ON THE SCALABILITY OF IMAGE AND SIGNAL PROCESSING PARALLEL APPLICATIONS ON EMERGING CC-NUMA MANY-CORES

Ghassan Almaless and 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 and many-cores of hundreds of cores are planned in the near future. This technological shift undertaking by the high-end computer-industry is converging with the design motivation of other domains like embedded and HPC industries. In this paper, we propose to investigate the scalability of the same four unmodified, shared-memory, image and signal processing oriented parallel applications on two targets: (i) embedded - TSAR, a single-chip 256-cores based, Cycle-Accurate-Bit-Accurate simulated, cc-NUMA many-core; and (ii) high-end - an AMD Opteron Interlagos, 64-core based, cc-NUMA many-core. Beside our scalability results on both cc-NUMA targets, our contributions include two operating system mechanisms: (i) a distributed, client/server based, scheduler design allowing the kernel to offer scalable inter-threads synchronization mechanisms; and (ii) a kernel-level memory affinity technique named Auto-Next-Touch allowing the kernel to transparently and automatically migrate physical pages in order to enforce the locality of thread's memory accesses. Although these two mechanisms are implemented and evaluated in ALMOS (Advanced Locality Management Operating System) running on the TSAR target, they remain applicable to other shared-memory operating systems.

1. INTRODUCTION

Since ten years, the high-end computer manufactures has reached physical limits regarding the clock-frequency and the heat-dissipation which forced them to turn to multi-cores designs [3]. Nowadays, single-chip cache-coherent multi-cores up to 100 cores are a reality [30, 8] and many-cores of hundreds of cores are planned in the near future [11]. This undertaken technological shift is converging with the design motivation of the embedded-computing industry regarding to energy efficiency and the already usage of multi/many-cores to get more parallelism and performances [32, 22, 23, 18, 17]. Meanwhile the HPC industry is looking closely with a high interest to many-cores as a way to get more performance and parallelism with lower energy cost [26, 16, 9]. The single-chip cache-coherent many-cores which are a general-purpose processors supporting the shared-memory programming paradigm, can constitute a common platform for these different domains for at least two reasons. The first, they ensure a soft transition for the software-industry by allowing legacy applications (sequential and parallel) to continue to run over the many-cores while new highly multi-threaded applications are emerging. The second, they reduce the time-to-market by allowing: (i) the reuse of the same general-purpose architecture (possibly with some adaptation) in different domains; (ii) the reuse of existing applications or computing kernels developed/used in other domains; and (iii) the significant reduction in new applications development time by the reuse of existing build tools and standard programming languages. Although the single-chip cache-coherent many-cores has this important potential, they come with constraints. As the cores number becomes large and for power efficiency reasons (performance per watt) cores become simpler with small L1 caches. The bus interconnect, which becomes a bottleneck, is replaced by a Network-on-Chip (NoC) interconnect while the LLC (Last Level Cache) is distributed on the chip. This causes a non-uniform latency when a core accesses to the logically-shared but physically-distributed memory (cc-NUMA). To get efficient use of parallelism offered by these many-cores, applications must be strongly multi-threaded. 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. In this cc-NUMA architecture the thread's data placement becomes a major issue as a miss placement can lead to sever performance drawback and more energy consumption (energy by moved bit). To maintain the single-chip general-purpose many-cores as an attractive point of convergence and to cope with their architectural constraints, a key point is the operating system regarding to its design, scalability and the execution environments which it provides to user applications.

In this paper, we propose to investigate the scalability

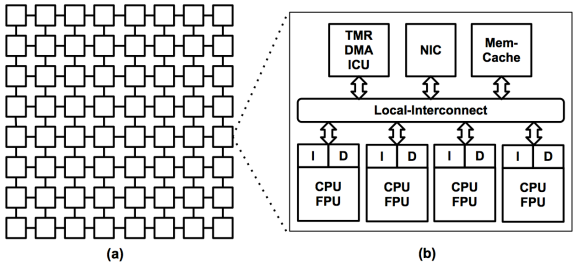

Fig. 1. (a) TSAR clustered architecture with 2D-Mesh NoC; (b) A cluster of TSAR which contains: a local-interconnect, up to 4-cores, Network-Interface (NIC) to the NoC, Memory-Cache, multi-timers, 4-channels DMA and an Interrupt Control Unit (ICU).

of the same four unmodified image and signal processing oriented parallel applications on two targets: (i) embedded - TSAR[1] a clusterized, cc-NUMA, single-chip many-core configured to 256-cores; and (ii) high-end - an AMD Opteron Interlagos, 64-core based, cc-NUMA many-core. The TSAR target is software-emulated using a full system, Cycle-Accurate-Bit-Accurate (CABA) simulator running ALMOS (a new research operating system targeting cc-NUMA many-cores). The AMD target is an industrial machine running a Linux-based operating system.

Beside our scalability results on both cc-NUMA targets, our contributions include two operating system mechanisms: (i) a distributed, client/server based, scheduler design allowing the kernel to offer a scalable inter-threads synchronization mechanisms; and (ii) a kernel-level memory affinity technique named Auto-Next-Touch allowing the kernel to transparently and automatically migrate physical pages in order to enforce the locality of thread's memory accesses. Although these two mechanisms are implemented and evaluated in ALMOS (Advanced Locality Management Operating System) running on TSAR target, they remain applicable to other shared-memory operating systems.

The remainder of this paper is organized as follows. Section 2 introduces the experimental testbeds and workloads. Section 3 presents our two kernel-level mechanisms related to scheduler design and memory-affinity. Section 4 presents the scalability results obtained on TSAR and AMD Opteron Interlagos many-cores while Section 5 reviews related work. The conclusions and future work are presented in Section 6.

2. TESTBEDS AND WORKLOADS

We conducted our scalability experiment on two platforms: 256-cores TSAR many-core running ALMOS and 64-cores AMD Opteron Interlagos running Linux. 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 before describing the AMD Opteron Interlagos target. Finally, we present the four evaluated image and signal processing workloads used in this experiment.

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 [21]. This homogeneous many-core has some common properties with a recent industrial many-core [8] such as the small L1 cache size, the 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 neither 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. The TLB MISSes are handled by a hardware table-walk. TSAR page tables have two-levels where two page sizes are supported (2 MiB and 4 KiB). 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 $N < \lambda$, otherwise 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

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

cache line size.

TSAR architecture is prototyped with a Cycle-Accurate-Bit-Accurate (CABA) SystemC [14] based simulator. This simulator is able to do accurate full-system simulation starting from reset interrupt. At the command line, the simulator takes the X and Y widths of the clusters mesh and the number of cores per cluster. All TSAR components are writing in RTL equivalent SystemC and they are used to co-simulated TSAR in SystemC and VHDL. The advantage of such accuracy is the precision in the results, which include all hardware resources contention. The drawback is the simulation time (2000 simulated cycles per second). In this study we use the flowing TSAR configuration: (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 8x8 = 64 clusters. This configuration is the same for all discussed experiments.

2.2. ALMOS

ALMOS [1] stands for Advanced Locality Management Operating System. It is a new research operating system targeting cc-NUMA many-core with hundreds of cores. It is intended to investigate the scalability of an operating system components on large-scale cc-NUMA many-cores [4]. 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. AMD Opteron Interlagos target

Our second target is a high-end many-core machine based on AMD Opteron Interlagos processor 6282 SE running Linux 2.6.39-4. This target is illustrated by the figure 2. Each Node consists of 4 computing modules. Each computing module has 2 cores having a common L1 instruction, per-core L1 data and a unified L2 cache. All of the for L2 caches share the node's L3 cache. The sizes of these caches are : 64 KB, 16 KB, 2048 KB, 12288 KB respectively. The system is made from 4 Interlagos processors interconnected with a full crossbar of Hyper-Transport-3.0 links (6.4GT/s). Each processor has the a total of 16 cores (2 nodes). It is made with 32nm technology and its consumption is estimated to 140 W. From operating system viewpoint, Linux detects 8 NUMA nodes of 8 cores each. In our experiment we pin explicitly threads to cores using Linux *pthread_setaffinity_np* function call.

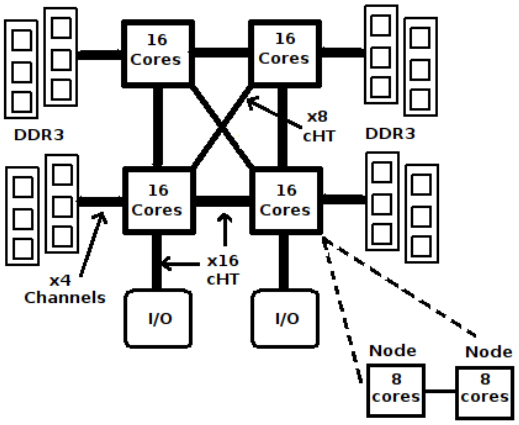

Fig. 2. AMD Opteron Interlagos 6282 SE. 4 chips interconnected by a full crossbar. The internal links are half bandwidth of the outer ones. Each chip has 2 nodes each of which has 8 cores.

2.4. Evaluated Workloads

In order to evaluate the scalability of image and signal processing parallel applications on emerging cc-NUMA many-cores, we selected this four applications: SPLASH-2 FFT, EPFilter, Histogram and Tachyon. All of these 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. [6]. The EPFilter is an industrial medical image noise-filtering application provided by Philips. It consists of applying a convolution filter of 201x35 pixels on its input image of 2-bytes per-pixel. The Histogram application, from Standford Phoenix project [24], generates the histogram of frequencies of pixel values in the red, green, and blue channels of a bitmap picture. The Tachyon application [28] is highly parallel ray tracing. Table refparam shows the input parameters used for these workloads. All of these applications follow the treatments scheme shown in figure 5.

3. KERNEL-LEVEL MECHANISMS

In this section we describe our two kernel-level contributions implemented in the kernel of ALMOS regarding the scheduling and physical pages placement. We first describe the distributed, client/server based, scheduler. Then we describe the memory affinity mechanism name Auto-Next-Touch.

Table 1. Input parameters of the evaluated workloads

Abbreviation	Comment
Histo8-A	Histogram executed on the target ALMOS/TSAR with 8.7 Mpix input image
Histo8-L Histo34-L	Histogram executed on the target Linux/AMD Opteron with an input image of 8.7 and 34.8 Mpix
EP1024-A EP2048-A	EPFilter executed on the target ALMOS/TSAR with an input image of 1 and 4.2 Mpix
EP1024-L EP2048-L	EPFilter executed on the target Linux/AMD Opteron with an input image of 1 and 4.2 Mpix
M18-A	FFT executed on the target ALMOS/TSAR with 262144 complex points
M18-L M20-L	FFT executed on the target Linux/AMD Opteron with 262144 and 1048576 complex points
512-A 1024-A 2048-A	Tachyon executed on the target ALMOS/TSAR with a scene (2 balls) resolution of 512x512, 1024x1024 and 2048x2048
512-L 1024-L 2048-L	Tachyon executed on the target Linux/AMD Opteron with a scene (2 balls) resolution of 512x512, 1024x1024 and 2048x2048

3.1. Distributed client/server based scheduler

Threads scheduling functionality provided by the kernel of an operating system allows, mainly, to temporally multiplex the execution of runnable threads on the available cores. To reduce the contention on the ready queue of threads (where runnable threads are attached), current SMP kernels, like Linux, has a ready queue by core protected by a lock to ensure the mutual exclusion in the case of contention. This contention can occur while two threads running on two different cores try to update (insert/delete) a core's ready queue. For instance, if a thread running on core A wants to wakeup another thread X belonging to a core B, it has to take first the lock protecting the ready queue of core's B scheduler. Then it has to: detach the thread X from core's B waiting queue; update thread X timing information; compute thread X new priority before inserting it to the ready queue of core B. Finally, it has to release the taken lock. These steps constitute the wakeup scheduling primitive (figre 3 (a)). Another scheduling primitive which needs to be executed in mutual exclusion is the election of a runnable thread for the next execution quantum. This scheduling primitive is executed when a thread wants to yield its current core or it is going to sleep. The kernel provides user applications synchronization mechanisms, like condition variables and barriers, which involve potentially all application's threads. The cost of executing the wakeup scheduling primitive for each application's sleeping thread can be substantial, specially in a cc-NUMA many-core because of the locking and the manipulation of remote data of each target scheduler.

In order to minimize the cost of scheduling primitives and to provide a scalable synchronization mechanisms, we propose in the kernel of ALMOS, a distributed client/server based scheduler. The main idea is to separate the wakeup event notification from its effective execution. If a thread running on a core A needs to wakeup a thread X belonging to core B, it just write a wakeup event to core B scheduler (figure 3 (b)). In a second time, the scheduler of core B executes this pending event (figure 3 (c)). Upon this scheme, the per-core scheduler acts as a server. It assigns to each thread, at the affectation time, a local identifier. All of the scheduling primitives are executed only by threads belonging to this scheduler at various scheduling points (e.g. when a thread yields its core or in the occurrence of clock interrupt). In this scheme, the clients are any thread looking to wakeup another thread.

a
```
Lock Core-A scheduler
Detach thread-X from the waiting queue
Update timing information of thread-X
Compute priority of thread-X
Insert thread-X in the ready queue
Unlock Core-A scheduler
```

b
```
Core-A->scheduler->pending_events[thread-X->id]=WAKEUP-EVENT
```

c
```
for all threads belonging to the current Core
    event = pending_events [ i ]
    if ( event == WAKEUP-EVENT )
        Detach thread-i from the waiting queue
        Update timing information of thread-i
        Compute priority of thread-i
        Insert thread-i in the ready queue
```

Fig. 3. Pseudo code showing the a summary of steps to do while executing wakeup operation in two approaches: (a) classical one with lock protection; (b/c) distributed one where (b) denotes the client side while (c) denotes the server side.

The direct benefits of this organization are: (i) the scheduler becomes lock-free; (ii) the wakeup scheduling operation can be done in parallel with yield/sleep operations without any contention; and (iii) the reduction in latency and interconnect traffics thanks to remote data manipulation avoidance.

In order to verify experimentally the scalability and performance of this distributed approach, we implemented, in the kernel of ALMOS, the two scheduler versions: the lock-based one named (v0) and the distributed lock-free (v1). In both versions the scheduling algorithm is round-robin with fixed priority and one ready queue per core. Using a 256-core TSAR, the figure 4 shows the cost of 3 barriers notification of the bench M18-A (table 1) for a threads number ranging from 4 to 256. The results show the efficiency and the scalability of the distributed client/server based scheduler over the lock-based one.

3.2. Memory affinity Auto-Next-Touch

We begin by identifying two problems related to the initialization phase in a typical treatment scheme that can be found in a parallel multi-threaded application. Then, we present a current solution Next-Touch which solve one of them. However, we show that this solution does not solve the second one

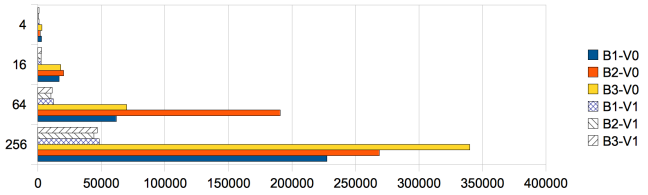

Fig. 4. Cost of barrier notification

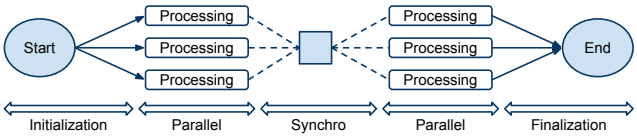

Fig. 5. A typical parallel treatment scheme.

and raises itself two new limitations. Finally, we present our solution, Auto-Next-Touch that overcomes these limitations and meet all of these problems.

A typical treatment scheme of a parallel multi-threaded application, shown in Figure 2, begins with a sequential initialization phase followed by parallel processing phase. The initial data are prepared in the initialization phase executed by a single thread. In order to control the placement of a physical page, the kernel use the First-Touch strategy. According to this strategy, when a thread accesses to a virtual address for the first time, the kernel allocates the requested physical page from the memory of the cluster in which the thread runs. Although the First-Touch strategy is adopted by default in all kernels, it has two problems. The first, it ignores the existence of a temporary sequential phase found in the majority of parallel programs. All physical page allocations needed to initialize global data are done locally from the cluster of the initiator thread but during the parallel phase, each worker thread will access remotely its own subset of these initial data in order to process them. These remote access during the parallel phase seriously impact the performance in a NUMA architecture. The second, it does not anticipate the memory needs of worker threads during the parallel phase. During the initialization phase, the memory allocations made by the initiator thread can potentially generate a local shortage of physical memory. This will impact the locality of memory allocations made in the parallel phase by threads assigned to the same cluster as the initiator one. This situation will cause the kernel to remotely allocate their requested physical pages, thus causing another performances drawback.

The idea of the Next-Touch strategy is to let to the programmer the care of explicitly indicating to the kernel, at the end of the initialization phase, the memory areas (virtual addresses) that are subject of parallel processing. The kernel does a page tables traversal of the process and marks each existing physical page corresponding to the virtual area for a migration on the next access. Thus, during the parallel phase, the physical pages containing the data to be processed by a thread will be migrated to local memory of the thread. However, beside the fact that this technique is absent on most operating systems because it is not standard (i.e not POSIX compliant), it has two limitations: (i) complexity of use, because the programmer must consider the memory access profile and informs the kernel about addresses of memory areas to be migrated on the next access; (ii) compatibility, because it needs to rewrite the existing applications. On the other hand, it does not solve the problem of potential dispersion of memory allocations during the parallel phase due to local physical memory shortage during the initialisation phase. Even though threads will ends up migrating their physical pages, the order in which new memory requests and page migrations arrives may not be priory known.

We propose a new solution that we call Auto-Next-Touch. It allows to (i) relieve the programmer from the task of specifying the memory areas to migrate during parallel phase; (ii) run an existing multi-threaded applications without any prior modifications; and (iii) ensure the availability of a local physical memory for parallel processing phase by imposing a memory allocation policy. Our solution relies on two techniques. The first one is to detect the moment when a single threaded process goes multi-threaded. At this point, the kernel automatically and transparently look for all process virtual regions belonging to the heap and the privately mapped. For each region it marks the page tables entries of all corresponding and existing physical pages to be migrated on the next access. The second one is to limit the amount of physical memory that a thread can locally allocates when it is the only thread of its process and to release the remaining when the process becomes multi-threads. This limitation of local physical memory quantities has the disadvantage of penalizing the performance of a single threaded application, since it introduces a dispersion of memory allocations and thus greatly degrades the locality of memory accesses. We have chosen to pay this price because we believe that the large majority of applications designed to run on many-cores architectures are multi-threaded.

In order to verify experimentally the relevance of this solution, we implemented it in the kernel of ALMOS. During the sequential initialization phase of SPLASH-2 FFT (M18-A), a set of memory areas are allocated and initialized. This implies a corresponding physical pages allocations from the local memory to the core on which the initiator thread runs. Figure 6 shows the speedup evaluation when executing the same unmodified application in two cases: when the Auto-Next-Touch is enabled in ALMOS kernel and when it is disabled which is the default behavior without any migration as it can be found in other kernels like Linux. The results show: (i) There is a scalability limitation beyond 16 cores in the case of a default kernel behavior preventing any farther performance; and (ii) This scalability limitation is resolved by the use of the kernel-level Auto-Next-Touch mechanism transparently, that

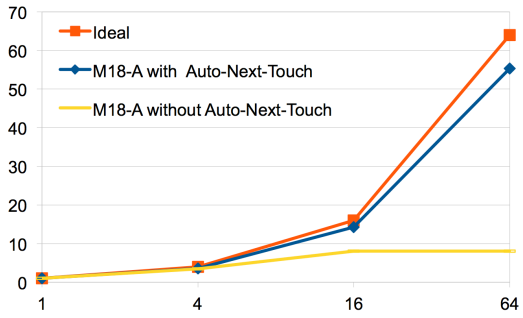

Fig. 6. Evaluation of Auto-Next-Touch technique (x-axis: number of cores; y-axis: speedup)

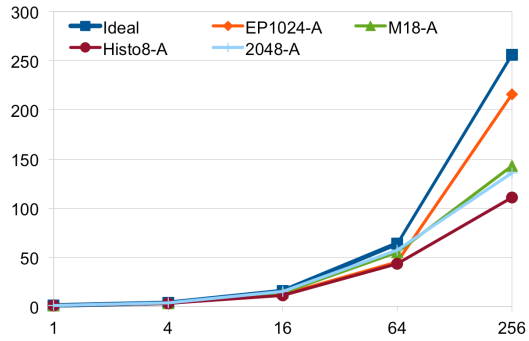

Fig. 7. Speedup evaluation on ALMOS/TSAR (x-axis: number of cores; y-axis: speedup)

is, without any prior modification of the user application.

4. EXPERIMENTAL RESULTS

In this section we present the experimental evaluation of four shared-memory, image and signal processing applications on two targets: (i) embedded - TSAR configured to 256-cores and running ALMOS operating system; and (ii) high-end - AMD Opteron Interlagos 64-cores running Linux 2.6.39-4. The evaluated application were not modified and they are compiled using gcc 4.4.3, MIPS32 barre-metal, cross compiler for the first target; and gcc 4.4.6, x86_64 on the second target. The gcc optimisation level used is -O3.

Although this experimental evaluation involves two different targets including different hardwares and operating systems, it aims to put some light about the adequation of each target to the same unmodified image and signal processing applications. This adequation is merely in term of scalability and performances per Watt.

Figure 7 shows the speedup results on the ALMOS/TSAR target. The results show a very good quasi-linear scalability for the four application up to 64 cores. The EPFilter application (EP1024-A) has the best speedup of 215 on 256 cores. The speedup of SPLASH2 FFT (M18-A) and Tachyon ray tracer on 256 cores are respectively 142 and 136 while the Histogram's one is of 112. These preliminary results shows a promising potential of a signal-chip, cc-NUMA embedded many-core to run unmodified shared-memory image and signal processing applications when using an optimized operating system.

Figure 9 shows the speedup of the same unmodified applications on both ALMOS/TSAR and Linux/AMD with various sizes. This comparison shows: (i) the same applications have best scalability on the target ALMOS/TSAR than the Linux/AMD; (ii) increasing the input size gives better speedup only for Histogram and Tachyon on Linux/AMD while it gives near the same speedup curve for EPFilter and Tachyon on ALMOS/TSAR; and (iii) As the applications are the same in both cases, the scalability drawback can be caused by Linux or the AMD 64 cores architecture.

Figure 8 shows the comparison between the time of execution in million of cycles for the four applications. For a small configuration involving up to 2 NUMA nodes (16 cores) and as each AMD core is a superscalar, multi-issues, the target Linux/AMD gives the best execution time. Nevertheless, ALMOS/TSAR is able to reach almost the same results as soon as the number of cores reaches 64. This shows that the embedded target, with its 32 bits simple cores and small caches gives a good performance by power envelop in comparison with the high-end Linux/AMD target with 64bits cores and big caches. An AMD processor (16 cores) measures 315 mm2 in 32 nm technology and consumes 140 W while a cluster (4 cores) of TSAR measure less than 2 mm2 in the same technology. The cumulative surface (without taking in consideration all the packaging and interconnection stuffs) of the 4 AMD processors which gives 64 cores is about 1260 mm2 while the surface of the single-chip TSAR many-core processor having 64 cores is about 32 mm2.

5. RELATED WORK

The locality of memory access is an important factor in the performance of NUMA machines. This factor depends on the strategies for resource allocation, processors and memory, and is known [25] since the first machines NUMA, CM* [29]

Rely on the operating system and virtual memory management allows to implements solutions around migration and replication of physical pages [27] [10]. The approach requires information provided by the hardware such as cache miss and TLB miss.

Use a method of allocating physical pages to the application or "first-touch" at the core and make use from user mode while relying on hardware trace information has been explored by [20].

Instead of trying to establish the location in a dynamic way, other studies show the importance of taking into account the topology of the architecture and exploiting it during the allocation of memory and processors [13].

The effectiveness of Next-Touch strategy has been stud-

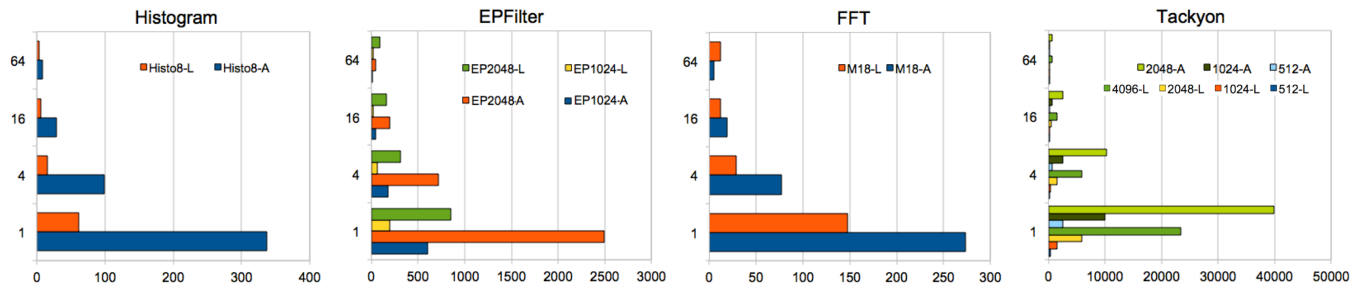

Fig. 8. Execution time in million of cycles on the two targets (x-axis: million of cycles; y-axis: number of cores)

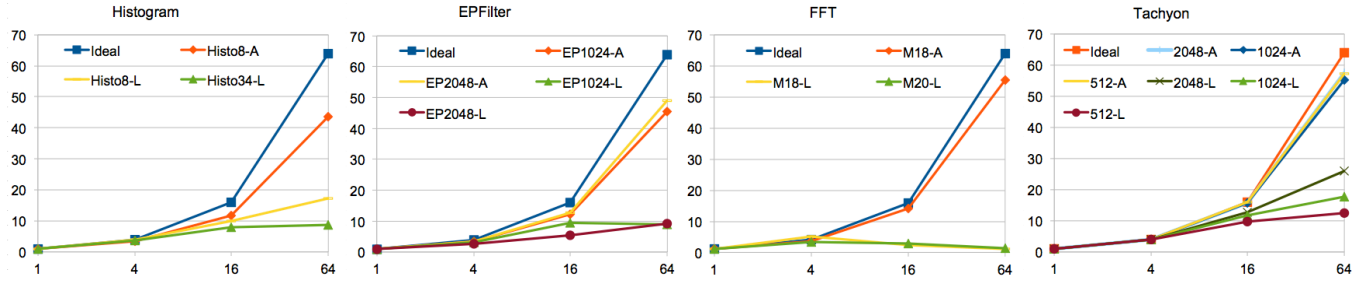

Fig. 9. Speedup on ALMOS/TSAR and Linux/AMD with several input sizes (x-axis: number of cores; y-axis: speedup)

ied [19] and its implementation in Linux has been suggested [15]. We proposed our kernel level solution, Auto-Next-Touch, to overcome the limitations of this strategy and making it automatic and transparent to user applications.

Research in operating systems has not stopped evolving since the early NUMA machines. Several recent research projects exist such as: the K42 [5] system based on micro-kernel, Barrelfish [7] which introduces the concept of multi-kernel or Corey [12] that allows the user to control the placement of some kernel data structures. Based on a micro-kernel approach FOS [31] argues for processors spatial partitioning between the system and user applications rather than share them temporally (timeslicing). The monolithic distributed design approach, used in the kernel of ALMOS, is different from the multi-kernel, the micro-kernel or the exo-kernel. Instead ALMOS has a monolithic kernel in shared memory. ALMOS shares some of K42[5] design goals regarding the importance of locality for the scalability and the organization in distributed objects. Both of ALMOS and Linux operating system are shared-memory, Unix-like, POSIX-compatible operating systems. Our contributions in ALMOS kernel can be reused in other shared-memory operating systems.

At the best of our knowledge, we are the first to experimentally evaluate the scalability of the four selected, shared-memory, PThreads-based, image and signal processing on single-chip, cc-NUMA, 256 cores based many-core and compare the results on the recent AMD Opteron 64-cores many-core.

6. CONCLUSION AND FUTURE WORK

In this paper, we have presented an experimental evaluation of the scalability of four unmodified shared-memory, PThreads-based, image and signal processing applications on both embedded and high-end many-cores. Our results show a good scalability of these applications on ALMOS/TSAR up to 256 cores. These preliminary results indicate the potential of general-purpose single chip cc-NUMA many-cores to run image and signal processing oriented applications with ease-of-use from the programmer point of view (the applications were not modified). From the other hand, our evaluations of the same applications on the Linux/AMD target indicate a scalability drawback due to Linux, the AMD Opteron architecture or both of them regarding these applications. By our two kernel-level contributions, we emphasis the importance of the operating system design to efficiency use the parallelism offered by an emerging single-chip, cc-NUMA many-core. Our evaluation of the distributed client/server scheduler design showed its relevance against the a lock-based scheduler design. Our Auto-Next-Touch mechanism is promising. We are currently working on a more generalized dynamic memory affinity solution and looking for results on TSAR configured to 512 cores.

7. REFERENCES

[1] Almos (advanced locality management operating system). www.almos.fr.

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

[3] V. Agarwal 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, ISCA '00, pages 248–259, New York, USA, Jun 2000. ACM.

[4] G. Almaless and F. Wajsburt. Does shared-memory, highly multi-threaded, single-application scale on many-cores? In *Proceedings of the 4th USENIX Workshop on Hot Topics in Parallelism*, Berkeley, California, USA, Jun 2012.

[5] J. Appavoo and al. Experience distributing objects in an smmp os. In *ACM Transactions on Computer Systems, 25(3)*, 2007.

[6] D. H. Bailey. Ffts in external or hierarchical memory. *J. Supercomput.*, 4:23–35, March 1990.

[7] A. Baumann 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] S. Bell 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*, pages 88 –598, feb. 2008.

[9] M. Berezecki, E. Frachtenberg, M. Paleczny, and K. Steele. Many-core key-value store. *International Green Computing Conference and Workshops*, 0:1–8, 2011.

[10] W. J. Bolosky and al. Numa policies and their relation to memory architecture. In *Proceedings of the fourth international conference on Architectural Support for Programming Languages and Operating Systems*, pages 212–221, Santa Clara, California, USA, Apr 1991.

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

[12] S. Boyd-Wickizer and al. Corey: An operating system for many cores. In *In Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation*, 2008.

[13] T. Brecht. On the importance of parallel application placement in numa multiprocessors. In *USENIX Systems on USENIX Experiences with Distributed and Multiprocessor Systems,*, pages 1–1, San Diego, USA, Sep 1993.

[14] R. Buchmann and A. Greiner. A fully static scheduling approach for fast cycle accurate systemc simulation of mpsocs. In *Microelectronics, 2007. ICM 2007. Internatonal Conference on*, pages 101–104, dec. 2007.

[15] B. Goglin and N. Furmento. Enabling high-performance memory migration for multithreaded applications on linux. In *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, pages 1–9, May 2009.

[16] F. Guim, I. Rodero, J. Corbalan, and M. Parashar. Enabling gpu and many-core systems in heterogeneous hpc environments using memory considerations. In *Proceedings of the 2010 IEEE 12th International Conference on High Performance Computing and Communications*, HPCC '10, pages 146–155, Washington, DC, USA, 2010. IEEE Computer Society.

[17] Kalray. Mppa: Multi-purpose processor array. http://www.kalray.eu/en/products/mppa.html.

[18] R. Kessler. Cavium 32 core octeon ii cn68xx. *Hot Chip 23*, Aug. 2011.

[19] H. Löf and S. Holmgren. Affinity-on-next-touch: increasing the performance of an industrial pde solver on a cc-numa system. In *Proceedings of the 19th annual international conference on Supercomputing,*, Cambridge, Massachusetts, USA, Apr 2005.

[20] J. Marathe and F. Mueller. Hardware profile-guided automatic page placement for ccnuma systems. In *Proceedings of the eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, New York, NY, USA, Mar 2006.

[21] I. Miro-Panades, A. Greiner, and A. Sheibanyrad. A low cost network-on-chip with guaranteed service well suited to the gals approach. In *IEEE 1st Internationnal Conference on Nano-Networks*, 2006.

[22] P. Paulin. Programming challenges & solutions for multiprocessor socs: an industrial perspective. In *Proceedings of the 48th Design Automation Conference*, DAC '11, pages 262–267, New York, NY, USA, 2011. ACM.

[23] Picochip. Pc205. http://www.picochip.com/page/76/Multicore-PC205.

[24] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. In *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, HPCA '07, pages 13–24, Washington, DC, USA, 2007. IEEE Computer Society.

[25] K. Schwan and A. K. Jones. Specifying resource allocation for the cm* multiprocessor. In *IEEE Software*, pages 3(3): 60–70, May 1984.

[26] J. Shalf, K. Asanovic, D. Patterson, K. Keutzer, T. Mattson, and K. Yelick. The manycore revolution: Will hpc lead or follow? In *SciDAC Review*, No. 14, pages 40–49. IOP in association with ANL, US DoE and Office of Science, 2009.

[27] V. Soundararajan and al. Flexible use of memory for replication/migration in cache-coherent dsm multiprocessors. In *Proceedings of the 25th annual International Symposium on Computer Architecture*, pages 342–355, Barcelona, Spain, Jun 1998.

[28] J. Stone. An efficient library for parallel ray tracing and animation. Technical report, In Intel Supercomputer Users Group Proceedings, 1995.

[29] R. J. Swan, S. H. Fuller, and D. P. Siewiorek. cm* a modular, multi-microprocessor. In *Proceedings of the national computer conference*, Dallas, USA, Jun 1977.

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

[31] D. Wentzla and A. Agarwal. Factored operating systems (fos): The case for a scalable operating system for multicores. In *Operating Systems Review, 43(2),*, 2009.

[32] W. Wolf, A. Jerraya, and G. Martin. Multiprocessor system-on-chip (mpsoc) technology. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(10):1701 –1713, oct. 2008.