. Comput. Syst. 1(1), 6–26 (2002)
2. Banakar, R., Steinke, S., Lee, B.S., Balakrishnan, M., Marwedel, P.: Scratch-pad memory: design alternative for cache on-chip memory in embedded systems. In: CODES 2002: Proceedings of the Tenth International Symposium on Hardware/Software Codesign, pp. 73–78 (2002)
3. Bienia, C., Kumar, S., Singh, J.P., Li, K.: The PARSEC benchmark suite: Characterization and architectural implications (October 2008)
4. Casas, M., Badia, R.M., Labarta, J.: Automatic structure extraction from MPI applications tracefiles. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 3–12. Springer, Heidelberg (2007)
5. Chang, Y.-J., Lan, M.-F.: Two new techniques integrated for energy-efficient tlb design. IEEE Trans. Very Large Scale Integr. Syst. 15, 13–23 (2007)
6. Chaudhuri, M.: Pagenuca: Selected policies for page-grain locality management in large shared chipmultiprocessor caches. In: In Proceedings of HPCA-15 (2009)
7. Corbalan, J., Martorell, X., Labarta, J.: Evaluation of the memory page migration influence in the system performance: the case of the SGI o2000. In: ICS 2003: Proceedings of the 17th annual international conference on Supercomputing, pp. 121–129 (2003)
8. Etsion, Y., Feitelson, D.G.: L1 cache filtering through random selection of memory references. In: PACT, pp. 235–244 (2007)
9. Hamerly, G., Perelman, E., Calder, B.: How to use simpoint to pick simulation points. SIGMETRICS Perform. Eval. Rev. 31, 25–30 (2004)
10. Holliday, M.A.: Reference history, page size, and migration daemons in local/remote architectures. SIGARCH Comput. Archit. News 17(2), 104–112 (1989)
11. Luk, C.K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V.J., Hazelwood, K.: Pin: building customized program analysis tools with dynamic instrumentation, pp. 190–200 (2005)
12. Muralimanohar, N., Balasubramonian, R., Jouppi, N.P.: Architecting efficient interconnects for large caches with cacti 6.0. IEEE Micro 28(1), 69–79 (2008)
13. Nguyen, N., Dominguez, A., Barua, R.: Memory allocation for embedded systems with a compile-time-unknown scratch-pad size. ACM Trans. Embed. Comput. Syst. 8(3), 1–32 (2009)
14. Nikolopoulos, D.S., Papatheodorou, T.S., Polychronopoulos, C.D., Labarta, J., Ayguadé, E.: User-level dynamic page migration for multiprogrammed shared-memory multiprocessors. In: ICPP 2000: Proceedings of the Proceedings of the 2000 International Conference on Parallel Processing, p. 95 (2000)
15. Rico, A., Cabarcas, F., Quesada, A., Pavlovic, M., Vega, A.J., Villavieja, C., Etsion, Y., Ramirez, A.: Scalable simulation of decoupled accelerator architectures. Tech. Rep. UPC-DAC-RR-2010-14, Universitat Politècnica de Catalunya (June 2010)
16. Scheurich, C., Dubois, M.: Dynamic page migration in multiprocessors with distributed global memory. IEEE Transactions on Computers 38, 1154–1163 (1989)

17. Swaminathan, S., Patel, S.B., Dieffenderfer, J., Silberman, J.: Reducing power consumption during tlb lookups in a powerpc" embedded processor. In: Proceedings of the 6th International Symposium on Quality of Electronic Design, ISQED 2005, pp. 54–58 (2005)
18. Udayakumaran, S., Dominguez, A., Barua, R.: Dynamic allocation for scratch-pad memory using compile-time decisions. ACM Trans. Embed. Comput. Syst. 5(2), 472–511 (2006)
19. Villavieja, C., Ramirez, A., Navarro, N.: On-chip distributed shared memory. Tech. Rep. UPC-DAC-RR-CAP-2011, Universitat Politècnica de Catalunya (February 2011)
20. Jeun, W.-C., Kee, Y.-S., Ha, S.: Improving performance of openMP for SMP clusters through overlapped page migrations. In: Mueller, M.S., Chapman, B.M., de Supinski, B.R., Malony, A.D., Voss, M. (eds.) IWOMP 2005 and IWOMP 2006. LNCS, vol. 4315, pp. 242–252. Springer, Heidelberg (2008)