IET Journals
The Institution of Engineering and Technology

Exploiting memory allocations in clusterized many-core architectures

ISSN 1751-8644
doi: 0000000000
www.ietdl.org

Rafael Garibotti[1], *Luciano Ost*[2], *Anastasiia Butko*[3], *Ricardo Reis*[4], *Abdoulaye Gamatié*[5], *Gilles Sassatelli*[5]

[1] *Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, Brazil*
[2] *Loughborough University, Leicestershire, UK*
[3] *Lawrence Berkeley National Laboratory, CA, US*
[4] *Federal University of Rio Grande do Sul, Porto Alegre, Brazil*
[5] *LIRMM lab., Montpellier, France*
✉ *E-mail: rafael.garibotti@pucrs.br*

Abstract: Power-efficient architectures have become the most important feature required for future embedded systems. Modern designs, like those released on mobile devices, reveal that clusterization is the way to improve energy efficiency. However, such architectures are still limited by the memory subsystem (i.e., memory latency problems). This work investigates an alternative approach that exploits on-chip data locality to a large extent, through distributed shared memory systems that permit efficient reuse of on-chip mapped data in clusterized many-core architectures. First, this work reviews the current literature on memory allocations and explore the limitations of cluster-based many-core architectures. Then, several memory allocations are introduced and benchmarked scalability, performance and energy-wise, compared to the conventional centralized shared memory solution to reveal which memory allocation is the most appropriate for future mobile architectures. Our results show that distributed shared memory allocations bring performance gains and opportunities to reduce energy consumption.

1 Introduction

In recent years, increased efforts to achieve power-efficient architectures are gaining attention over performance designs. One effort, which is already a reality, is to explore clusterization to implement heterogeneous processor architectures. For example, modern many-core architectures, such as those released on mobile chips, are inspired by ARM's so-called big.LITTLE technology [1]. Samsung with Exynos chips and Qualcomm with its Snapdragon CPU are few examples in such a trend. Instead of increasing core numbers to improve performance, these architectures are distinguished by their efficient utilization, where clusters are created to treat differently light activities from high workloads. This demonstrates that efficient many-core architectures will be constrained by memory allocation and architectural optimizations (e.g., clusterized shared/distributed - memory architectures and interconnection). Advanced memory optimizations are even more important as the number of cores integrated into a chip increases since the on-chip data exchange has a direct impact on system performance and energy-efficiency.

Studies show that processors are the major on-chip power consumers. According to [2], the processor consumption corresponds to more than 50% of the total energy capacity of the mobile device when the brightness of the screen drops to 25%. Specific benchmarks have been used to further analyze processor consumption on mobile devices [2–6]. These works revealed that the energy spent on data movement in mobile processors is significant, achieving on average 35% of total device energy [2]. Therefore, minimizing memory latency is a primary obstacle to improve processor performance efficiently and reduce energy consumption.

As a consequence, the memory subsystem has to evolve into some different organization that overcomes memory latency problems. According to [7], one way to mitigate these problems is by adopting a distributed memory system architecture. The use of distributed memories reduce the communication volume compared to the traditional shared memory solution, improving scalability and alleviating the presence of hotspots in the system. From this perspective, distributed on-chip memories will likely become mainstream for mitigating memory access bottleneck in emerging many-core architectures [8].

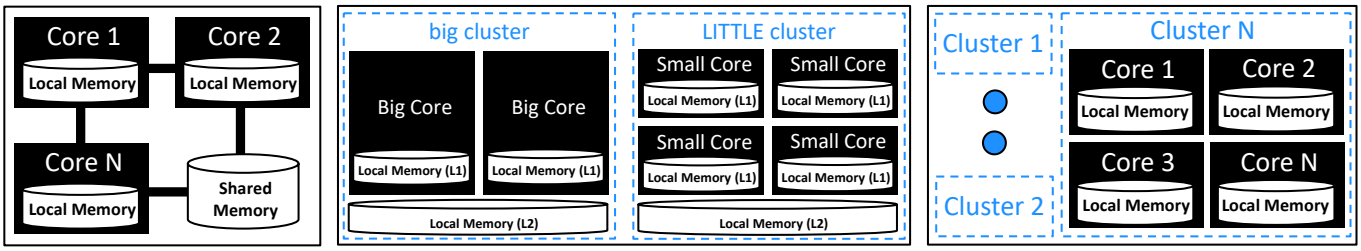
In this context, the *purpose* of this paper is to review the current literature on memory allocations and explore the limitations of cluster-based many-core architectures to reveal which memory allocation is the most appropriate for future mobile architectures. The *novelty* of this work focuses on the exploration of memory data allocation and architectural optimizations concerning emerging clusterized architectures which employ multiple memories in a unified memory space, organized according to the system and application requirements. In this sense, the use of *distributed shared memory* (DSM) allocations will allow the efficient reuse of on-chip mapped data in these cluster-based architectures.

The *contribution* of this work is twofold. First, we present a detailed study that reveals the limitations of cluster-based many-core architectures. To overcome the obstacles encountered, our second contribution is to analyze the benefits of employing a proper memory allocation in clusterized many-core architectures targeting three metrics: (*i*) scalability, (*ii*) performance, and (*iii*) energy efficiency. To perform these experiments, we have adapted a many-core RTL platform and developed a SystemC model that enables to investigate the data locality and memory allocations impact on large-scale clusterized many-core architectures.

In the rest of this paper, Section 2 presents related works that discuss on-chip data locality and memory allocations in clusterized many-core architectures. Section 3 overviews the evolution of many-core architectures, introduces memory allocation alternatives and presents the reference architecture abstractions. Section 4 investigates the impact of on-chip data locality on large-scale systems, revealing possible limitations. Section 5 shows experimental results on how on-chip distributed cache memory can influence the scalability, performance, and energy-efficiency of future cluster-based mobile devices. Finally, Section 6 points out the conclusions.

2 Literature Review

Mobile is by far the device most people own worldwide, snapping up more than 80% of the worldwide device shipments [9]. This explosive market growth motivated studies to further improve these many-core systems, from methodologies to optimize applications in mobile processors [10], to the use of *dynamic voltage and frequency scaling* (DVFS) aiming at reducing power consumption [11].



(a) Many-core Architecture (b) big.LITTLE Architecture (c) Clusterized Many-core Architecture

Fig. 1: Overview of many-core architectures, from the initial concepts, through the disruptive innovation of the big.LITTLE architecture with the introduction of *clusters*, reaching the reference clusterized many-core architecture of this work.

A crucial aspect underlying the proper development of mobile devices is to have, at the design phase, benchmarks that are representative of real applications, like MobileBench [3] and Moby [4]. These works claimed that to improve mobile efficiency and performance is essential to consider benchmarks that better match typical use cases along with their library and system interactions. Using such benchmarks, Gutierrez et al. [5] observed that smartphone applications could spend up to 15% of their execution time in the *operating system* (OS) code. Identifying these patterns may drive the development of future mobile devices.

These characterizations help to identify possible limitations on many-core architectures. One of the most striking is the inefficient use of on-chip memories, resulting in natural bottleneck appearances due to the increased number of cores, and consequently, a system performance degradation. Such limitations instigated researchers to work on data locality for many-core systems [2, 12, 13].

In [12], abstract models are proposed to investigate the effects of memory locality on the placement of threads in shared-memory applications with different distance metrics. They predict that future many-core systems will have a distributed memory architecture, with memory embedded in the processor tiles or stacked on top of them. Following this trend, some works already provide a distributed global address space, commonly using scratchpad memories, making the on-chip data memory to be physically distributed among all cores [14–16].

Yan and Fu [13] aim to reduce memory latencies for mobile devices by addressing data locality improvements through application profile and partitioning. This research focuses on the energy-efficient cache design in emerging mobile devices. They propose to partition the *level-2* (L2) cache into two distinct segments, which are user and kernel space. Authors argue that more than 40% of L2 cache accesses are OS kernel accesses in interactive applications. Their approach results in 15% energy savings without performance loss, due to its better utilization, thereby reducing the memory area.

Pandiyan and Wu [2] investigate the impact of data movement on overall energy consumption in smartphone devices. Results show that data movement is responsible for 35% of the total device energy. Authors also concluded that data exchange energy could reach 41% for realistic web browsing and on average 23.5% for processor stalls in current smartphones. This characterization study shows the impact that a poor memory subsystem causes in the efficiency of a mobile device.

Due to the great success of big.LITTLE architectures [1], recent studies also address architecture limitations in clusterized many-core systems. In [17], a technique for deploying hierarchical data flow graphs efficiently onto clustered many-core processors is proposed to help retrieve data locality, which is crucial for high performances and power consumption. On and Hussin [18] analyzed the impact that different many-core clustering methods have on multiprocessing architectures. To improve performance, Kakoee et al. [19] proposed a shared-L1 cache architecture for tightly coupled processor clusters. These works demonstrate that memory access latencies differ strongly in such architectures, depending on the data locality on the clusters. In architectures with *non-uniform memory access* (NUMA) characteristics, this problem is further aggravated, since the remote memory access imposes high overhead, making them more sensitive to data locality. This data locality problem was our motivation to deeply explore the memory allocation in these modern clusterized many-core architectures.

Madalozzo et al. [7] proposed a scalability evaluation in many-core systems, comparing shared and distributed memory architectures. Ma et al. [8] explore five different memory hierarchies, from centralized to distributed memories, revealing their effects on the scalability of many-core embedded systems. Results show excellent performance scalability using distributed memories for many-core embedded systems up to 32 cores. Unlike, our work extends both simple analyzes by exploring the scalability, performance, and energy-efficiency of different memory allocations in emerging clusterized many-core architectures. First, we investigate the limitations caused by the memory subsystem on large-scale many-core systems through a trace-driven simulation technique. Then experiments are performed through a precise and realistic platform.

Furthermore, some authors [2] argue the high stalled cycle energy consumption justifies the inclusion of more cores to achieve power-efficient architectures. Other authors claim that the number of cores will be regulated by the power wall and utilization wall [20]. In this regards, this work developed a SystemC model that shows the proper cluster size for each memory allocation. These tips can help future mobile devices to achieve the best trade-off between power consumption and performance.

3 Target Architecture Definitions

3.1 Many-core Architecture Models

This section overviews the evolution of many-core architectures. This architectural breakthrough is because the application performance requirements cannot be satisfied merely by raising the frequency of a monolithic core, which increases the chip's overall temperature and power consumption. One solution to overcome these bottlenecks is through the integration of multiple cores on the same chip [21].

Initially, many-core architectures were designed using symmetric or asymmetric cores organized through traditional memory hierarchies composed of several memory levels (e.g., L1, L2, and L3). As core count increases, *networks-on-chip* (NoC) have become the *de facto* standard on-chip communication infrastructure to maintain system scalability. Figure 1a is an example of a NoC-based many-core architecture with a hybrid memory architecture.

Such systems face two major challenges. First, to handle the execution of multiple applications concurrently (i.e., with different workloads). Second, to provide a proper memory architecture that promotes greater performance. Figure 1b shows the big.LITTLE architecture, in which *CPU clusters* of heavy and lightweight cores are coupled. Both cores are capable of executing the same instructions, and the difference lies in the way the cores handle the execution. Heavy cores are responsible for computing intensive tasks, such as high-definition video playback, whereas the lightweight cores handle lesser demanding tasks, such as text editing. Developers reported over 50% in energy savings for popular activities such as web browsing and music playback with the duo configuration [1]. Unlike, the second challenge remains open, which motivates us to investigate different memory allocations in this new target device.

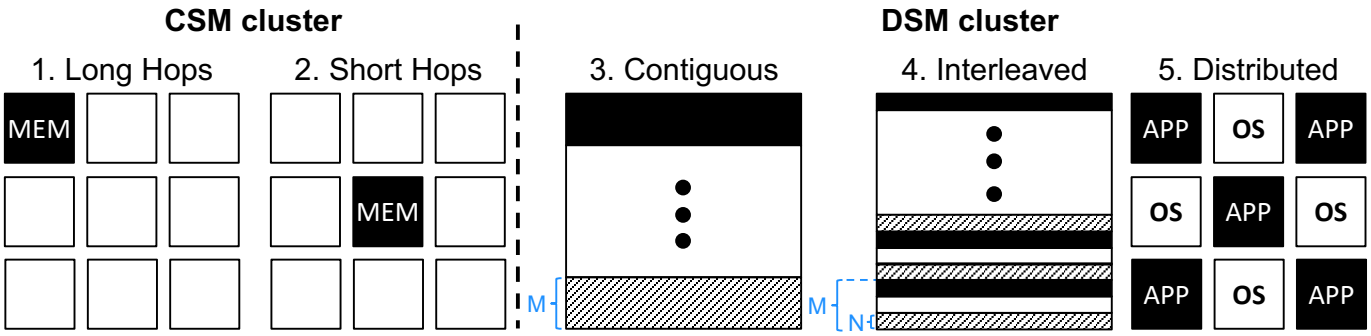


Fig. 2: Visual representation of five different memory allocations that cover CSM and DSM clusters.

3.2 *Shared Memory Models*

Inspired by ARM's big.LITTLE technology [1], modern mobile devices are being released with support for multiple clusters. To investigate the opportunities that adequate memory allocation can bring to these devices, we defined a reference clusterized many-core architecture, as shown in Figure 1c. From here, *tile* is defined as the set of core, local memory, and a network interface; and the *cluster* is a set of tiles. On top of that NUMA architecture, two shared memory models can be exploited within a cluster:

– *Centralized Shared Memory* (CSM): Host tile may share a region of its local memory with remote tiles.
– *Distributed Shared Memory* (DSM): All tiles participating in a given cluster share a region of their local memory with other participating tiles.

CSM is similar to current mobile architectures since a single local memory is shared within the cluster and used as L2 cache memory. On the other hand, DSM is the architecture to be exploited. As suggested by Ma et al. [8], DSM can ensure the energy-efficient expected for future mobile architectures. Note that in this implementation only instructions get distributed across tiles participating to a cluster whereas shared data remain hosted on a host tile. This distinction is because mobile applications are typically user-interactive applications, which involve rich GUI display using shared libraries and system codes, leading to a large instruction working set [2, 4, 5]. On the other hand, the data access patterns in mobile applications have small footprints and exhibit good locality [4, 5].

Figure 3 illustrates a comparison between CSM and DSM systems when both make available the same global address space size. To create a global address space in DSM, part of the local memory range in each tile is reserved for global use. All these regions are then composed in a global address range, logically contiguous but physically distributed inside the cluster. However, the shared memory solution shown in CSM clusters cause architecture limitations due to temporary high contention on single resources, which generate work imbalance and reduce performance.

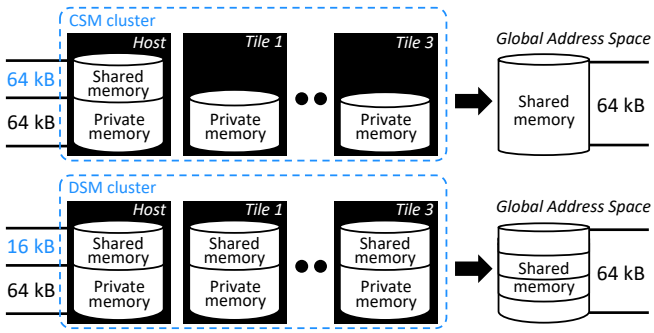


Fig. 3: A physical view of on-chip CSM and DSM systems. In CSM, Host has the physically shared memory, while in DSM, several tiles are clustered to build the global address space. This non-fixed shared memory architecture in DSM produces a more efficient design.

3.3 *Memory Allocation Alternatives*

To select the most suitable memory allocation for future mobile devices, we picked five distinct ones, from centralized to distributed memories, as suggested by Ma et al. [8]. They have different characteristics and are used either in current mobile devices or are trends presented in the literature to improve architecture scalability.

Figure 2 illustrates the exploited memory allocations. Two memory allocations are considered for the CSM system. *Long Hops* represents the memory allocation of current mobile devices, where cores participant of the same cluster shares single memory access. The second CSM allocation, referred to as *Short Hops*, consists in assigning all of the shared memory in a central tile for better reachability.

On the other hand, three memory allocations are assessed for DSM. *Contiguous* memory allocation is the default. It will be used in early DSM investigations, where instruction or data blocks are stored in each local memory. In contiguous memory allocation, the shared memory is divided by sequential memory blocks, and each pattern in Figure 2 represents one memory block of **M** kB that will be assigned to a tile. For example, considering a shared memory of 64 kB, a 4x4 cluster has a memory block size of 4 kB (*M=4*), while an 8x8 cluster has a memory block size of 1 kB (*M=1*). While Ma et al. [8] explore flavors of this unique DSM allocation, playing with interconnecting latencies, we further investigate other memory allocations. Depending on application behavior, contiguous memory allocation tends to degrade system performance due to the many requests coming from different cores simultaneously.

Interleaved memory allocation has the same memory block size as **M** kB. However, instead of subsequent memories, it consists of assigning every other memory position (**N**) to another tile in a cyclic fashion, where each *N* has 32-bit word size. Interleaved outperforms contiguous memory allocation for critical code regions. However, it is less suitable for regular applications that contain a balanced number of memory requests.

Finally, *Distributed* memory allocation assigns memory blocks on the sole basis of their nature: microkernel and applications, using a honeycomb pattern. Within a category (i.e., OS or APP), *Contiguous* memory allocation is used. This memory allocation is investigated because studies show that future mobile devices need to better handle the large and varied code footprints of interactive applications [5, 13]. Some architectures already presented such asymmetric behavior, as Cortex-A57 and Cortex-A73 (as will be seen in Table 2). Thus, this growing gap between data and code footprints must be effectively addressed.

3.4 *Reference Architecture Abstractions*

Different abstractions (Figure 4) of our reference clusterized many-core architecture has been developed to accurately evaluate promoted memory allocations, considering different metrics like support for large scenarios (Section 4), accuracy (Section 5.2) and performance and energy savings (Sections 5.3 and 5.4). A trace-driven simulation technique is employed to investigate the on-chip data locality impact on large-scale system performance, allowing the exploration of clusters composed of a large number of cores. This

Fig. 4: Different abstraction layers are necessary to cover evaluations with distinct goals, ranging from accurate to fast simulations.

experiment aims to find out the possible limitations of cluster-based designs. Further, the trace-driven technique collects data (i.e., traces) to be used afterward.

A precise and realistic platform is necessary to analyze the hardware impact of this paradigm shift, from centralized- to distributed-shared memory architectures. This platform model described in RTL provides accurate data, such as memory latency, which shows the scalability achieved by the DSM approach. Finally, a SystemC model is developed using as input the same trace file captured from the trace-driven technique. Such abstraction allows the exploration of several memory allocations, providing hints on performance and energy-wise for future clusterized many-core architectures.

4 Limitations on Clusterized Many-core Architectures

4.1 Introduction

To assess the influence of on-chip data locality on large-scale mobile architectures, more efficient simulation and modeling techniques become stringent requirements to investigate systems at these scales. In this context, some system-level architecture simulators (e.g., gem5 [22]) have gained momentum to perform design space exploration. For example, some works [23, 24] used gem5 to accurately model and simulate ARM cores, such as Cortex-A9 and Cortex-A15.

In previous work [25], we demonstrate that it is possible to model and exploit state-of-the-art many-core architectures using a system-level architecture simulator. The proposed big.LITTLE architecture, based on the Samsung Exynos 5 (model 52422), was evaluated against a real computer board to assess the models' accuracy. Our gem5 model predicts performance with a 20% error [25].

To exploit the big.LITTLE architecture model for large-scale designs, we applied a trace-driven technique in gem5 [26]. This technique decreases simulation complexity by abstracting away core execution into traces. Results show that cache misses of all cores follow the same memory access time pattern. Thus, this pattern can be used as a trace template to be replicated among more cores. This approach revealed opportunities to investigate performance scalability limitations on emerging large-scale clusterized mobile devices. Besides, extracted traces are also used to explore different memory allocations in many-core architectures via a SystemC model.

4.2 Trace-driven Simulation

Figure 5 shows the trace-driven simulation flow, which is composed of three phases: (*A*) trace collection, (*B*) trace processing and (*C*) trace simulation. First, we define the target system (i.e., our big.LITTLE architecture). The hardware architecture comprises up to 4 cores, each having its private instruction and data caches. Communications between cores and external memory are achieved through a stream of request and response events via a communication infrastructure that comprises a shared level-2 cache memory, such as those found in the Samsung Exynos 5 design. After characterization, execution traces are collected (*A*) from gem5 simulator [26], at the boundary between private caches and interconnect.

Trace files are then processed (*B*) and prepared for use in the target trace-driven simulation. In the last phase (*C*), recorded traces are injected through injectors. Whenever a request is issued from an injector to the memory, a regular simulation procedure is initiated, so that interconnect and memory congestion are adequately accounted. In this work, this technique was used to explore up to 256 cores.

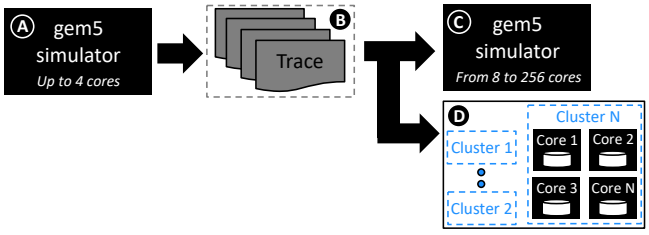


Fig. 5: Our experimental flow has two proposals. First three phases (*A*-*C*) show the trace-driven simulation flow, used to extrapolate the available number of cores in mobile architectures. On the other hand, the SystemC model (*D*) uses the traces collected up to the second phase (*B*) as input files to evaluate different memory allocations.

Table 1 Simulation results (gem5): speedup, memory footprint and accuracy.

Applications		FFT	HIST	MJPEG	OCEAN	REDUCT	SW
Simulation time (s)	*FS*	119.57	60.01	3.62	15.92	97.82	3.05
	TD	14.08	0.073	0.03	2.82	0.14	0.03
Speedup gain		8.5	800	136	6	734	122
Trace file (GB)		11.6	0.06	0.04	2.48	2.49	0.05
Execution time (s)	*FS*	1.745	0.2672	0.0264	0.6027	0.3716	0.0186
	TD	1.852	0.2649	0.0262	0.6025	0.3715	0.0178
Exec. Time Error (%)		5.79	0.85	0.98	0.03	0.02	4.26

4.3 Limitations Assessments

Trace files are collected from the *timing full-system* configuration (FS) of the gem5 simulator (i.e., our reference architecture) and then executed in our *trace-driven* (TD) simulator without any architectural change. Table 1 shows for several benchmarks the simulation speedup and observed mismatch, expressed in execution time on 4-core ARMv7 architecture. Results provided by the TD simulation show a speedup gain of up to 800×, depending on the workload nature, along with an execution time error below 5.79%. The highest percentage of execution time error comes mainly from applications composed of multiple threads with dependencies between them, e.g., induced by synchronizations. Another source of error comes from the cold-start bias problem [27], i.e., the mismatch caused by simulations that begin with an empty cache memory subsystem while the reference simulation has information on it. Although we have eliminated the cold-start error for private caches (i.e., L1), we are not able to predict the other levels of cache (e.g., L2). This small mismatch is seen in OCEAN and REDUCT workloads. In short, results shown in Table 1 reveal that the trace-driven technique can be used to decrease simulation time while maintaining accuracy.

The proposed trace-driven simulation technique reveals the limitations of future large-scale cluster-based mobile architectures. Figure 6 shows the platform behavior when the number of cores varies for TD, using FS as the 1-core reference. Experiments with up to 256 cores have been conducted, showing that most of the simulation effort falls into cache subsystem simulation (i.e., cache coherence/bus snooping). As the results show that 88% of simulation runtime is spent on handling the cache subsystem, this will be the biggest villain for future mobile devices. Further, it is well known that on-chip caches consume a significant fraction of the total processor energy budget [13]. However, this is the first time someone has quantified these limits targeting future clusterized mobile devices.

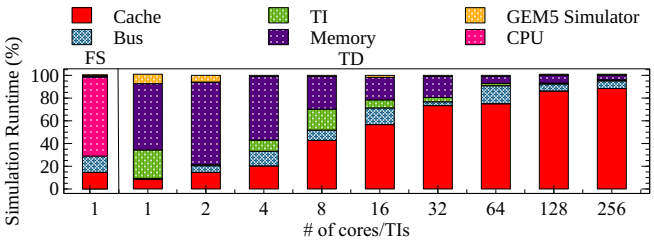


Fig. 6: Simulation time breakdown for *timing full-system* (FS) and *trace-driven* (TD) simulation modes, highlighting the increasing pressure that cache subsystem will cause in large scenarios.

Table 2 Level 1 memory system of Cortex-A series processors [28].

SoC	*cores*	*Technology*	*L1 cache size*
Qualcomm Snapdragon S4 Play	Cortex-A5	45nm	32kB
Samsung Exynos 4 Dual	Cortex-A9	45nm	32kB
Samsung Exynos 5	Cortex-A15 Cortex-A7	28nm	32kB 8kB, 16kB, 32kB, 64kB
Qualcomm Snapdragon 810	Cortex-A57 Cortex-A53	20nm	inst.: 48kB, data: 32kB 8kB, 16kB, 32kB, 64kB
Qualcomm Snapdragon 660	Cortex-A73 Cortex-A53	14nm	inst.: 64kB, data: 32kB, 64kB 8kB, 16kB, 32kB, 64kB
Qualcomm Snapdragon 845	Cortex-A75 Cortex-A55	10nm	64kB 16kB, 32kB, 64kB

Table 3 Summary of architecture set evaluated throughout all experiments. Note that not all parameters are related to all abstraction models, e.g., the target technology is only relevant to the RTL model.

Cluster sizes	4x2; 4x4; 4x8; 8x8
Topology	2-dimension mesh
Communication	Dual 32 bits channel NoC @ 500MHz
CPU core	32 bit, 5 pipeline stage Microblaze ISA @ 500MHz
CPU caches	8kB-64kB direct mapped L1 I$ and D$ caches, 256 bit/line
Tile local shared RAM	8kB-64kB
Tile local private RAM	64kB microkernel + 64kB code/data
Thread assignment	1 worker thread/tile for avoiding performance penalties from context switching, main thread on host tile
Target technology	45nm CMOS bulk - FreePDK library [30]
Memory information	Evaluation by using NVSIM tool [32]

Furthermore, Pandiyan and Wu [2] investigated the impact of data movement using a Samsung Galaxy S3 smartphone, which houses an Exynos SoC with a quad-core Cortex-A9 processor [29]. They observe that, on average, 23.5% of the device's total energy is spent on stall cycles in the application processor. This result is quite similar to the simulation runtime spent on cache memory using 4 cores, as shown in Figure 6. This corroborates that our trace-driven approach can be used to represent and evaluate current mobile devices.

This high-level abstraction of a clusterized many-core architecture shows that we can simulate faster and still maintain the accuracy of experiments. Besides, results support that cache memory is the bottleneck for future mobile devices, being our motivation to explore memory allocations that can alleviate the memory subsystem.

5 Memory Allocation Analysis

5.1 Experimental Setup

Over the last few years, several generations of mobile architectures have been designed, including the introduction of the big.LITTLE architecture. While the miniaturization of process technology and powerful processors were expected, one feature caught the attention due to its constancy. Table 2 shows the level-1 cache memory sizes in different multicore and big.LITTLE architecture generations. The latest releases of multicore architectures presented the same L1 cache size (32kB) for data and instruction on each core. In big.LITTLE architectures, heavy cores have almost the same fixed cache memory size (i.e., 32kB or 64kB). However, lightweight cores have a broad range, but of the same L1 cache memory sizes, from 8kB to 64kB. Also, some cores, such as Cortex-A57, have larger instruction cache sizes than data cache sizes, which corroborates with the discussion on distributed memory allocation in Section 3.3. These are the cache memory sizes that must be evaluated to determine what impact the memory allocations can have on future clusterized many-core architectures.

Table 3 gives the details of the architecture configurations used throughout these experiments, highlighting the extracted range of cache memory sizes and small-medium sized clusters. Furthermore, Table 2 showed that technological advances do not directly influence the L1 cache size. In this regard, the technology was set at 45nm using an open-source library [30], which facilitates the replication of the experiments by other researchers.

For memory allocation exploration, a critical step is communication within a cluster in which the network topology selection has a direct impact on the overall system performance. However, the NoC topology exploration results are out of the scope of this paper. In this sense, we chose to explore the 2-dimension mesh topology. Besides being one of the most used topologies in literature, routing in a 2-dimensional mesh is easy, resulting in potentially small switches, high capacity, short clock cycle, and overall scalability [31]. Furthermore, 2-dimensional mesh topology well matches the planar, regular layout of a cluster-based design, which can increase the scalability of architectures such as ARM's big.LITTLE.

In this work, we consider a set of workloads from scientific to multimedia computing domain. These workloads were selected according to their characteristics to deeply evaluate the different memory allocations. Some of them come from the SPLASH-2 benchmark suite [33]: *Radix*, *Barnes*, *LU*, and *Ocean*, which are relevant due to the presence of multiple dependencies on corresponding algorithms. In addition, we adopted further workloads: *Motion JPEG* (MJPEG), *Finite Impulse Filter* (FIR), *Smith-Waterman* (SW), *Advanced Encryption Standard* (AES), *Histogram graph computing*, *Merge Sort*, *N-body* for simulating a dynamical system of particles, *Reduction of vectors*, *Vector Operations* (VO) and *Fast Fourier Transform* (FFT). These workloads cover from low to intensive memory utilization and have a wide coverage of memory locality.

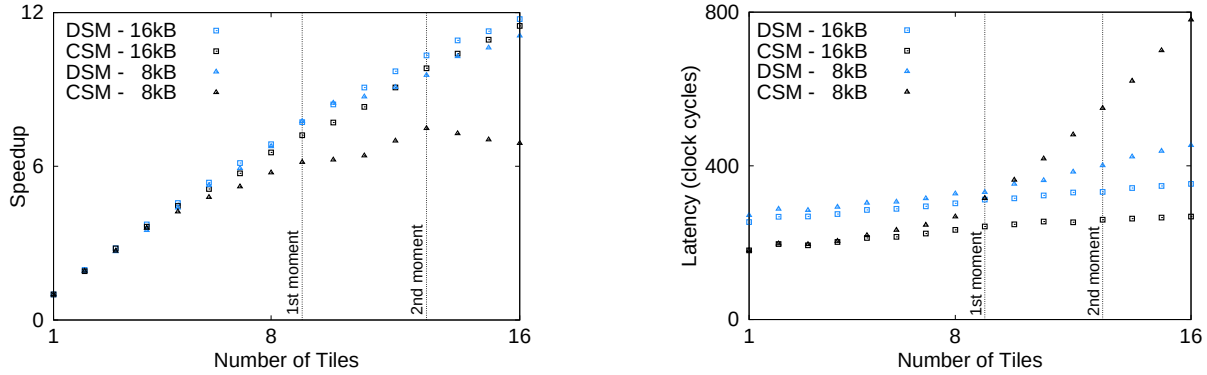
5.2 Scalability Analysis

To measure the scalability impact on clusterized distributed and centralized-memory architectures, we first model both architectures in a precise and realistic RTL platform [34]. This platform is an open-source RTL multiprocessor core designed for scalability and adaptation. A shared-memory programming API is supported in such distributed memory platform based on the POSIX thread API. It features most common primitives such as locks and semaphores so that porting from existing code is trivial [34]. Furthermore, this programming model has the advantage of being scalable and have easy programmability.

To maintain the scalability of many-core architectures, designers have split workloads among multiple threads to scale system performance through parallel applications effectively. However, scalability may be compromised by several aspects, including contention for shared resources and available tiny cache memory sizes on mobile devices. In this regard, workloads were distributed to run on physically centralized- and distributed-shared memory cluster-based architectures to evaluate the gains and penalties inherent in both implementations. The speedup of each workload is normalized to the minimum cluster size (1-tile) to facilitate comparison. First, the Smith-Waterman kernel was evaluated in cluster sizes ranging from 1 to 32. Due to its intrinsic characteristics as a compute-intensive workload, both CSM and DSM memory architectures work properly, resulting in limited cache miss rate and near-linear scalability.

On the contrary, video applications (e.g., MJPEG) often used in mobile devices are memory-intensive workloads, suffering a significant impact from the implemented memory architecture. For example, Figure 7a shows that the CSM model (*Long Hops* type) presents a speedup limitation for small cache memory sizes (i.e., 8kB), which impacts on the performance of lightweight cores, such as Cortex-A7 and Cortex-A53. This scalability degradation is shown through the two highlighted moments in Figure 7a. Before the first moment, the scalability is almost proportional to the increase in the number of tiles, since the latency remains practically constant as shown in Figure 7b. Between the two moments, the latency begins to increase with each addition of a new tile, however, there is still a slight increase in speedup. It occurs because the time to process the local cache information is longer than the time to request a new instruction block. Finally, from the second moment, the tiles are no longer able to process the data sufficiently to overcome the latency time, causing the increase of tiles to worsen the scalability.

On the other hand, results show that by aggregating the bandwidth of available multiple distributed memories, MPJEG presents



(*a*) Performance (*b*) Cache Miss Latency

Fig. 7: MJPEG performance (*a*) for several cluster sizes and configurations are presented and correlated to average cache miss latency (*b*). Results show a bottleneck principle as the number of tiles increases, highlighting the huge impact that memory allocation makes on memory-intensive workloads.

performance improvement of up to 52% in the DSM design (*Contiguous* type) in a small cluster size (i.e., 4x4), as shown in Figure 7a. DSM designs help alleviate the congestion occurring on the communication network because it avoids having requests for the same tile, resulting in decreased cache miss latency as illustrated in Figure 7b. This increase in cache miss latency in CSM over DSM results in longer execution time.

As this memory-intensive workload presents a high-pressure principle on the communication/memory subsystem, detailed evaluations were performed according to the monitoring information shown in Figure 8. Here, the comparative DSM design is configured by installing workload code over 10 tiles out of the 16 tiles available in a 4x4 cluster.

Figure 8a clearly shows the workload code installed on host tile (CSM) or distributed across several tiles (DSM). These different memory architectures have a direct impact on the MJPEG scalability. Figure 8b shows the NoC bandwidth. In CSM, all traffic naturally converges to the host tile, leading to a communication and memory access bottleneck. It causes an overload on the host tile to handle all remote memory access, as illustrated in Figure 8c.

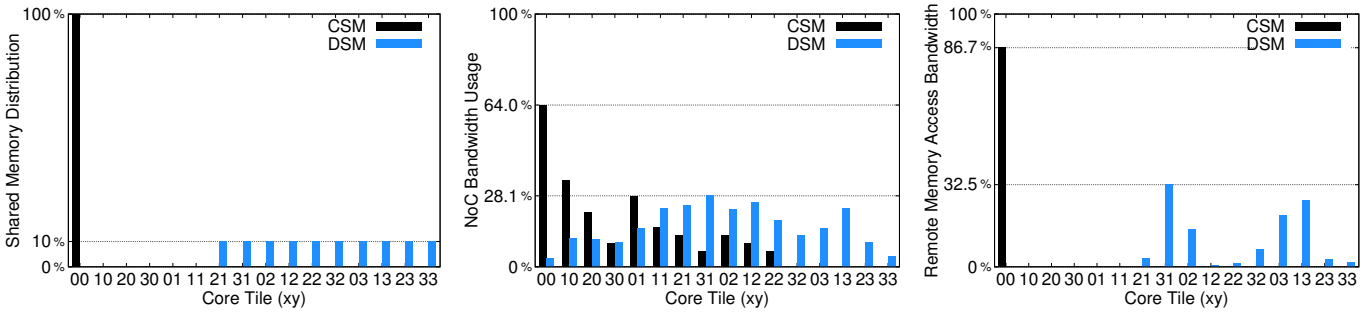
In contrast, the DSM system does not suffer from this problem, as the communication load is distributed across several tiles, as illustrated in Figure 8b. Results show that the peak NoC bandwidth usage has been reduced from 64.0% to 28.1%. Behavior may significantly differ from one application to another, as critical code regions are not homogeneously distributed across the global address space, as clearly shown in the tiles that deal with more remote memory requests in Figure 8c, e.g., *tiles 31* and *13*. Although there is a significant imbalance in memory traffic, and therefore some critical regions in NoC traffic. These results clearly show that DSM outperforms CSM, exhibiting an improvement over 2× in NoC traffic and almost 3× to handle memory requests for a memory-intensive workload.

This section showed a first impact, where the proper use of on-chip distributed memories might mitigate memory requests for memory-intensive workloads and performs similarly for compute-intensive workloads. These results corroborate with those presented by Ma et al. [8], indicating that distributed-shared memory architectures are a prominent approach for future mobile devices to alleviate memory congestion problems and extend system scalability. Furthermore, this approach can keep constant the cache memory sizes in future mobile devices.

5.3 *Performance Analysis*

Section 4 showed a massive perturbation in-memory communication when the number of cores in a cluster-based architecture grows exponentially. Further, investigations on scalability in Section 5.2 revealed a pattern of uneven access to multiple on-chip memories when contiguous memory allocations are used to treat memory-intensive workload. Previous analyzes have shown that memory allocations may help alleviate the problem to achieve better scalability and possibly better performance in future clusterized many-core architectures. However, for further analysis, it is mandatory to raise the abstraction level above that offered by RTL. The presented trace-driven simulation technique was adapted for this purpose, the collected traces are no longer injected into another gem5 simulation to evaluate future mobile designs, but instead, are injected in a cycle-accurate model comprising communication architecture and memories. This cycle-accurate model can analyze system behaviors by exploiting a wide range of parameters, such as different memory allocations, memory latency, bus channel width, and others.

Figure 5 describes our experimental flow, where we developed a SystemC model (*D*) to replace the trace-driven simulation (*C*). From here, the target simulation contains cycle-accurate SystemC NoC



(*a*) Shared Memory Distribution (*b*) NoC Bandwidth usage (*c*) Remote Memory Access

Fig. 8: Evaluation of different metrics in a cluster composed of 4x4 tiles executing MJPEG workload with 10 threads. Regarding memory distribution, CSM has one host responsible for storing the entire workload code. On the other hand, DSM has a smooth code distribution over the available tiles. These memory architecture differences have a direct impact on the MJPEG scalability. Furthermore, the other two metrics have proven that DSM outperforms CSM by reducing memory latency.

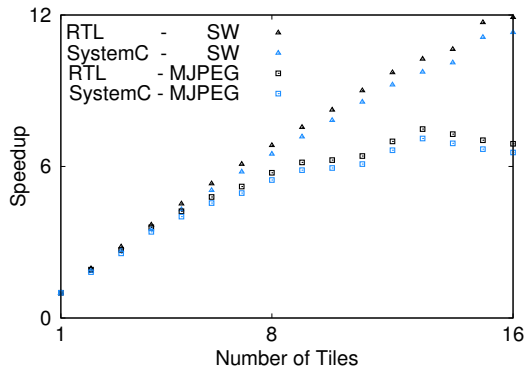


Fig. 9: Consistent behavior between RTL and cycle-accurate model, enabling the utilization of this SystemC model for memory allocation explorations.

and memory models. Trace injectors read trace files and, according to the given memory allocation, generate the memory request transactions that are then transmitted to the NoC.

To validate the cycle-accurate model, consistent behavior was obtained by comparing normalized speedups in the RTL model, as presented in Section 5.2, and SystemC model for Smith-Waterman and MJPEG workloads. Figure 9 presents speedup mismatches below 5% using a CSM mode with 8kB shared memory. Besides such accuracy, *the cycle-accurate model is at least one order of magnitude faster than RTL model*. In light of these advantages, the proposed cycle-accurate model is more suitable for dealing with large systems and easy to assess different memory allocations, encouraging its use to properly adjust the cluster size accordingly without losing its efficiency.

On top of this cycle-accurate model, we set up the five memory allocations proposed in Section 3.3. To illustrate the performance improvements brought about by careful selection of these memory allocations, Figure 10 shows CSM Long Hops-normalized execution time for different cluster sizes. These experiments were conducted considering from current octa-core up to 64-core systems. This ceiling is because maintaining a reduced SoC size is crucial to the success of mobile phones. Besides, Figure 6 showed the memory subsystem is responsible for more than 74% of the execution time in systems with more than 32 cores, indicating a saturation principle.

Results demonstrate that evaluated memory allocations can bring a huge impact on cluster performance, alleviating contention in shared resources by reducing memory accesses. CSM *Short-hops* memory allocation leads to a hardly noticeable improvement performance-wise compared to CSM *Long-hops*. It originates from previously observed memory bottleneck not addressed by this memory allocation, which only relocates all shared data to a central tile, as shown in Figure 2.

On the other hand, DSM always performs better than CSM architectures, as first introduced by scalability analysis in Section 5.2. However, different performances are observed over the three distributed memory allocations. *Contiguous* memory allocation tends to preserve critical code regions due to the huge block sizes, making these critical code regions not being homogeneously distributed across the global address space. It is the reason for *Interleaved* and *Distributed* memory allocations overcome *Contiguous* memory allocation. The better distribution of memory requests and the charge reduction over the tiles minimize the possibility of memory bottleneck for this kind of imbalance memory requests.

Interleaved and *Distributed* memory allocations obtain a significant reduction in execution time for most workloads, especially for large array sizes. The difference between them heavily depends on the behavior of each application and the size of the performed cluster. For further analysis, Figure 11 depicts the same information. However, emphasizing the average execution time of the entire set of benchmarks. It highlights that for performance-wise, *Interleaved* memory allocation is the best choice. Besides, for large systems like 8x8 tile array size, up to 3× reduction in execution time is observed. It shows the criticality of memory access for such workloads.

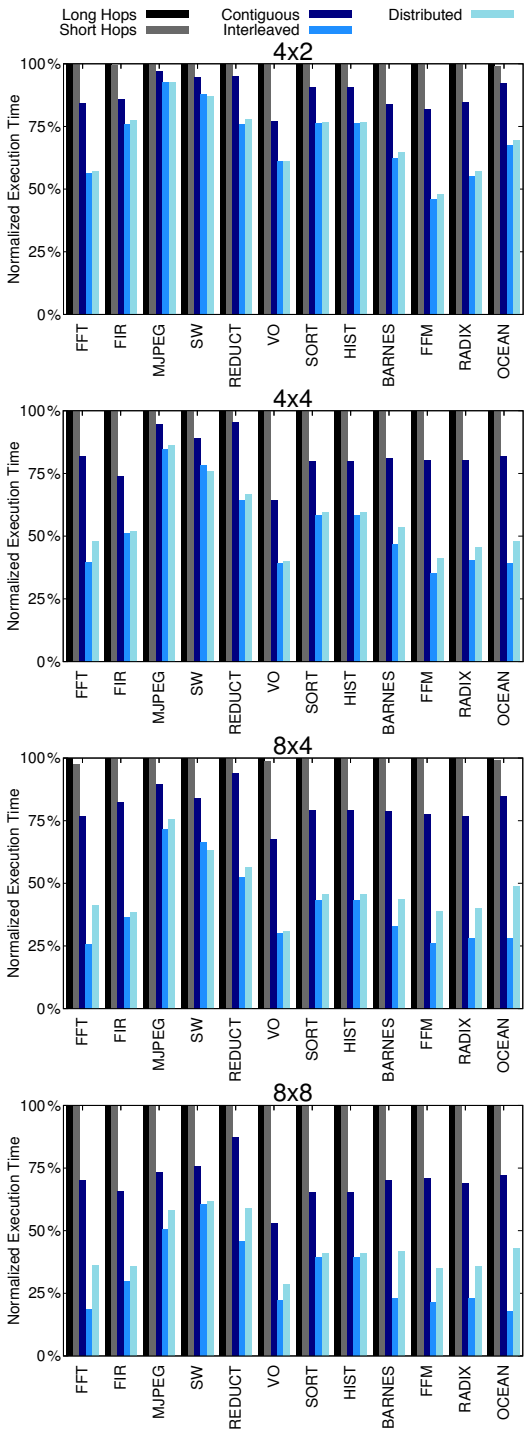


Fig. 10: Execution time for those memory allocations is categorized using the following cluster sizes: 4x2, 4x4, 8x4 and 8x8.

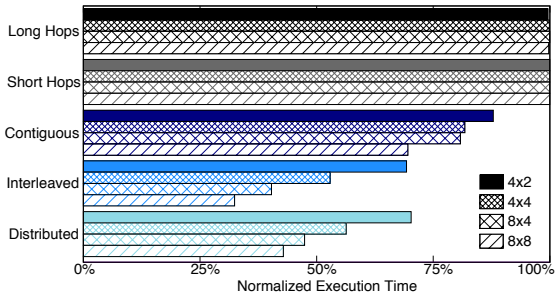


Fig. 11: Averaged execution time, highlighting that the interleaved memory allocation is the best choice for performance-wise.

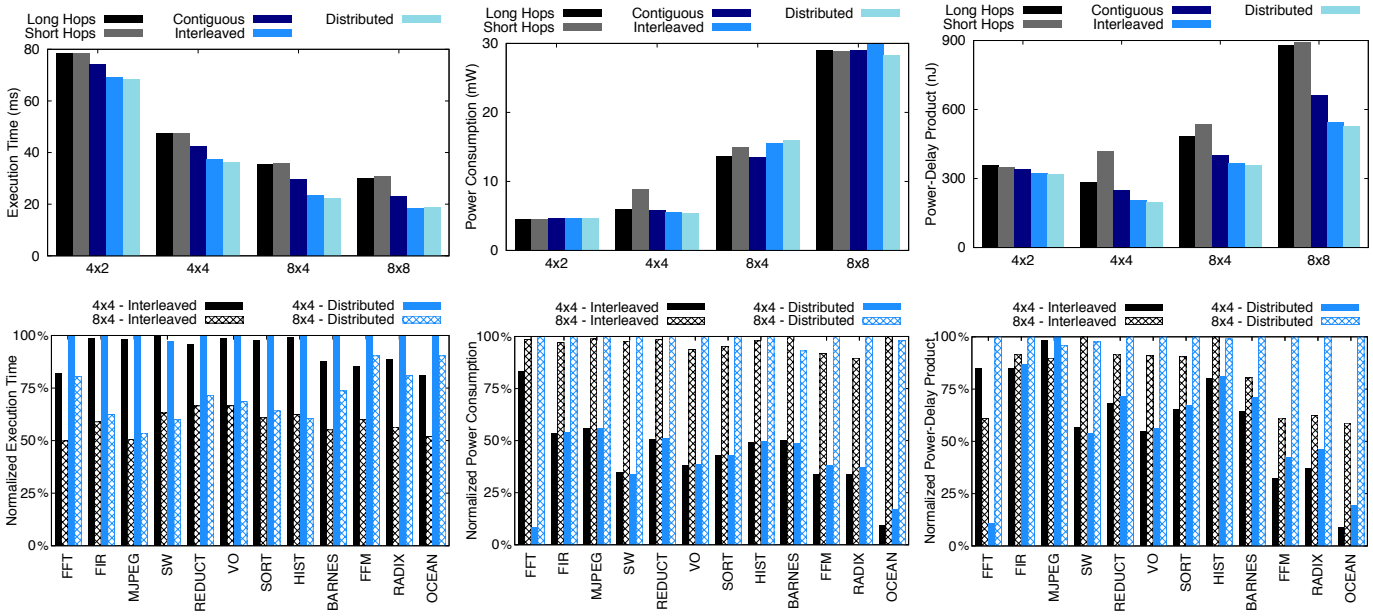


Fig. 12: Execution time, power consumption and power-delay product for (*top*) all memory allocations using Smith-Waterman keel and (*bottom*) normalized metrics using all benchmarking kernels for the two best performance-wise memory allocations.

5.4 Energy Efficiency Analysis

Mobile devices are driven by energy efficiency because this metric directly affects consumer perception. As battery-dependent devices, it reduces the time between recharges and can still relieve the overall weight of the device. In this sense, we first created a power consumption model based on the model presented by Hu and Marculescu [35]. It models a regular tile-based NoC architecture using a mesh topology with XY routing, similar to the one used in this paper. Monitors spread throughout the system have collected the required information. The remaining data to characterize the model were obtained through tools shown in Table 3.

However, power consumption varies depending on the program being executed. Although this metric is important for measuring the heating of the device, choosing a memory allocation based only on the power consumption metric can be misleading. In this sense, we must measure power and performance together for a given program execution by using a fused metric, such as the *power-delay product* (PDP). In general, the PDP-based formulations are more appropriate for low-power, portable systems in which battery life is the primary index of energy efficiency [36].

Figure 12 (*top*) presents the execution time, average power consumption and corresponding power-delay product for different cluster sizes for all five memory allocations. For a fair comparison, all experiments run a 64-thread Smith-Waterman application. Note that in a system with 64 tiles (8x8 cluster size), each thread is allocated to a single tile. Unlike, other cluster sizes must execute more than one thread per tile. For example, 8 threads are allocated to each tile in a 4x2 cluster size.

Due to the compute-intensive nature of this workload, the best performance is obtained for large cluster sizes. However, the power-delay product is interestingly obtained for 4x4 cluster sizes for most memory allocations. It suggests that the overall increase in communication distance, related to the code distribution across more tiles, is not compensated by the reduction in execution time.

To refine the choice in finding an energy-efficient memory allocation for future cluster-based mobile devices, we focus on the two best performance-wise memory allocations discussed in Section 5.3: *Interleaved* and *Distributed*. In addition to their intrinsic ability to better spread memory traffic across multiple memories, the average transaction route length also impacts on energy savings.

Figure 12 (*bottom*) shows the same data for all benchmark kernels when *Interleaved* and *Distributed* memory allocations set to *4x4* and *8x4* cluster sizes are evaluated. Results are normalized for each benchmark against the worst memory allocation performance. The benefit is tightly correlated with the characteristics of benchmarks. For example, *Distributed* memory allocation presents better results for compute-intensive workloads (e.g., SW). On the other hand, *Interleaved* memory allocation is best for memory-bound workloads, such as MJPEG.

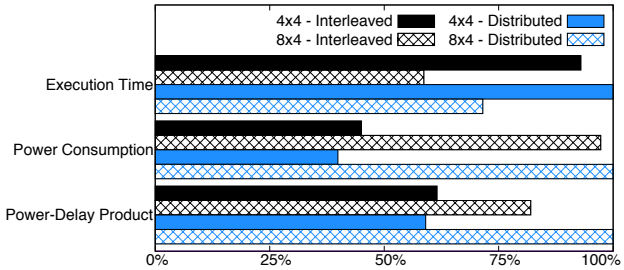


Fig. 13: Averaged and normalized execution time, power consumption and power-delay product, highlighting that the distributed memory allocation is the best choice for energy saving clusters.

As these memory allocations have distinct advantages, the results were aggregated into Figure 13 and show the average for all benchmarks. From these results, two suggestions may be drawn. For *energy saving clusters*, *Distributed* memory allocation is the best choice, and the cluster size shall be set to 16 nodes at most. However, for *performance clusters*, *Interleaved* memory allocation presented better results, and larger cluster sizes may be used at the expense of lower energy efficiency.

It is not a final decision because the application scalability has a huge impact and must be considered. For kernels whose execution time is dominated by sequential code regions, cluster size shall be limited. Unlike, highly parallelizable kernels may benefit from larger cluster sizes at the cost of increased power consumption.

In this work, the number of cores inside different cluster sizes was investigated to find an efficient clusterized many-core architecture concerning scalability, performance and energy efficiency. For many-core architectures like ARM's big.LITTLE [1], our insights show that *energy saving* clusters can be extended up to 16 lightweight cores if we have parallel applications or a sufficient amount of concurrent applications to justify this choice. On the other side, *performance* clusters have no restriction on cluster sizes, which means that it will be restricted by mobile SoC size and processor performance in the coming years, depending on the processing

demand. These observations can guide the development of future energy-efficient clusterized many-core architectures.

6 Conclusions

Cluster-based mobile devices are a reality and inspired us to explore the limitations of this new target device. Thanks to a novel trace-driven simulation technique, results might be extended to large cluster-size architectures (up to 256 cores). It reveals that the greatest villain for future mobile devices is the memory subsystem.

Explorations on five different memory allocations are performed to find power-efficient architectures. *Centralized shared memory* is the most common architecture found in the literature. However, initial investigations on scalability have revealed an uneven access pattern to multiple on-chip memories when using *Contiguous* memory allocation. Differently, *distributed shared memory* has the distinct advantage of aggregating the bandwidth of multiple physical memories, resulting in performance gains and opportunities to reduce energy consumption.

For future cluster-based mobile devices to benefit from better throughput, *Interleaved* is the most appropriate memory allocation, and larger cluster sizes may be used at the expense of lower energy efficiency. This decision may be guided by the scalability of a given application, making it inefficient if sequential code regions dominate the evaluated kernel. Unlike, for workloads where energy-savings are mandatory, our experiments show that cluster size shall be set to 16 tiles at most along with the *Distributed* memory allocation.

In addition to helping discuss the impact of architectural decisions made regarding memory allocations for future cluster-based mobile devices. The trace-driven SystemC model leaves the legacy that can be used to explore other many-core architectures, or even used to evaluate a specific application on mobile devices quickly.

7 References

1 ARM: "big.LITTLE Technology: The Future of Mobile", 2013, http://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf.

2 Pandiyan, D., and Wu, C.-J.: "Quantifying the Energy Cost of Data Movement for Emerging Smart Phone Workloads on Mobile Platforms", *IEEE International Symposium on Workload Characterization (IISWC)*, October 2014, pp. 171–180.

3 Pandiyan, D., Lee, S.-Y., and Wu, C.-J.: "Performance, Energy Characterizations and Architectural Implications of An Emerging Mobile Platform Benchmark Suite - MobileBench", *IEEE International Symposium on Workload Characterization (IISWC)*, September 2013, pp. 133–142.

4 Huang, Y., Zha, Z., Chen, M., and Zhang, L.: "Moby: A Mobile Benchmark Suite for Architectural Simulators", *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, March 2014, pp. 45–54.

5 Gutierrez, A., Dreslinski, R.G., Wenisch, T.F., Mudge, T., Saidi, A., Emmons, C., and Paver, N.: "Full-System Analysis and Characterization of Interactive Smartphone Applications", *IEEE International Symposium on Workload Characterization (IISWC)*, November 2011, pp. 81–90.

6 Bournoutian, G., and Orailoglu, A.: 'Application-Aware Adaptive Cache Architecture for Power-Sensitive Mobile Processors', *ACM Transactions on Embedded Computing Systems (TECS)*, **13(3)**, December 2013, pp. 41:1–41:26.

7 Madalozzo, G., Duenha, L., Azevedo, R., and Moraes, F.G.: "Scalability Evaluation in Many-core Systems due to the Memory Organization", *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, December 2016, pp. 396–399.

8 Ma, S., Huang, M., Cartwright, E. and Andrews, D.: "Scalable Memory Hierarchies for Embedded Manycore Systems", *International Conference on Reconfigurable Computing: Architectures, Tools and Applications (ARC)*, March 2012, pp. 151–162.

9 Gartner: "Gartner Says Worldwide Device Shipments Will Increase 2 Percent in 2018, Reaching Highest Year-Over-Year Growth Since 2015", October 2017, http://www.gartner.com/newsroom/id/3816763.

10 Cheng, K.-T. T., Yang, X., and Wang, Y.-C.: "Performance Optimization of Vision Apps on Mobile Application Processor", *International Conference on Systems, Signals and Image Processing (IWSSIP)*, July 2013, pp. 187–191.

11 Reddy, B.K., Singh, A.K., Biswas, D., Merrett, G.V. and Al-Hashimi, B.M.: "Inter-cluster Thread-to-core Mapping and DVFS on Heterogeneous Multi-cores", *IEEE Transactions on Multi-Scale Computing Systems*, **PP(99)**, September 2017, pp. 1–14.

12 Khanjari, S.A. and Vanderbauwhede, W.: "Evaluation of the Memory Communication Traffic in a Hierarchical Cache Model for Massively-Manycore Processors", *Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, February 2016, pp. 726–733.

13 Yan, K. and Fu, X.: "Energy-Efficient Cache Design in Emerging Mobile Platforms: The Implications and Optimizations", *IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)*, April 2015, pp. 375–380.

14 Alvarez, L., Vilanova, L., Moreto, M., Casas, M., Gonzalez, M., Martorell, X., Navarro, N., Ayguade, E. and Valero, M.: "Coherence protocol for transparent management of scratchpad memories in shared memory manycore architectures", *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, June 2015, pp. 720–732.

15 Shoushtari, M. and Dutt, N.: "SAM: Software-Assisted Memory Hierarchy for Scalable Manycore Embedded Systems", *IEEE Embedded Systems Letters*, **9(4)**, December 2017, pp. 109–112.

16 Ceriani, M., Secchi, S., Villa, O., Tumeo, A. and Palermo, G.: "Exploring Efficient Hardware Support for Applications with Irregular Memory Patterns on Multinode Manycore Architectures", *IEEE Transactions on Parallel and Distributed Systems*, **28(6)**, June 2017, pp. 1635–1648.

17 Hascoët, J., Desnos, K., Nezan, J.F. and Dinechin, B.D.: "Hierarchical Dataflow Model for efficient programming of clustered manycore processors", *IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, July 2017, pp. 137–142.

18 On, O.J. and Hussin, F.A.B.: "Evaluation and performance analysis of heterogeneous multicore cluster processor architecture", *International Conference on Frontiers of Communications, Networks and Applications (ICFCNA)*, November 2014, pp. 1–6.

19 Kakoee, M.R., Petrovic, V. and Benini, L.: "A multi-banked shared-l1 cache architecture for tightly coupled processor clusters", *International Symposium on System on Chip (SoC)*, October 2012, pp. 1–5.

20 Esmaeilzadeh, H., Blem, E., Amant, R.St., Sankaralingam, K. and Burger, D.: "Dark Silicon and the End of Multicore Scaling", *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, June 2011, pp. 365–376.

21 Borkar, S.: "Thousand core chips: A technology perspective", *IEEE Design Automation Conference (DAC)*, June 2007, pp. 746–749.

22 Binkert, N., Beckmann, B., Black, G., Reinhardt, S.K., Saidi, A., Basu, A., Hestness, J., Hower, D.R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M.D., and Wood, D.A.: "The gem5 Simulator", *ACM SIGARCH Computer Architecture News*, **39(2)**, May 2011, pp. 1–7.

23 Gutierrez, A., Pusdesris, J., Dreslinski, R., Mudge, T., Sudanthi, C., Emmons, C., Hayenga, M. and Paver, N.: "Sources of error in full-system simulation", *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, March 2014, pp. 13–22.

24 Endo, F., Courousse, D. and Charles, H.-P.: "Micro-architectural simulation of in-order and out-of-order arm microprocessors with gem5", *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, July 2014, pp. 266–273.

25 Butko, A., Gamatié, A., Sassatelli, G., Torres, L. and Robert, M.: "Design Exploration for next Generation High-Performance Manycore On-chip Systems: Application to big.LITTLE Architectures", *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2015, pp. 551–556.

26 Butko, A., Garibotti, R., Ost, L., Lapotre, V., Gamatié, A., Sassatelli, G., and Adeniyi-Jones, C.: "A trace-driven approach for fast and accurate simulation of manycore architectures", *Asia and South Pacific Design Automation Conference (ASP-DAC)*, January 2015, pp. 707–712.

27 Uhlig, R.A.: "Trap-driven Memory Simulation", *Ph.D. dissertation*, University of Michigan, 1995.

28 ARM: "Cortex-A series processors - A Technical Reference Manual", December 2017, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.set.cortexa/index.html.

29 Samsung: "Exynos 4 Quad News Roundup", May 2012, http://www.samsung.com/semiconductor/minisite/exynos/newsroom/blog/exynos-4-quad-news-roundup/.

30 Stine, J.E., Castellanos, I., Wood, M., Henson, J., Love, F., Davis, W.R., Franzon, P.D., Bucher, M., Basavarajaiah, S., Oh, J., and Jenkal, R.: "FreePDK: an open-source variation-aware design kit", *IEEE International Conference on Microelectronic Systems Education*, June 2007, pp. 173–174.

31 Kumar, S., Jantsch, A., Soininen, J.-P., Forsell, M., Millberg, M., Öberg, J., Tiensyrjä, K., and Hemani, A.: "A network on chip architecture and design methodology", *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, April 2002, pp. 117–124.

32 Dong, X., Xu, C., Xie, Y., and Jouppi, N.P.: "NVSim: a circuit-level performance, energy, and area model for emerging nonvolatile memory", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **31(7)**, June 2012, pp. 994–1007.

33 Woo, S.C., Ohara, M., Torrie, E., Singh, J.P., and Guptat, A.: "The splash-2 programs: characterization and methodological considerations", *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, June 1995, pp. 24–36.

34 Garibotti, R., Ost, L., Busseuil, R., Kourouma, M., Adeniyi-Jones, C., Sassatelli, G., and Robert, M.: "Simultaneous Multithreading Support in Embedded Distributed Memory MPSoCs", *IEEE Design Automation Conference (DAC)*, June 2013, pp. 1–7.

35 Hu, J. and Marculescu, R.: "Energy-aware mapping for tile-based NoC architectures under performance constraints", *Asia and South Pacific Design Automation Conference (ASP-DAC)*, January 2003, pp. 233–239.

36 Brooks, D.M., Bose, P., Schuster, S.E., Jacobson, H., Kudva, P.N., Buyuktosunoglu, A., Wellman, J., Zyuban, V., Gupta, M., and Cook, P.W.: "Power-aware microarchitecture: design and modeling challenges for next-generation microprocessors", *IEEE Micro*, **20(6)**, Nov/Dec 2000, pp. 26–44.