Efficiency modeling and exploration of 64-bit ARM compute nodes for exascale

Joël Wanza Weloli, Sébastien Bilavarn, Martin de Vries, Said Derradji, Cécile Belleudy

Efficiency Modeling and Exploration of 64-bit ARM Compute Nodes for Exascale

J. WANZA WELOLI[1], S. BILAVARN[2], M. DE VRIES[1], S. DERRADJI[1], AND C. BELLEUDY[2]

[1] *Bull atos technologies, Les Clayes Sous Bois, France*
[2] *LEAT, CNRS UMR7248, University of Nice Sophia-Antipolis, France*

This paper investigates the use of 64-bit ARM cores to improve the processing efficiency of upcoming HPC systems. It describes a set of available tools, models and platforms, and their combination in an efficient methodology for the design space exploration of large manycore computing clusters. Experimentations and results using representative benchmarks allow to set an exploration approach to evaluate essential design options at micro-architectural level while scaling with a large number of cores. We then apply this methodology to examine the validity of SoC partitioning as an alternative to using large SoC designs based on coherent multi-SoC models and the proposed SoC Coherent Interconnect (SCI).

INTRODUCTION AND CONTEXT

The performance of supercomputers has traditionally grown continuously with the advances of Moore's law and parallel processing, while energy efficiency could be considered as a secondary problem. But it quickly became clear that power consumption was the dominant term in the scaling challenges to reach the next level. It is roughly considered that 20 times energy efficiency improvement is required for exascale computing (10^{18} FLOPS) to cope with the tremendous electrical power and cost incurred by such computational capacity. The idea of using concepts borrowed from embedded technologies has naturally emerged to address this. First prototypes based on large numbers of low power manycore microprocessors (possibly millions of cores) instead of fast complex cores started to be investigated, putting forward a number of proposals for improvement at node level architecture to meet HPC demands.

These works covered a variety of 32-bit RISC cores ranging from ARM Cortex-A8 [1] and Cortex-A9 [2][3][4] to more recent Cortex-A15 and Cortex-A7 cores [5]. [2][3] and [4] addressed, for example, dual and quad core systems based on ARM Cortex-A9 cores. The different results indicated various processing limitations to meet HPC performance requirements, in terms of double precision floating point arithmetic, 32-bit memory controllers (limiting the address space), ECC memory (e.g. for scientific and financial computing), and fast interconnect (communication intensive applications). [6] and [7] additionally confirmed that the variability in performance and energy could largely be attributed to floating point and SIMD computations, and interactions with the memory subsystem. Other works, which addressed explicit comparison against x86 based systems, also pointed out the need for higher levels of performance to meet HPC demands. [4] concludes that the cost advantage of ARM clusters diminishes progressively for computation-intensive applications (i.e. dynamic Web server application, video transcoding), and other works like [8] conducted on ARM Cortex-A8, Cortex-A9, Intel Sandybridge, and Intel Atom confirmed that ARM and x86 could achieve similar energy efficiency, depending on the suitability of a workload to the microarchitectural features at core level.

Of the works addressing the feasibility of ARM SoCs based HPC systems, efforts focused widely on single-node performance using microbenchmarks. Less studies considered large-scale systems exceeding a few cores even though multi-node cluster performance is an essential aspect of future Exascale systems [11]. Considering

further that new generations of cores such as the *ARMv8-A* ISA support features to improve specifically on HPC workloads (64-bit address space, 64-bit arithmetic, high speed interconnects, fast memory hierarchies), this work is one of the first to describe outcomes of research opened up with these perspectives. Therefore we provide an evaluation of available tools, models and platforms able to set the foundations of a methodical system level exploration approach for HPC applications scaling up to 128 64-bit ARM cores and show how it was used to examine the relevance of SoC partitioning to limit complexity, cost and power consumption.

This work is carried out under a long-term European effort called Mont-Blanc. Mont-Blanc is one of the many H2020+ projects funded by the European Commission to support Exascale research. Started in october 2011, phase 1 (Mont-Blanc 1) investigated the adoption of embedded mobile processors in a HPC system [9] and led to the establishment of a prototype based on Exynos 5 compute cards (ARM 32-bit cores) [10]. The goal of the subsequent project Mont-Blanc 2 was to develop a full software ecosystem along with architecture exploration in a joint co-design (hardware / software) methodology. Finally the last phase and ongoing Mont Blanc 3 project (started in october 2015) aims to exploit the strong knowledge developped to produce the future high-end HPC platform able to realize the level of performance and energy ratio required for exascale class applications. The work presented in this paper is in the background of Mont-Blanc 2 and Mont-Blanc 3 projects. It addrresses advanced HPC compute nodes upon the *ARMv8-A* ISA with special attention on the overall on-chip memory consistency, scalability, cost and energy efficiency. This program prefigures features of the upcoming Bull sequana platform based on Cavium ThunderX2 ARMv8-A processors.

The outline of the paper is the following. We present in section 2 different modeling and simulation tools suited to the analysis of HPC systems with lately available ARM 64-bit based platforms (ARM Juno, AMD Seattle, AppliedMicro X-Gene). Using the defined methodology, section 3 explores in detail a set of architectural propositions aiming at reducing SoC and cache coherence complexities and examines their impacts from application parallelism perspective in different programming models. Section 4 summarizes the main conclusions from the various results and exposes next directions of research.

METHODOLOGY

In the following, we investigate the use of available tools, models and platforms to define an exploration approach matching our needs. We then characterize a set of relevant HPC benchmarks on different platform configurations to verify that we meet all conditions for exploration effectiveness given a set of architectural requirements to consider (performance, memory architecture, interconnect, scalability).

Modeling

ARMv8-A platforms

The Juno ARM Development Platform is one of the first available development platforms for *ARMv8-A*. Our interest goes mainly for the performance cluster (dual Cortex-A57 of the MPCore big.LITTLE processor), the Cache Coherent Interconnect (CCI-400) and the DDR3-1600 dual channel memory controller. In this characterisation effort of real hardware, we also addressed the use of an AMD Seattle board based on four clusters of two Cortex-A57 cores with AMD Coherent Interconnect at 2 GHz and two DDR4-3733 memory controllers. We additionally considered an AppliedMicro (APM) X-Gene1 in which the SoC includes four clusters of the two 64-bit cores running at 2.4 GHz, APM coherent network Interconnect and DDR3 controller (16 GB).

We therefore employ virtual platforms to possibly extend exploration perspectives to the support of large scalability (up to 128 cores) and use of upcoming 64-bit cores (e.g. Cortex-A72). ARM Fast Model virtual platforms (AFM) is the largest platform that can be configured in this regard, using only ARM available fast model IPs up to 48 Cortex-A57/A72. The limitation to 48 cores in current releases comes from the Cache Coherent Network CCN-512 interconnect supporting a maximum of 12 coherent clusters while each cluster can contain up to four cores. We finally exploit the Virtual Processing Unit (VPU) and Task Modeling framework which is part of a Synopsys methodology for large scale SoC, coherent interconnect and memory sub-system exploration. This framework provides further abstraction of multicore SoC platforms in SystemC/TLM with

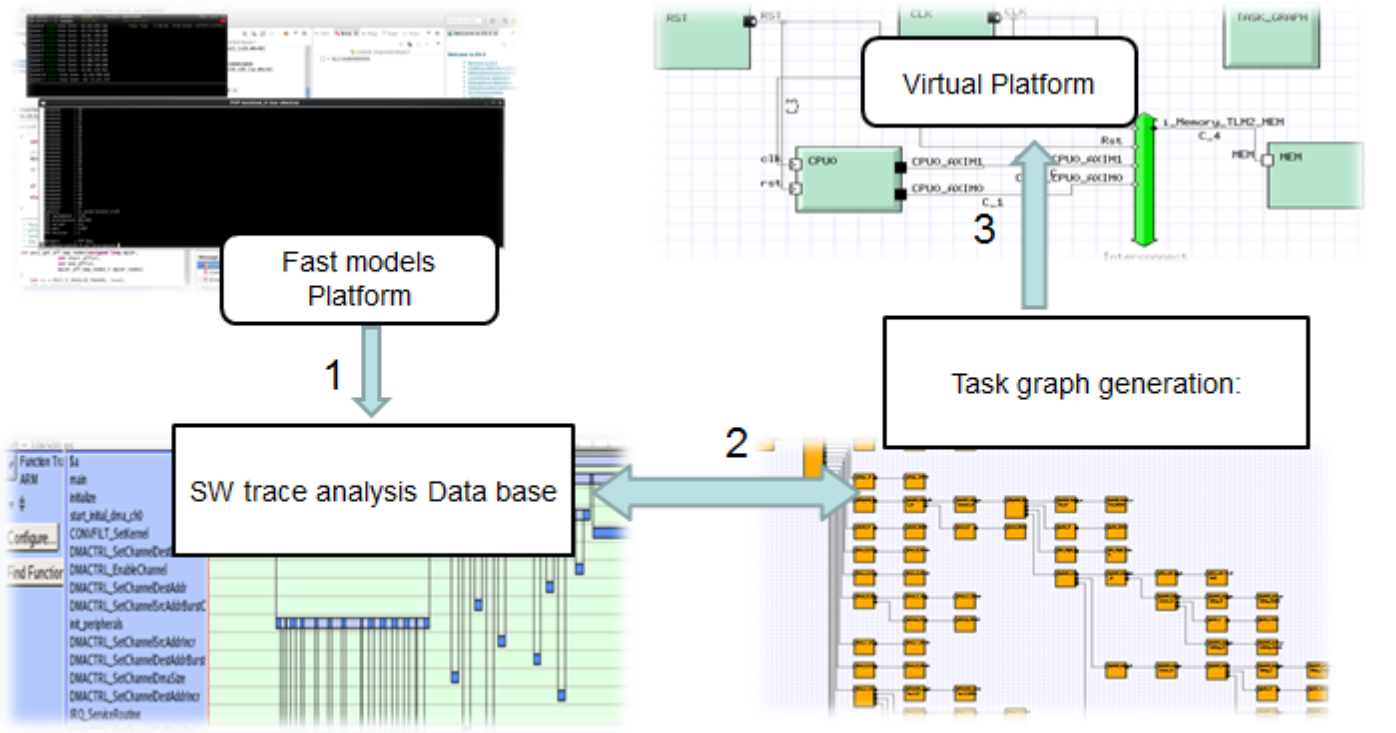


Fig. 1. Task graph based methodology.

an interactive Traffic Generation and Cycle-Accurate TLM Interconnect Models based on software traces of previous ARM Fast Models which can be used to address higher levels of scalability.

Simulation tools

AFM and VPU platforms are considered with Synopsys Platform Architect for performance and power analysis. Figure 1 provides an overview of the simulation methodology which is based first on a combination of real and virtual platform executions when the system has only a few nodes. Task graphs can then be produced from the results and used to further simulate up to 128 nodes using the VPU platform. In addition to benefiting from both virtual platforms in the same design environment, this framework provides a review on existing ARM models and a mean to improve architectural analysis with either type of complementary platforms AFM and VPU. In the following benchmarking study (section B), VPU and AFM platforms are configured with features corresponding to the aforementioned real boards in order to verify the correlation in terms of performance (GFLOPS, Cycle Per Instruction, etc.) and memory hierarchy (cache statistics, memory bandwidth).

Table 1. STREAM kernels.

Functions	Operations
Copy	$a(i) = b(i)$
Scale	$a(i) = q * b(i)$
Sum	$a(i) = b(i) + c(i)$
Triad	$a(i) = b(i) + q * c(i)$
Mean	$(Copy + Scale + Sum + Triad)/4$

Applications

Floating point benchmarking is based on SGEMM (Single-precision General Matrix multiply), DGEMM (Double-precision General Matrix multiply [12]) and HPL (High Performance Linpack from Top500 [13]).

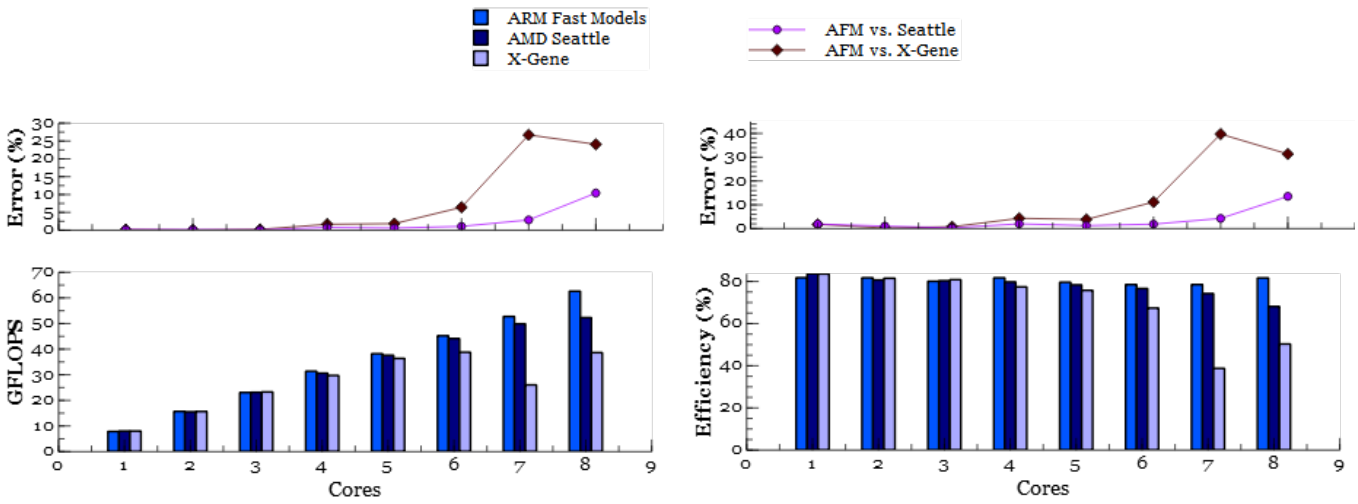


Fig. 2. Performance and efficiency of ARM Fast models vs. AMD Seattle and X-Gene platforms on SGEMM benchmark.

SGEMM and DGEMM measure the floating point rate of execution of respectively single precision and double precision real matrix-matrix multiplication while HPL measures the floating point rate of execution for solving a linear system of equations. These benchmarks are commonly used in practice to help characterize system performances in terms of floating point operations for HPC systems. In addition, we use the STREAM benchmark [14] to address more memory related aspects (cache stimulation, memory bandwidth) with four types of different kernels reported in table 1.

Benchmarking

We analyze the relevance of models and tools against real platforms (up to eight cores), firstly in terms of floating point processing performance and efficiency. We then focus on the memory and cache architecture, analyze the conditions of validity of the results, and extend the methodology to support robust analysis for a larger number of cores (possibly up to 128).

Performance models

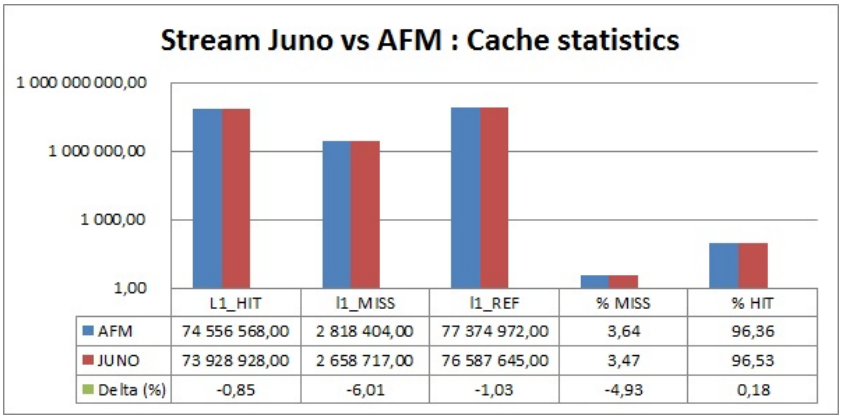
We consider two metrics to evaluate the processing efficiency. The first one is based on floating point operations per second (GFLOPS) which is reflective of the processing power for HPC workloads, and the second is the FLOPS efficiency expressing the ratio of actual versus theoretical FLOPS supported by the system. ARM Fast models are used as one objective is to examine the organization of efficient clusters, which can well benefit here from an ARM CoreLink CCN-512 interconnect model supporting up to twelve clusters of four A57 cores. Two real platforms (AMD Seattle, AppliedMicro X-Gene) supporting both four clusters of two *ARMv8-A* 64-bit cores with their built-in interconnect are thus used to compare the models with reality as reported in figure 2. These two platforms are thus modeled with the afore-mentioned Synopsys virtual platform using A57 AFM models for all cores and the CCN-512 interconnect model in the absence of interconnect models for the AMD and AppliedMicro platforms.

In the following performance measurements, each simulation/execution was reproduced ten times to ensure that variations were negligible. The results indicate an average GFLOPS and efficiency accuracy of respectively 1.1% and 2.5% up to six cores. Then, the disparity of interconnects on the different platforms reflects in deviations that are higly sensitive with the growing number of cores. The results show therefore that the global accuracy of AFM based virtual platforms is very good with less than 1.8% in average using ARM Fast models, but greatly dependent on the relevance of the interconnect model in configurations exceeding

six cores. However, simulation times further limit the use of this platform in complex configurations (twenty two cores requires two days on a desktop workstation). This model can therefore be useful to explore core level, interconnect and intra-cluster configurations. Further scalability will thus be addressed in another way as depicted in section B.3 and the memory hierarchy is addressed in the following section.

Memory and cache architecture

The goal here is to extend previous approach to allow the robust analysis of memory hierarchy performance (execution time and throughput) and performance scalability (considering the possible impacts of cache). Given previous outcome, these metrics and more especially the cache statistics only relates to the cluster



	L1_HIT	I1_MISS	I1_REF	% MISS	% HIT
AFM	74 556 568,00	2 818 404,00	77 374 972,00	3,64	96,36
JUNO	73 928 928,00	2 658 717,00	76 587 645,00	3,47	96,53
Delta (%)	-0,85	-6,01	-1,03	-4,93	0,18

Fig. 3. L1 cache statistics

level (L1 and L2 cache) and more importantly to the L1 cache which has the most performance impact and should typically have a hit rate above 95% in real world applications. The Juno ARM platform gives access to advanced performance monitoring features of A57 and A53 cores. We can therefore configure and experiment with AFM platforms based on A57 cores to examine the precision of memory models against the Juno platform. The following thus describes simulations of a Cortex-A57 core running the STREAM benchmark where the results (figure 3) are plotted against the Juno A57 core in terms of cache statistics (L1 miss, L1 hit) with an average precision of 2.6%.

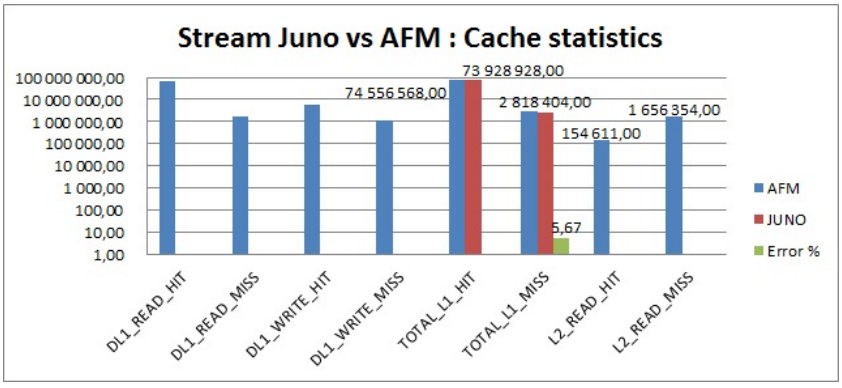
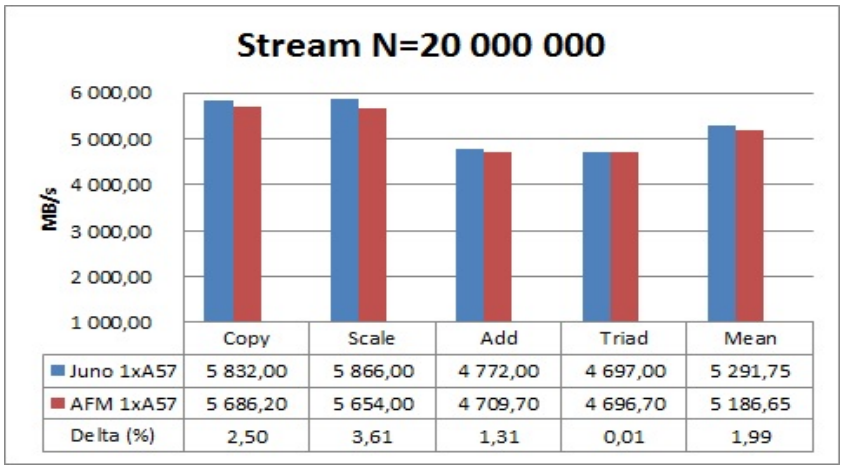


Fig. 4. L1 vs. L2 cache statistics

Figure 4 confirms that the L1 cache statistics are more meaningful than L2, which is logical because L2 cache traffic comes essentially from L1 misses of the A57 core used here. We can also observe that the cache miss correlation error doesn't exceed 6%.

It is not possible to compare AFM platforms against real numbers in configurations exceeding two cores since only two A57 are available on the Juno platform. Anyway as pointed out previously, growing deviation is likely to occur due to the difference between interconnects. However, on the 1xA57 reference configuration used to

run the five STREAM kernels, AFM provides pinpoint accuracy with 1, 9% on memory bandwidth estimation on average (figure 5). Therefore, this setup can be profitably used to identify improvement opportunities at node/cluster level concerning the effect of different cache configurations (size, policy, topology) on system performances. STREAM parameter N is the length of the arrays used in the source code test to cover a large enough portion of the memory. There are three double precision (64 bits = 8 bytes) vectors A, B and C in the test, so the size required in the memory is 3*8*N bytes (457 MB when N=20 000 000).



	Copy	Scale	Add	Triad	Mean
Juno 1xA57	5 832,00	5 866,00	4 772,00	4 697,00	5 291,75
AFM 1xA57	5 686,20	5 654,00	4 709,70	4 696,70	5 186,65
Delta (%)	2,50	3,61	1,31	0,01	1,99

Fig. 5. Memory bandwidth.

Large scale simulation

The scope of this part is to extend the analysis at a larger scale with available ARM fast models IPs. Due to CCN-512 limitations, we target configurations up to 48 cores in the following simulations (figure 6 and 7). Figure 6 reports performance metrics in terms of execution time, number of floating point operations per second and efficiency while increasing the number of threads. The efficiency is defined as the ratio between the measured and the theoretical maximum performance. X-axis represents the number of threads and the values on the y-axis relies on a common scale for performance (seconds and GFLOPS) and efficiency (%).

Inspecting the time and GFLOPS traces, system performance increases until the 22nd thread and then drops, indicating a peak for a 8192*8192 configuration (involving 1.5 GB of RAM). This means that beyond this peak value, increasing the number of cores is useless for this benchmark configuration. As the parallelism grows, the distribution of workload reaches a point above which there is an heterogeneity of computations caused by desynchronization between threads due to an under-utilization of some cores. This is also the reason for multiple non deterministic variations we can observe after this point. A larger SGEMM matrix size would be required to reach the peak performance at the 48th thread, but we start to exceed here the limits of AFM abstraction level leading to prohibitive simulation times (more than 2 weeks). The reduction of execution time is exponential as we process the same workload with increasing number of cores. As previously noted, there is a point where thread heterogeneity limits the efficiency of parallelization leading therefore to a performance threshold level.

Figure 7 reports performance and efficiency analysis of the HPL benchmark using larger AFM platforms configured for 8, 16, 32 and 48 threads. FLOPS efficiency increases gradually with parameter N. Optimized ATLAS libraries (Automatically Tuned Linear Algebra Software) are used in a way to reach the peak performance for 48 cores. However, larger values of N are needed to prevent the system from being under-used as visible in the results. Again this has not been further investigated because of excessive simulation times, but in spite of this, different information can be exploited. SGEMM benchmark show for instance that using more than 22 cores is not relevant for this benchmark configuration (figure 6) or that more memory would be needed to exploit up to 48 cores for a more parallel version of this application. This means that the platform is undersized in terms of memory allocated per core to keep a high computing efficiency. This approach represents therefore valuable feedback in terms of possible hardware and software co-design analysis to find a better balance of the

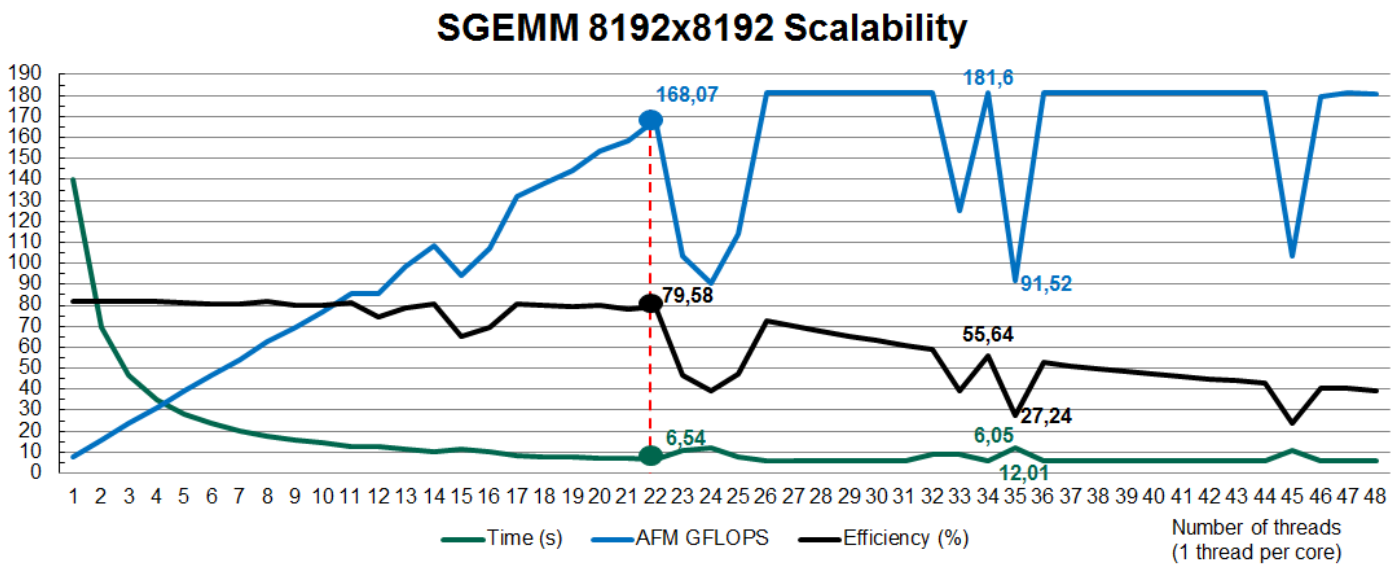
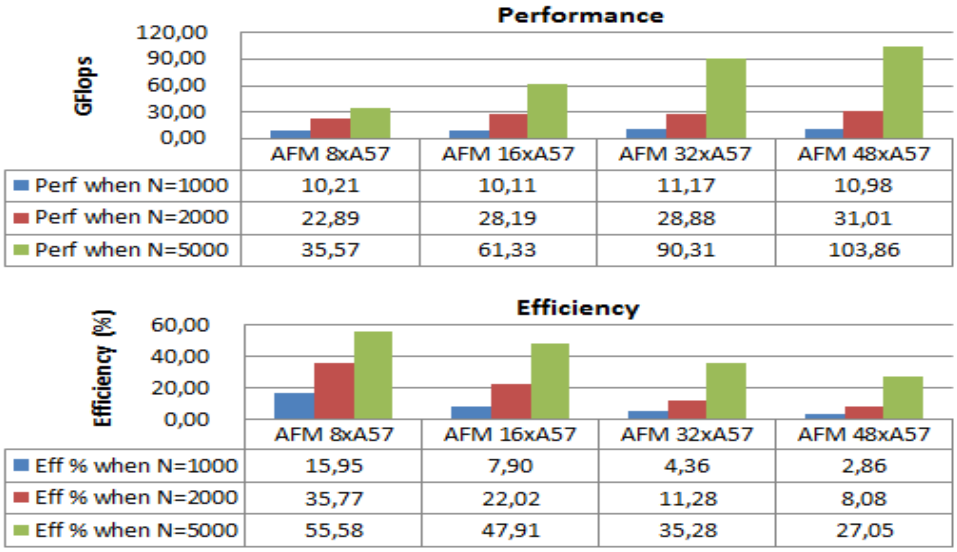


Fig. 6. Scalability on SGEMM benchmark.

system.

These results set the foundations for proper exploration and evaluation of architecture capabilities in terms of processing efficiency, memory hierarchy, interconnect, topology and scalability. Since the target platform is designed to take advantage of large *ARMv8-A* clusters, communication topology and memory system are key issues to address. In that respect, SoC partitioning becomes an attractive option to consider due to high



	AFM 8xA57	AFM 16xA57	AFM 32xA57	AFM 48xA57
Perf when N=1000	10,21	10,11	11,17	10,98
Perf when N=2000	22,89	28,19	28,88	31,01
Perf when N=5000	35,57	61,33	90,31	103,86

	AFM 8xA57	AFM 16xA57	AFM 32xA57	AFM 48xA57
Eff % when N=1000	15,95	7,90	4,36	2,86
Eff % when N=2000	35,77	22,02	11,28	8,08
Eff % when N=5000	55,58	47,91	35,28	27,05

Fig. 7. Scalability on HPL benchmark with BLAS optimized librairies.

development and production costs of large monolitic chips in the latest silicon technologies.Next architectural study extends therefore previous exploration approach (Platorm Architect, 64-bit ARM cores and a specific interconect) with necessary hardware requirements, especially regarding inter chip cache coherence support between compute nodes, in a way to study the impacts and efficiency conditions in different partitioning scenarios.

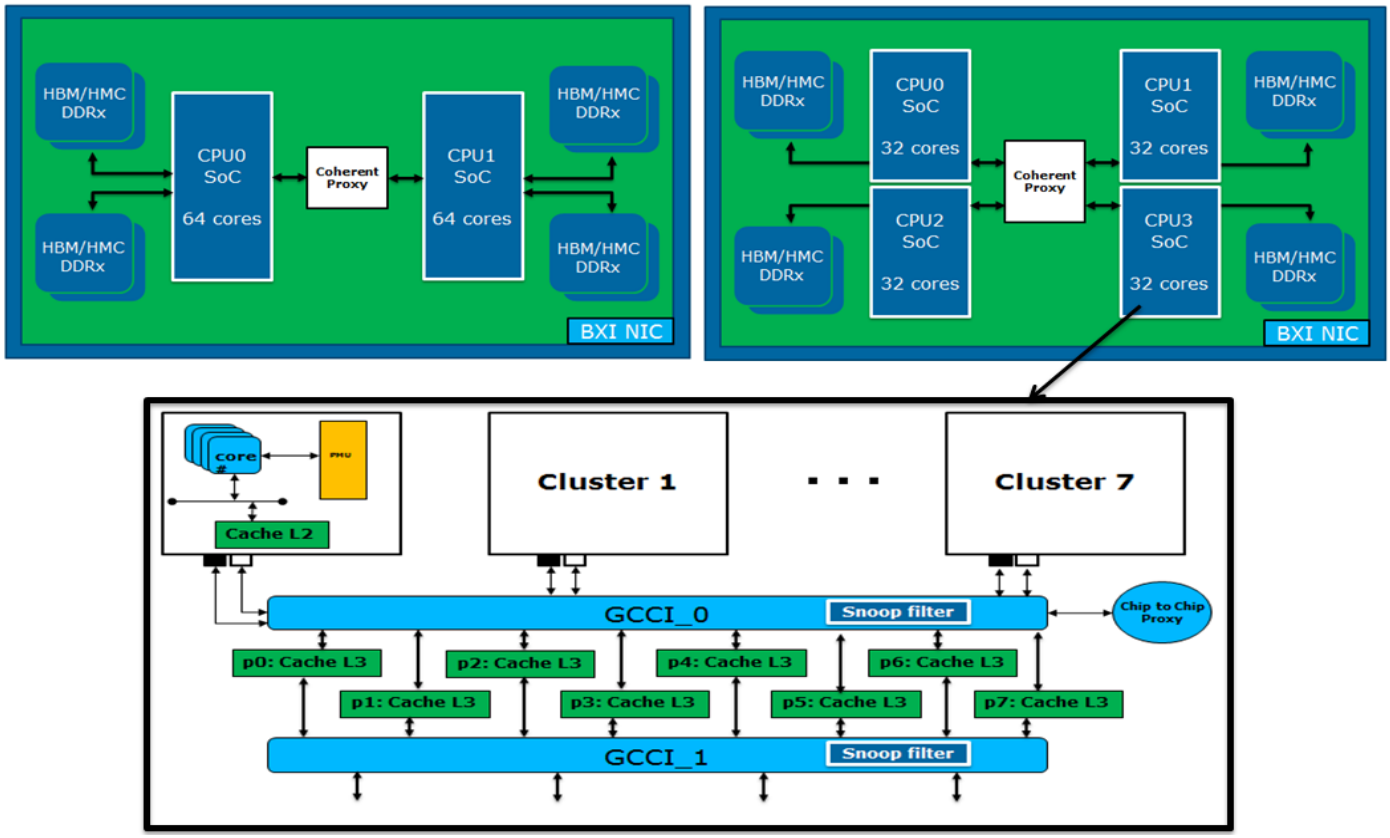


Fig. 8. SoC and cluster internal views (two SoCs / 64 cores, four SoCs / 32 cores).

ARCHITECTURAL EXPLORATION

SoC and interconnect partitioning

This exploration study addresses the validity of SoC partitioning as an alternative to using large SoC designs. On the basis of a monolithic SoC design integrating 128 *ARMv8-A* cores, high bandwidth memory (HBM/HMC) and a Bull Exascale computing network interface controller [12], the idea is to evaluate coherent multi-SoC models (i.e. two SoCs / 64 cores, four SoCs / 32 cores) communicating through chip-to-chip ports and coherent proxies (top of figure 8). Considering the global architecture model (bottom of figure 8), a bottleneck lies in the cache coherence management because existing snoop based protocols don't scale with the large number of caches that are commonly found in HPC processing. Therefore we introduce a coherence extension of the SoC interconnect (SCI) required for this partitioning. This aims at reducing the complexity of the coherence protocol and additionally can significantly reduce the energy consumption from the interconnect as well as the tag lookups in the remote caches.

Coherent interconnect

Overview of coherency protocols

A suitable definition of the coherence is given in [15] as *single-writer-multiple-reader (SWMR)* invariant. That means that for any memory location space, at a given cycle time, there may only be one single writer or a number of cores that may read it. Consequently, implementing a cache coherence mechanism requires avoiding the case where two separated caches contain two different values for the same memory address at the same moment. So a cache coherence protocol addresses the way of maintaining consistency between the multi-level system cache hierarchy and the main memory[16].

In the past, one way to maintain coherency in a multi-cache system was to use software but the required performance challenge became even greater as systems got bigger[16]. This is typically inadequate for HPC

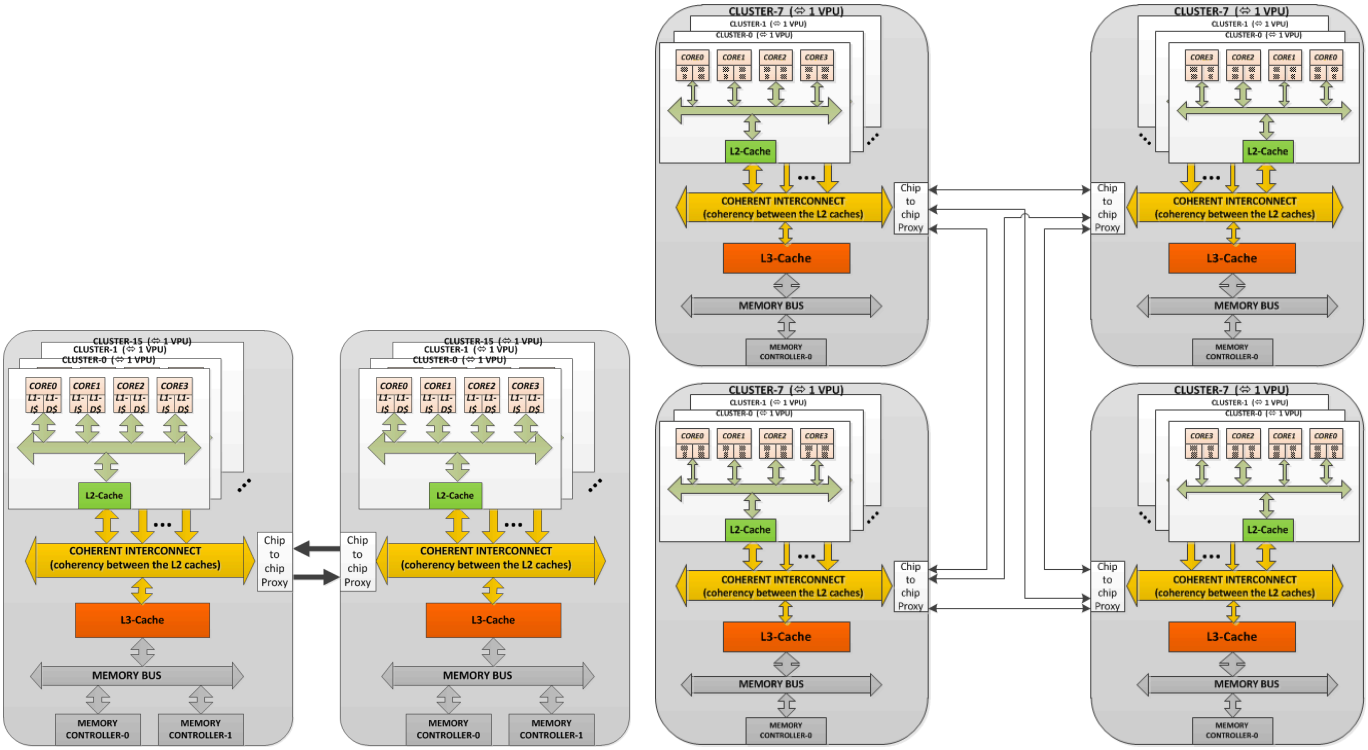


Fig. 9. 2x64 and 4x32 SoC partitioning with coherent proxy ports.

systems. Hardware-based cache coherence protocols thus appeared to be more efficient and are commonly used in many computing systems such as servers. A cache coherence protocol specification must define several key elements of the protocol background such as hardware components, coherency granularity, caches sizes, transactions types, the coherence interconnect channels or the impact of transaction on the cache line states. Therefore for the same coherence protocol, there may be several implementations depending on the type of multiprocessor or architecture.

There are many comparison studies between the two main hardware based coherency approaches: snooping cache coherency protocols and directory-based cache coherency protocols [15][17][18]. It always emerges that the first one is logically simple, ideal for ring interconnect topology but does not scale to large numbers of caches because the number of snooping broadcast transactions increases quadratic with $N * (N - 1) = N^2 - N$, where the N is the number of coherent masters [19]. The second one is perfectly scaling but with the drawback that transactions take more time because of the directory filtering complexity where the traffic scales in theory at order $N * ((N - 1) + 1) = N^2$, where $+1$ is the request for directory lookup [19]. In reality, this overhead only happens when all caches in a system have a copy of the requested data which is very rare in a large scale system (ten of caches). As it also depends on the workload parallelism, snooping traffic could thus be reduced between caches sharing a copy of the same data. The main counterpart of directory based cache coherency techniques is their implementation cost based upon on-chip SRAM storage whose size depends on the number of cache lines to manage within the system. Therefore, the potential for reducing cache coherence complexity attracts a lot of interest to improve the processing efficiency of large scale compute nodes. We explore in the following the impact of such question on a set of HPC benchmarks based on the previously defined methodology.

Coherence extension

We consider two partitioning scenarios of the SCI resulting from splitting a single-SoC 128 cores topology in two and four partitions. Figure 9 shows the two block diagrams for the corresponding cluster configurations of 1x128 and 2x64 cores. All the processors in a partitioned scenario must be able to communicate coherently as if they were connected to the same on chip coherent interconnect. All SoCs are thus connected through

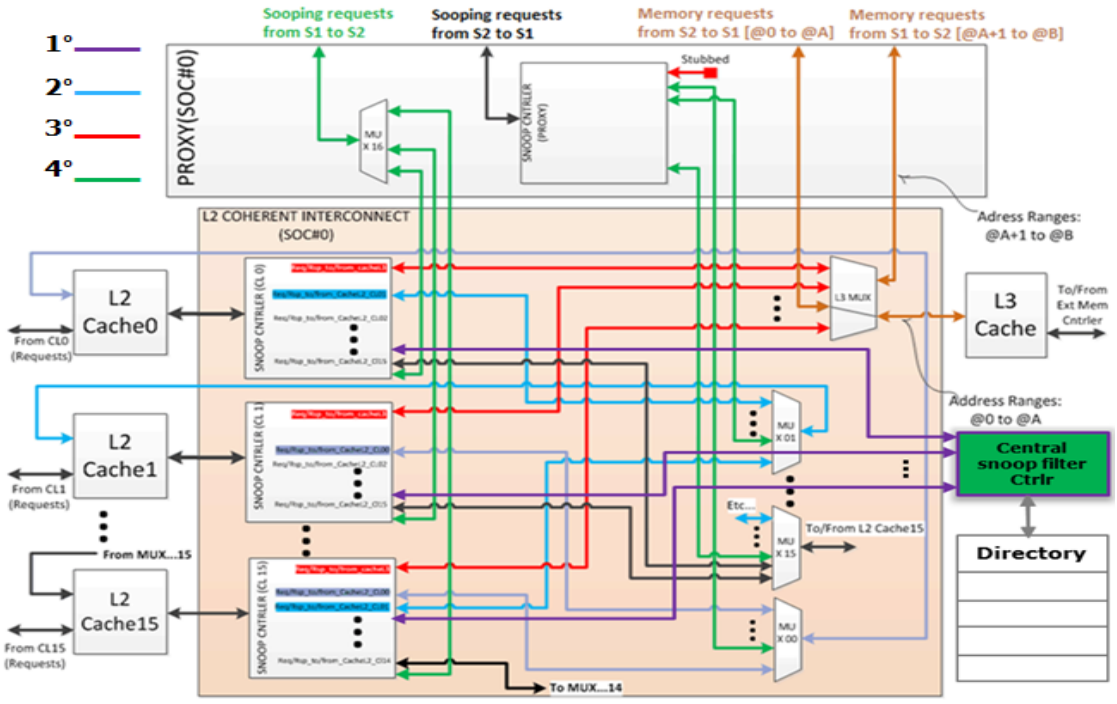


Fig. 10. Chip-to-chip coherence extension through a Proxy component (2x64 SoC partitioning).

chip-to-chip coherent proxy ports. The main role of these coherent proxy ports is precisely to enable both coherent transactions with the neighbouring sockets and accesses to external memory areas. The L3 cache is considered to be LLC (Last Level Cache) near each memory controller to save latencies for memory requests.

Figure 10 depicts exactly the SystemC/TLM2 directory based filetring model as it would be implemented in the SCI. For example, in the scenario where there is an incoming snoop request from L2 cache, the snoop controller sends a request to the directory to locate all copies of the data in the nearby peer caches (in its shareability domain). Then it queries directly the caches identified in the local socket or through the proxy extension if an external transaction is involved. In case of a snoop miss, the request is forwarded to the next cache level (L3).

The proxy component architecture is mainly similar to an empty L2 cache. It repeats incoming snooping requests from a SoC to the other(s). This leads to additional delays and asymmetric waiting response times to the snoop controller requests. The snoop controller is a module attached to each coherent interface (port) in the SCI. It is responsible for processing coherent transactions from a cluster (VPU) to the directory and the (N-1) other peer interfaces. When the SCI manages 32 coherent ports (32 clusters of four cores), each snoop controller must be able to communicate with the other 31 controllers. In the scenario of partitioning in two SoCs, it will communicate only with 16 snoop controllers (15 coherent VPU ports and the proxy one) instead of 31 snoop controllers in a large SCI (figure 10). This reduces de facto the logic design complexity and the corresponding physical area. The idea remain the same when partitionning in four SoCs, reducing connections from 31 links to 8 (7 peer ports + 1 hub proxy port) in a *one-to-all* topology or from 31 to 10 (7 peers ports + 3 direct proxy port) for an *all-to-all* topology.

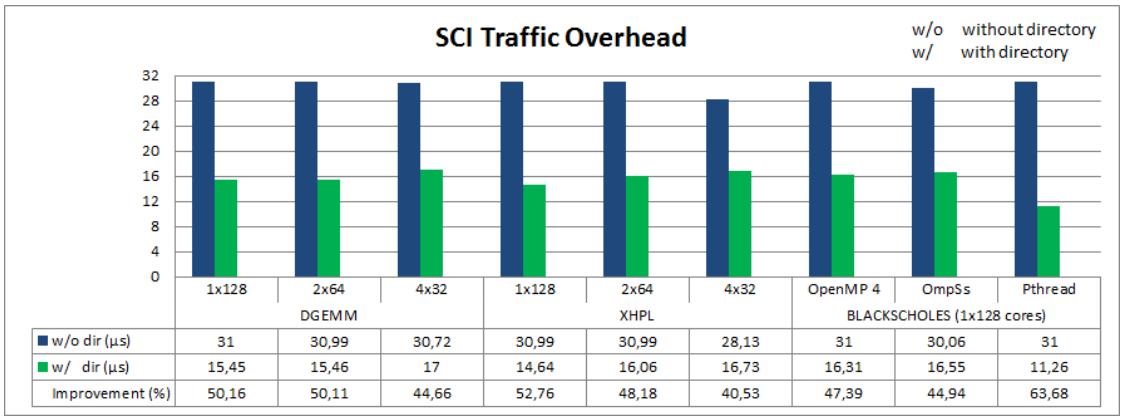
Simulations

In this assesment study, we address first the ability of using previous directory based snoop filering model in the SCI to reduce the complexity of cache coherence management and the impact on performances considering different types of benchmarks. We then analyse the relevance of two SoC partitioning configurations of two SoCs / 64 cores (2x64) and four SoCs / 32 cores (4x32) against one SoC / 128 cores (1x128), with and without the proposed snoop filtering scheme. Finally we consider more specifically the effect of different

parallel programming paradigms on the internal traffic of the coherent interconnect and the corresponding performances.

Directory-based filtering benefits

Figure 11 reports the analysis of snoop transactions generated in each of the three configurations of 32 masters (128 cores). Without directory, 31 transactions need to be generated for each cache miss (or invalidation request) to update every copy of the data. Including a directory reduces the traffic generated in the SCI between

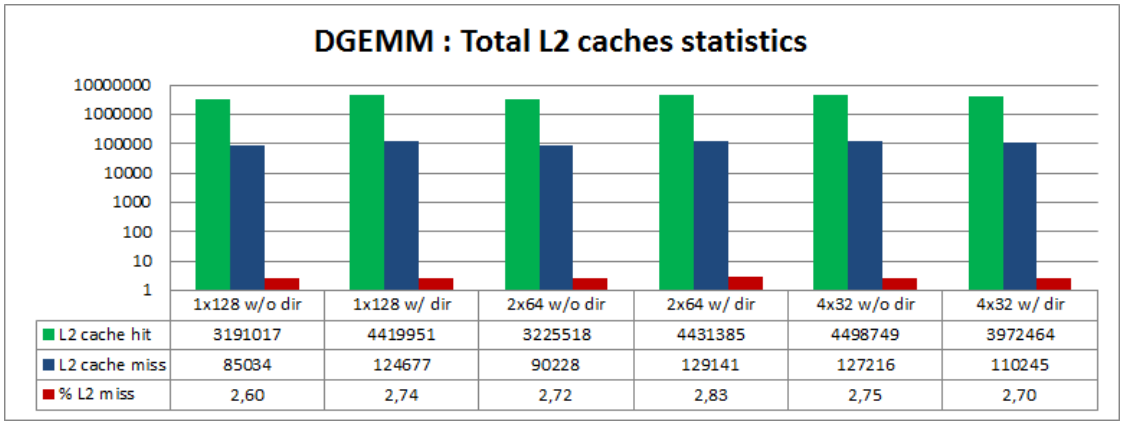


	1x128	2x64	4x32	1x128	2x64	4x32	OpenMP 4	OmpSs	Pthread
	DGEMM			XHPL			BLACKSCHOLES (1x128 cores)		
w/o dir (µs)	31	30,99	30,72	30,99	30,99	28,13	31	30,06	31
w/ dir (µs)	15,45	15,46	17	14,64	16,06	16,73	16,31	16,55	11,26
Improvement (%)	50,16	50,11	44,66	52,76	48,18	40,53	47,39	44,94	63,68

Fig. 11. Number of transactions for different SoC and directory configurations (Overhead = number of transactions generated for one incoming request).

40% and 63%, which is in line with the reduction of size of the snoop controller by a factor of two, from 31 snooping output ports to 16 (section B.2). In turn, transaction benefits improve benchmark execution times by reducing L2 cache miss penalties. Therefore, figure 13 compares execution times of the same benchmarks, with and without directory, to examine these gains. The impact of using a directory in our coherent multi-SoC architecture model is thus an average execution time improvement of 14% (4% to 26%). Despite transaction savings of about two, net performance gains are limited by the relative low number of cache misses in pratice in real applications (around 2.7% reported for DGEMM in figure 12).

However, the consistency of data shared between processors is very complex in large scale computing systems. The use of snoop filtering in the three considered SoC models ensures both data coherency and



	1x128 w/o dir	1x128 w/ dir	2x64 w/o dir	2x64 w/ dir	4x32 w/o dir	4x32 w/ dir
L2 cache hit	3191017	4419951	3225518	4431385	4498749	3972464
L2 cache miss	85034	124677	90228	129141	127216	110245
% L2 miss	2,60	2,74	2,72	2,83	2,75	2,70

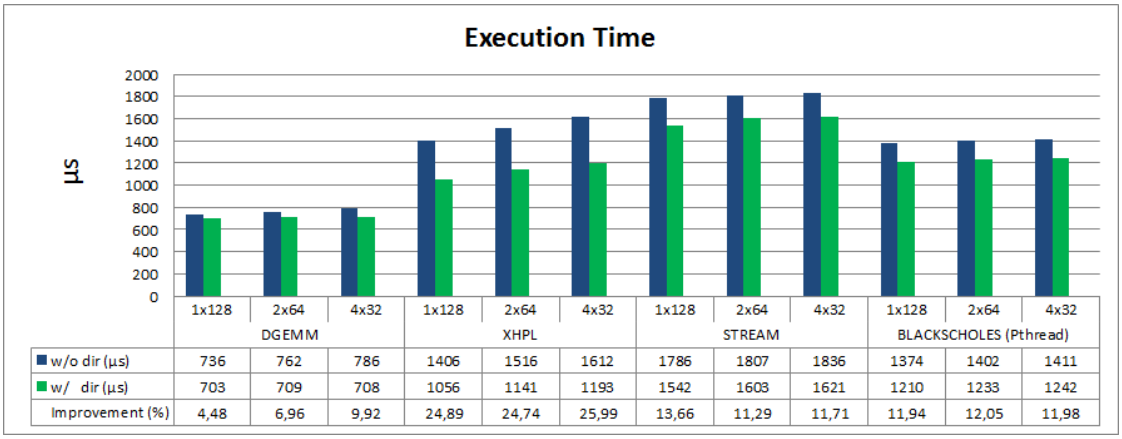
Fig. 12. DGEMM cache statistics.

processing efficiency by reducing transactions to those that are strictly necessary. The overhead for each incoming transaction is thus reduced by more than 40% which is an important matter in consumption and scalability at the memory subsytem level, while performance may be slightly improved at the same time. Memory consistency among clusters is thus ensured with significantly less coherence traffic in light of

these results. Despite little benefits in terms of net system performance, this should however bring enough improvement to promote multi-SoC partitioning at low chip-to-chip communication costs, which is the question discussed next.

Partitioning analysis

If we focus more specifically on multi-SoC configurations (2x64, 4x32 w/o dir) against single-SoC (1x128 w/o dir) in the results of figure 13, we can observe a legitimate deterioration of performances when no directory



	1x128	2x64	4x32	1x128	2x64	4x32	1x128	2x64	4x32	1x128	2x64	4x32
	DGEMM			XHPL			STREAM			BLACKSCHOLES (Pthread)		
w/o dir (µs)	736	762	786	1406	1516	1612	1786	1807	1836	1374	1402	1411
w/ dir (µs)	703	709	708	1056	1141	1193	1542	1603	1621	1210	1233	1242
Improvement (%)	4,48	6,96	9,92	24,89	24,74	25,99	13,66	11,29	11,71	11,94	12,05	11,98

Fig. 13. Benchmark performance for different SoC and directory configurations.

is used, increasing gradually with SoC partitioning due to the additional latencies coming from chip-to-chip transactions. Considering SoC partitioning in a large scale manycore system does not seem at first to be an effective approach as shown with up to 15% drop in performance for XHPL in configuration 4x32.

However, the presence of a directory improves both single-SoC and multi-SoC performances by respectively 13.7% and 14.33% (in average) since transactions are reduced to what is strictly required for cache coherence. The additional complexity introduced by the directory is widely compensated by the removal of internal/external waiting delays. This gain rise up from 4% up to 25% depending on the workload profil. The XHPL benchmark, which employs a large memory space and is very sensitive to latencies, is therefore the ideal example to show the impact of a directory in our on-chip interconnect with 25% performance improvement.

In addition, performances remain stable in the presence of a directory with an average variation of 1.9% accross all SoC configurations and benchmarks. If we compare the partitioned (2x64, 4x32 w/ dir) versus single-SoC (1x128 w/ dir) topologies, it can be verified that the impact of partitioning on execution times has been efficiently limited through the use of the directory (4.3% for 2x64 and 5.4% for 4x32). The performance level is more stable in both partitioned SoCs because there is always a copy of shared data in nearby caches resulting in no snoop misses.

These results let us therefore expect an average performance penalty of 5.2% resulting from SoC partitioning (in two and four SoCs for a 128 nodes example). But employing the defined directory-based filtering scheme is efficient enough at reducing chip-to-chip cache coherency transactions and to get rid of this overhead with even a mean improvement of 10.1% (over single-SoC without directory). The coherence extension scheme then promotes interesting opportunities such as the integration of more compute nodes directly on an interposer based System-in-Package (SiP), possibly based on 3D Through Silicon Vias (TSVs) using High Memory Bandwidth (HBM), to approach the processing power and efficiency of Exascale requirements.

Parallel programming efficiency

Another factor which may affect the value of SoC partitioning relates to software parallelism. The issue here is how to best minimize outgoing transactions between the partitioned SoCs at the programming level. We have thus considered three parallel programming models (OpenMP, OmpSs and POSIX Threads) on a *blackscholes* application (part of the PARSEC benchmark suite [20]) to investigate their influence on the efficiency of the directory.

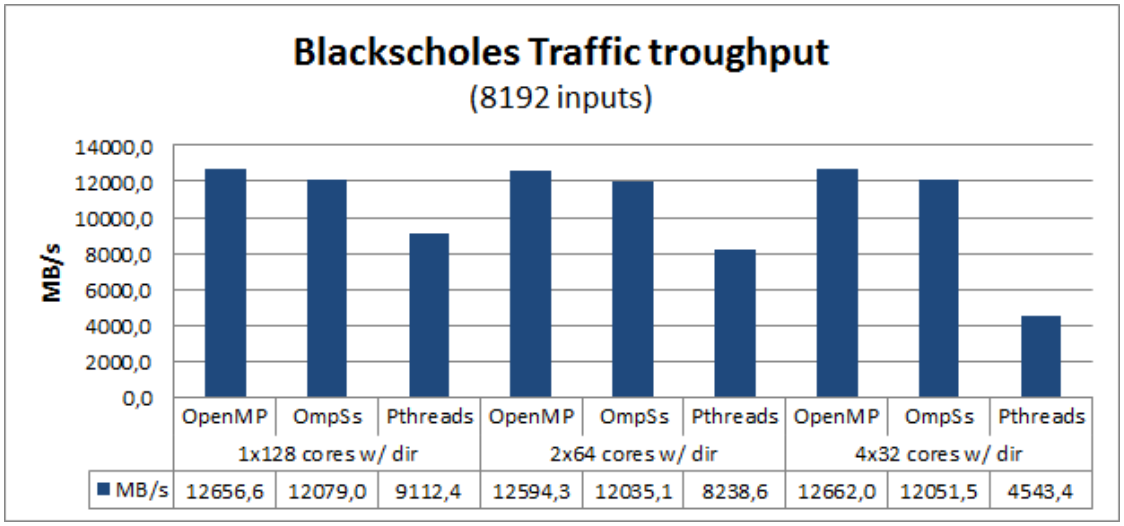


Fig. 14. Impact of programming models on throughput (blackscholes).

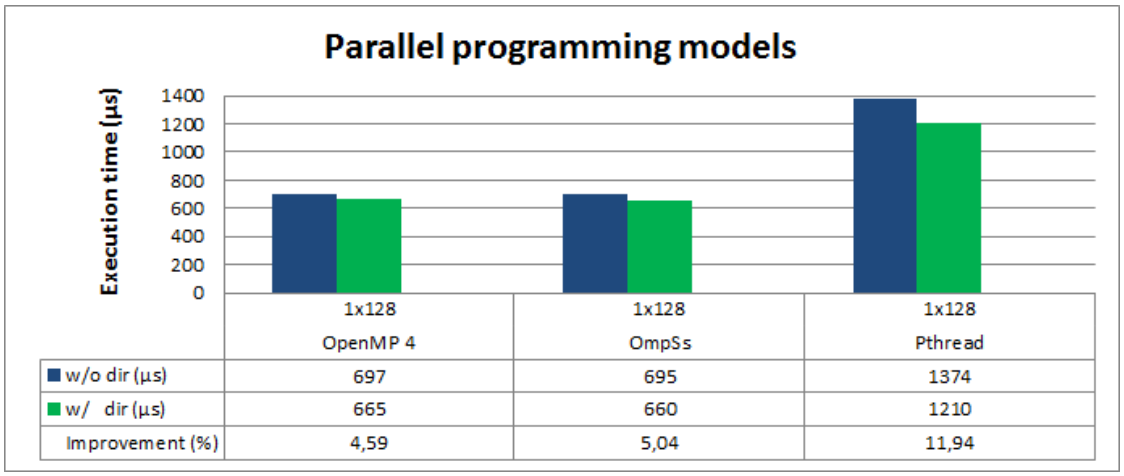


Fig. 15. Impact of programming models on performance (blackscholes, 1x128 cores).

OpenMP, OmpSs and POSIX Threads are application programming interfaces (APIs) for multi-platform shared-memory parallel programming. OpenMP provides high level threading options using code and dataflow annotations that are then used by the runtime system for execution and synchronization. POSIX threads is a lower level API for working with threads offering fine-grained threading-specific code to permit control over threading operations. Unlike OpenMP, the use of Pthreads requires explicit parallelism expression in the source code (e.g. hard-coded number of threads). OmpSs is a mix of OpenMP and StarSs, a programming model developed by the Barcelona Supercomputing Center. It provides a set of OpenMP extensions to enable asynchronous tasks, heterogeneity (accelerators) and exploit more performance out of parallel homogeneous and heterogenous architectures.

With previous results showing little influence of partitioning when using a directory, the following analysis is restricted to a single-SoC configuration. Figure 15 reports execution times of the *blackscholes* benchmark for each programming model. There are comparatively few differences between OpenMP and OmpSs results because of their similarity. The application receives little benefits (4.6% performance improvement) from OmpSs specific features to better exploit the architecture model. However, figures 14 and 15 show two clear inflection points with 82.6% less performance and 40.9% fewer application throughput using Pthreads compared to OpenMP and OmpSs. Investigating further shows that Pthreads has 28% less throughput than OpenMP (in configuration 1x128 with directory) for 79% more cache misses (figure 16). In turn, cache misses

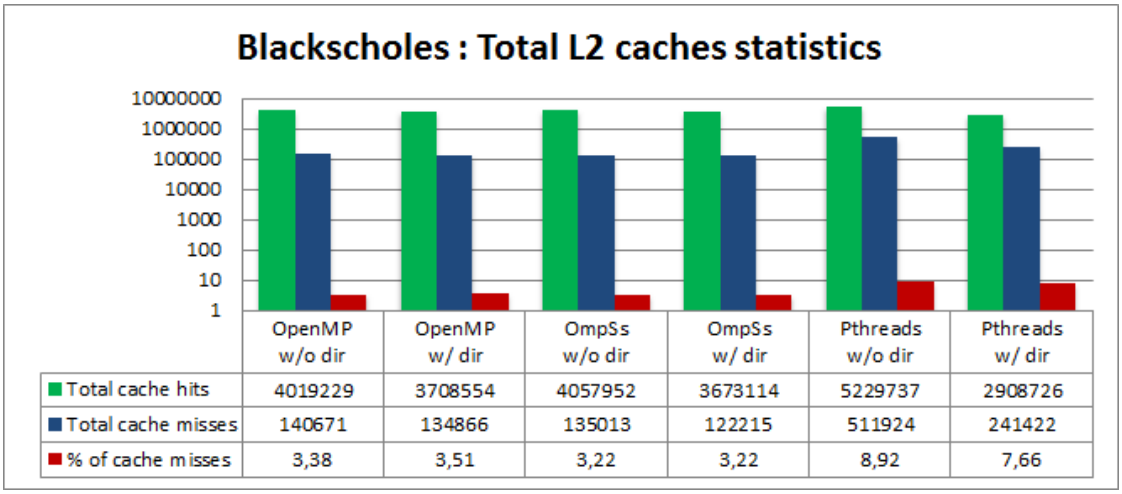


Fig. 16. Impact of programming model on L2 cache misses (blackscholes, 1x128 cores).

are responsible for generating 23.5% more traffic in the SCI compared to OpenMP (figure 17).

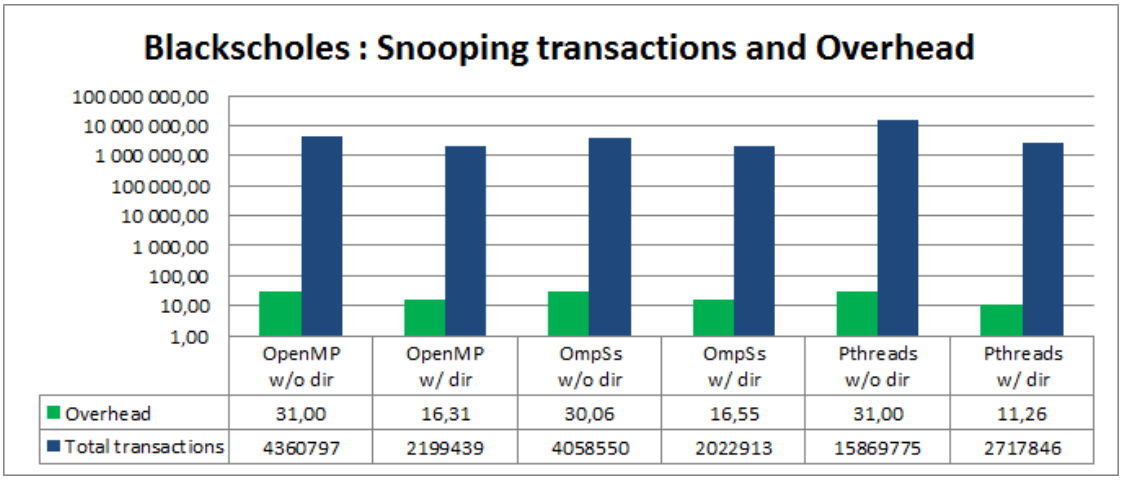
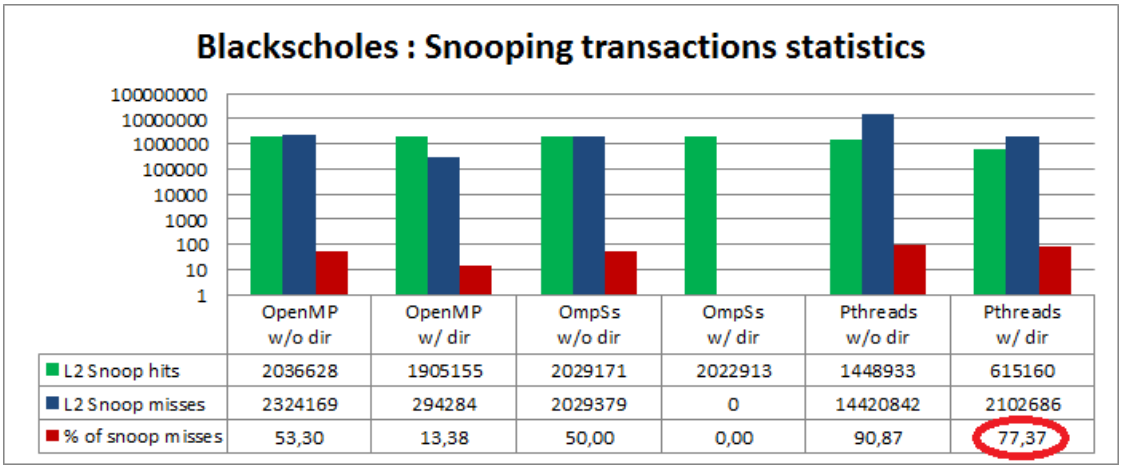


Fig. 17. Impact of programming models on the number of snooping transactions and overhead (blackscholes, 1x128 cores).

Figure 18 confirms that Pthreads has not led to an efficient use of the directory. With 77.37% snoop misses, the on-chip interconnect had to carry out the transport of five times more transactions than OpenMP. The reasons of these weaknesses lie mainly in the capability of the software model to address efficiently high degrees of parallelism. In the task-graph based parallel versions of *blackscholes* used for OpenMP and OmpSs, the work is better divided into units of a predefined block size which allows having much more task instances and better load balance than Pthreads. The effects from less reliable parallelism exploitation can therefore increase significantly (up to a factor of two) when scaling up to 128 cores.

Besides the fact that limited conclusions can be drawn on the effectiveness of a programming model which depends on how well the software was partitioned and coded, this study shows however the potential for deep analysis of appropriate parallelism exploitation by the application. These results are beneficial to help tuning the architecture and design of algorithms and software, to identify and correct programming shortcomings and further improve parallel processing efficiency.



	OpenMP w/o dir	OpenMP w/ dir	OmpSs w/o dir	OmpSs w/ dir	Pthreads w/o dir	Pthreads w/ dir
L2 Snoop hits	2036628	1905155	2029171	2022913	1448933	615160
L2 Snoop misses	2324169	294284	2029379	0	14420842	2102686
% of snoop misses	53,30	13,38	50,00	0,00	90,87	77,37

Fig. 18. Snooping traffic statistics (blackscholes, 1x128 cores).

CONCLUSION AND PERSPECTIVES

In this paper, we have examined in detail how a combined use of relevant models, tools, platforms and benchmarks could be used to define a robust design space exploration approach adapted to the tight processing efficiency constraints of upcoming HPC, especially in the new perspectives offered by 64-bit *ARMv8-A* cores. Proper architectural exploration is decomposed in two steps that allow i) reliable modeling and simulation at node/cluster level and ii) scalability analysis of a larger number of nodes using *ARMv8-A* core models. Reported experiments and results have shown the ability of the approach to reliably study central design parameters, namely in terms of FLOPS performance and efficiency, cache and memory hierarchy, and scalability support up to 128 nodes.

Using this methodology, we have explored opportunities for multi-SoC partitioning based on a directory-based coherent interconnect (SCI) defined specifically for this purpose. Exploration of partitioned (2x64, 4x32) versus single-SoC (1x128) topologies with this coherent interconnect have shown to decrease significantly the associated internal traffic (55.3%) and to limit enough the existing partitioning overhead (4.3% for 2x64 and 5.4% for 4x32) such as to permit an average 10.1% execution time saving compared to the situation where no partitioning / directory is used. Additionally, the analysis of parallel programming efficiency on a concrete example confirmed the validity of directory filtering with the ability to identify and correct software weaknesses for better parallel processing efficiency.

The perspectives from this work[1] are to build on the results achieved to combine efficiently all the elements identified for improvement (64-bit *ARMv8-A* cores, SoC and interconnect partitioning, interconnect and cache coherency complexity, parallel programming and 3D integration perspectives) such as to extend significantly the computing efficiency in large scale HPC systems.

REFERENCES

1. K. Fürlinger, C. Klausecker, and D. Kranzlmüller, Towards Energy Efficient Parallel Computing on Consumer Electronic Device, International Conference on Information and Communication on Technology for the fight against Global Warming (ICT-GLOW'11), 2011, Toulouse, France.
2. E. L. Padoin, D. A. G. de Oliveira, P. Velho, P. O. A. Navaux, B. Videau, A. Degomme, and J.-F. Mehaut, Scalability and Energy Efficiency of HPC cluster with ARM MPSoC, Workshop on Parallel and Distributed Processing (WSPPD), 2013, Porto Allegre, Brazil.

[1]The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] and Horizon 2020 under the Mont-Blanc Projects, grant agreement n° 288777, 610402 and 671697. It was also partly funded by a French ANRT CIFRE partnership between Bull (Atos technologies) and LEAT (CNRS UMR7248, University of Nice Sophia Antipolis).

3. N. Rajovic, A. Rico, J. Vipond, I. Gelado, N. Puzovik, and A. Ramirez, Experiences with mobile processors for energy efficient HPC. Design, Automation and Test in Europe Conference and Exhibition (DATE), 2013, Grenoble, France.
4. Z. Ou, B. Pang, Y. Deng, J. K. Nurminen, A. Yla-Jaaski, and P. Hui, Energy and cost-effciency analysis of ARM based clusters, Symposium on Cluster, Cloud and Grid Computing (CCGRID), 2012, Ottawa, Canada.
5. M. F. Cloutier, C. Paradis, and V. M. Weaver, Design and Analysis of a 32-bit Embedded High-Performance Cluster Optimized for Energy and Performance, Hardware-Software Co-Design for High Performance Computing (Co-HPC), 2014, New Orleans, USA.
6. M. A. Laurenzano, A. Tiwari, A. Jundt, J. Peraza, W. A. Ward Jr, R. Campbell, and L. Carrington, Characterizing the Performance-Energy Tradeoff of Small ARM Cores in HPC Computation, European Conference on Parallel Processing (Euro-Par 2014), Porto, Portugal.
7. J. Maqbool, S. Oh, and G. C. Fox, Evaluating Energy Efficient HPC Clusters for Scientific Workloads, Concurrency and Computation Practice and Experience, Volume 27, Issue 17, 2015.
8. E. R. Blem, J. Menon, and K. Sankaralingam, Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures, Symposium on High Performance Computer Architecture (HPCA 2013), Shenzhen, China.
9. N. Rajovic, P. Carpenter, I. Gelado, N. Puzovic and A. Ramirez, Are mobile processors ready for HPC?, International Supercomputing Conference (SC13), 2013, Denver, USA.
10. A. Ramirez, European scalable and power efficient HPC platform based on low-power embedded technology, European Exascale Software Initiative (EESI), 2011, Barcelona, Spain.
11. A. Bhatele, P. Jetley, H. Gahvari, L. Wesolowski, W. D. Gropp, and L. Kale, Architectural constraints to attain 1 exaflop/s for three scientific application classes, Parallel & Distributed Processing Symposium (IPDPS), 2011, Anchorage, USA.
12. C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh, Basic Linear Algebra Subprograms for FORTRAN usage, ACM Transactions on Mathematical Software (TOMS), Volume 5, Issue 3, 1979.
13. J. Dongarra, P. Luszczek, and A. Petitet, The LINPACK Benchmark: Past, Present, and Future, Concurrency and Computation Practice and Experience, Volume 15, Issue 9, 2003.
14. J. D. McCalpin, Sustainable memory bandwidth in current high performance computers, Technical report (1991 - 2007), University of Virginia, http://www.cs.virginia.edu/stream/.
15. D. J. Sorin, M. D. Hill, and D. A. Wood, A Primer on Memory Consistency and Cache Coherence, Synthesis Lectures on Computer Architecture, Morgan & Claypool, 2011.
16. R. Weber, Modeling and Verifying Cache-Coherent Protocols, International Symposium on Circuits and Systems (ISCAS), 2001, Sydney, Australia.
17. A. R. Lebeck and D. A. Wood, Dynamic self-Invalidation: Reducing Coherence Overhead in Shared-Memory Multiprocessors, International Symposium on Computer Architecture (ISCA'95), 1995, Santa Margherita Ligure, Italy.
18. A. Moshovos, G. Memik, B. Falsafi and A. Choudhary, Jetty: filtering snoops for reduced energy consumption in SMP servers, Symposium on High-Performance Computer Architecture (HPCA'01), 2001, Nuevo Leone, Mexico.
19. A. Stevens, Introduction to AMBA 4 ACE and big.LITTLE™ Processing Technology, ARM Holdings, July 2013.
20. C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications, Parallel Architectures and Compilation Techniques (PACT), 2008, Toronto, Canada.

VITAE

Joel Wanza Weloli received his M.E. degree in Electronics and Embedded Systems from University of Cote d'Azur, France in 2014. He is currently occupying a PhD student position at Bull (Atos technologies) as member of the hardware architecture team. His research interests are power and energy models and design space exploration especially for ARM based compute nodes and clusters in the field of High Performance Computing.

Sébastien Bilavarn received the B.S. and M.S. degrees from the University of Rennes in 1998, and the Ph.D. degree in electrical engineering from the University of South Brittany in 2002 (at formerly LESTER, now Lab-STICC). Then he joined the Signal Processing Laboratories at the Swiss Federal Institute of Technology (EPFL) for a three year post-doc fellowship to conduct research with the System Technology Labs at Intel Corp., Santa Clara. Since 2006 he is an Associate Professor at Polytech'Nice-Sophia school of engineering, and LEAT Laboratory, University of Nice-Sophia Antipolis - CNRS. His research interests are in design, exploration and optimization from early specifications with investigations in heterogeneous, reconfigurable and multiprocessor architectures, on a number of french, european and international collaborative research projects.

Maarten De Vries received his M.Sc in Telecommunications engineering from Télécom-ParisTech school (Paris, France), in 1991. As an engineer, he has more than 25 years of experience, through various R&D positions in major companies like Philips, and STMicroelectronics/ST-Ericsson. His expertise covers Hardware and Software developments as well as system architecture including virtual platforms modeling in SystemC. Since 2012, he is in charge of functional verification for ASIC development at Bull (Atos) for the BXI project (Bull Exascale interconnect for HPC). Since 2015, he's the WP7 lead of the MontBlanc2 project.

Said Derradji As a hardware architect at Bull, Said Derradji has been working first on several custom ASIC design interconnecting processors and focusing on cache coherency. He also worked on board design and participated on TERA-100 system delivery in 2010 (ranking 9 in Top500.org). Since 2012, he is working in the hardware architecture team at Bull, which specified recently the open exascale supercomputer, code-named SEQUANA. His areas of expertise are on ASIC/FPGA design, HPC servers architecture and on high performance interconnect technology such as the recently announced BXI (Bull Exascale Interconnect). He is BULL's representative at PCI-SIG (PCI Special Interest Group) and IBTA (InfiniBand®Trade Association) consortiums.

Cécile Belleudy is an Associate professor at University of Côte d'Azur. She currently leads the MCSOC team (Modelisation and system design of Communicating Objects) at LEAT laboratory and the Master Degree in Electronics, Systems and Telecommunications at University of Nice Sophia Antipolis. Her research interests are in the general field of System-on-Chip design with a specific interest in power optimisation, including power management, low power and real time scheduling, DVFS, DPM techniques and applications to multiprocessor architectures, multi-bank memories, operating systems.