Evaluation of Heterogeneous Multicore Cluster Architectures Designed for Mobile Computing

David Novo, Alejandro Nocua, Florent Bruguier, Abdoulaye Gamatié, Gilles Sassatelli

Evaluation of Heterogeneous Multicore Cluster Architectures Designed for Mobile Computing

David Novo, Alejandro Nocua, Florent Bruguier, Abdoulaye Gamatie, and Gilles Sassatelli
LIRMM, Université de Montpellier and CNRS
161 rue Ada, 34095 Montpellier, France
Email: first.last@lirmm.fr

***Abstract*—Heterogenous multicore architectures are becoming ubiquitous in high-end smartphones. In this paper, we present a study of two modern heterogeneous multicore architectures based on real measurements. One architecture implements the well-established ARM big.LITTLE 2-cluster approach while the second one takes a leap forward and extends the concept to 3 clusters. In concrete, we analyze the performance of those platforms running parallel workloads. Furthermore, we study the energy and power consumption of the 3-cluster architecture and the effects of high clock frequency on SoC temperature.**

I. Introduction

Traditionally, the continuous increase in processor performance was mostly sustained by technology scaling, which brought higher clock frequencies with every increase in transistor density. However, power density constraints have kept clock frequency fairly constant since 2005 and architects have resorted to multicore architectures to continue increasing processor performance [1].

Initially, multicore architectures included multiple identical cores. More recently, the need for higher energy efficiency motivated the introduction of heterogeneous multicore architectures, which are firmly consolidating as the main gateway to higher energy efficiency in the mobile computing market. Particularly interesting is the concept of single-ISA heterogeneous multicore systems introduced by Kumar et al. [2], which is an attempt to include heterogeneity at the microarchitectural level while preserving a common abstraction (i.e., the ISA or Instruction-Set Architecture) to the software stack. In single-ISA heterogeneous multicore systems, all cores execute the same machine code and thus, any core can execute any piece of the code, which greatly simplifies compilation and run-time management. A body of work built on Kumar's idea to explore the architectural design-space of single-ISA heterogeneous multicore systems [3], [4], [5]. Although very valuable from a methodological perspective, most of this work is based on abstract architectural models that are not always in sync with reality and can lead to erroneous conclusions [6]. Accordingly, we believe that there is a need to complementarily work on the evaluation of modern real hardware to constantly verify the conclusions of such architectural explorations and challenge and improve the used abstract models.

In this paper, we present the evaluation of two single-ISA heterogeneous multicore *System-on-Chip* (SoC) architectures present in the mobile market. The first SoC includes eight cores organized as two heterogeneous clusters and the second SoC includes ten cores and extends the heterogeneity to three clusters. Each of these clusters includes multiple identical cores that share a common L2 cache memory, such as the three clusters illustrated in Figure 1. We first compare the performance of the two SoCs to then continue with a more detailed evaluation of the 3-cluster architecture. The latter includes power consumption, energy consumption, and thermal behavior. Our study shows that the 3-cluster architecture is able to achieve an optimal tradeoff between energy consumption and execution time. Furthermore, we show evidence of the importance of run-time thermal management to keep SoC temperature within reasonable limits.

TABLE I
Specifications of the studied SoCs.

	MediaTek Helio X20 (MT6797)	Samsung Exynos 7 Octa (7420)
Launched	2016 Q1	2015 Q2
CMOS techn.	20 nm (TSMC)	14 nm (Samsung)
Application CPUs	10 (4+4+2)	8 (4+4)
Memory	2 GiB LPDDR2 (up to 14.9 GB/s)	3 GiB LPDDR4 (up to 24.8 GB/s)
OS	Android	Linux
Products incl. SoC	Xiaomi Redmi Note 4	Samsung Galaxy S6

II. Heterogeneous Multicore Architectures

In this section, we introduce the two mobile SoCs selected for our study, namely the MediaTek helio X20 MT6797 [7] and the Samsung Exynos 7 Octa (7420) [8]. Both include ARMv8 heterogeneous multicore CPUs and are referred in the rest of the paper as `HelioX20` and `Exynos7`, respectively. We study these SoCs from a MediaTek X20 development board and a Howchip EXSOM-7420 evaluation platform. Some relevant specifications of the SoCs are shown in Table I.

The `Exynos7` is based on the ARM big.LITTLE technology [9], which instantiates two types of cores. LITTLE cores are designed for maximum power efficiency while big cores are designed to provide maximum compute performance. Interestingly, both types of cores are coherent and share the same ARMv8 ISA. Each cluster is composed of four cores and has its private L2 cache memory. The big cluster features out-of-order ARM Cortex-A57 cores clocked at a maximum frequency of 2.1 GHz. Instead, the LITTLE cluster features

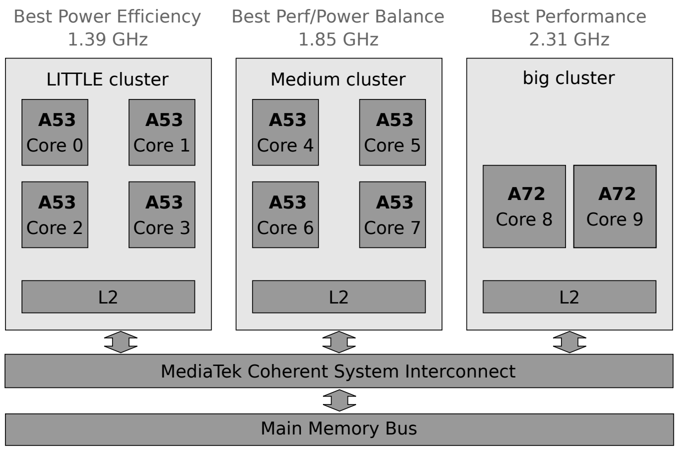

Fig. 1. The Mediatek Helio X20 MT6797 includes a 3-cluster heterogeneous multi-core architecture. One cluster is optimized for best power efficiency, a second cluster for a balance between performance and power efficiency and a last cluster for best performance.

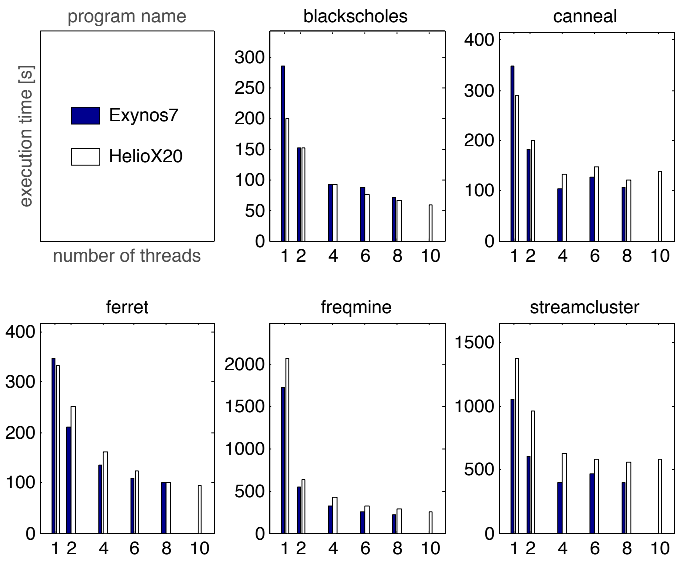

Fig. 2. Execution time of the Region of Interest of the selected PARSEC programs with `native` input set when varying the number of threads.

in-order ARM Cortex-A53 cores clocked at a maximum frequency of 1.5 GHz.

The `HelioX20` takes a leap forward in core heterogeneity and includes the 3-cluster architecture shown in Figure 1. One cluster is optimized for best performance, which includes two out-of-order ARM Cortex-A72 cores clocked at a maximum frequency of 2.31 GHz. A second cluster is optimized for power efficiency, which includes four in-order ARM Cortex-A53 cores clocked at a maximum frequency of 1.39 GHz. Lastly, a third cluster is optimized for a good performance/power balance, which also includes four ARM Cortex-A53 cores but able to reach a higher maximum frequency of 1.85 GHz. In an attempt to maintain a certain consistency between the two SoCs, we have renamed the `HelioX20` clusters as big.Medium.LITTLE.

III. Performance Evaluation

In this section, we evaluate the performance of the selected SoCs running five programs of the PARSEC 3.0 benchmark suite [10]. The benchmark originally includes 13 programs, but only 11 were successfully compiled and executed on the `Exynos7`, which runs Linux. However, the `HelioX20` does not support Linux but Android. As a result, only 5 out the 11 programs could be compiled (by setting the static flag) and successfully executed on Android.

All the programs in the benchmark suite support multi-threaded parallelization and follow a data-parallel or pipeline parallelization strategy. The user can control the number of threads spawned with a parameter passed to the binary. `blackscholes`, `canneal`, `ferret`, and `streamcluster` are parallelized using `pthreads` while `freqmine` is parallelized using `OpenMP`, which has a much higher level of abstraction.

We first present the performance evaluation of the entire SoC to then analyze the performance differences of the individual cores.

A. SoC Performance

The `HelioX20` and `Exynos7` run the Linux kernel v3.18.22 and v3.10.61, respectively. Note that Android includes a Linux kernel. Accordingly, both kernels use the *Native POSIX Thread Library (NPTL)* v2.21. The NPTL is a so-called 1×1 threads library, in that threads created by the user via the `pthread_create()` library function (also known as user-level threads) are in 1 to 1 correspondence with schedulable entities in the kernel (also known as kernel-level threads). Furthermore, our Linux kernel includes the extensions implemented by the Linaro group to implement multi-core load balancing in ARM big.LITTLE architectures. Concretely, we make use of the *Global Task Scheduler (GTS)*, which has the ability to use all cores simultaneously and it is designed to improve peak performance and to include finer-grained control of workloads that are migrated between cores, reducing kernel overhead. In the `Exynos7`, the GTS scheduler creates a list of big and LITTLE cores that is used to pick the target core for a particular task. Then, using runnable load average statistics, the modified Linux scheduler tracks the average load of each task and migrates tasks to the best core. High-intensity tasks (i.e., tasks with a high average load) are migrated to the big core(s) and are also marked as high-intensity tasks for more efficient future allocations. Low-intensity tasks remain resident on the LITTLE core(s). The GTS scheduler is not limited to two clusters and in the `HelioX20` the scheduler is configured to work with three clusters. In this case, the LITTLE and Medium cores are used to run the low and medium intensity tasks, respectively, and the big cores are only used to deliver the instantaneous peak performance demanded by the computing-intensive or time-responsive tasks [11].

Figure 2 shows the execution time of the different programs when varying the number of generated threads in both the

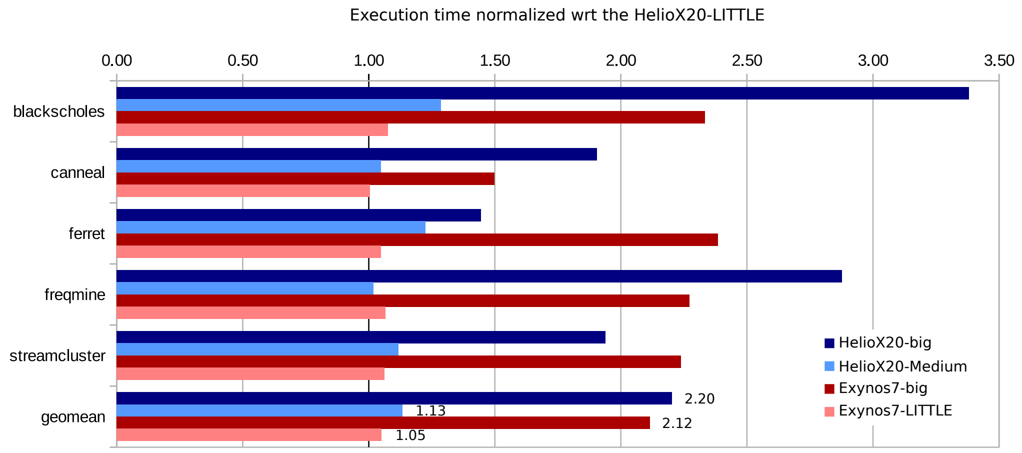

Fig. 3. Core performance relative to the HelioX20 LITTLE core. The execution times include the Region of Interest of the simlarge input set.

Exynos7 and the HelioX20 SoCs. The reported execution times correspond to the *Region of Interest (ROI)* of the native input set—the largest available input set. In general, both SoCs follow a similar trend and more threads running on parallel generally lead to faster executions times. For the single-threaded execution, the HelioX20 is faster in 3 out of the 5 programs. However, for the 2-threads execution, the Exynos7 catches up and achieves a faster execution in all the programs. This may be due to cluster management differences between the two SoCs. Furthermore, we observe that programs like blackscholes or freqmine are still able to achieve a speedup in the 8 to 10 threads transition while others such as canneal or streamcluster start to saturate after 4 threads.

B. Cluster Management

By default, the Exynos7 has all the cores online while the HelioX20 has most of the cores offline and wakes them up on demand. The Mediatek core manager, known as hps or hot-plug strategy, continuously adapts the online cores to the computation load, which makes it very hard to monitor the thread-to-core pinning of a particular execution.

Besides, the Exynos7 uses the interactive frequency governor for all its CPUs, while the HelioX20 uses interactive for the slow A57 cores (i.e., the LITTLE cluster) and the A72 cores (i.e., the big cluster), and ondemand for the fast A57 cores (i.e., the Medium cluster). The ondemand governor is commonly chosen by smartphone manufacturers because it is well-tested, reliable, and virtually guarantees the smoothest possible performance for the phone. This governor scales its clock speed in a work queue context. Accordingly, once the task that triggered the clock speed ramp is finished, the clock speed will be moved back to the minimum. However, If the user executes another task that triggers a new ramp, the clock speed will bounce from minimum to maximum, which can lead to negative effects on battery life. Instead, the interactive governor scales the clock speed over the course of a timer set arbitrarily by the kernel developer. As a result, if an application demands a ramp to maximum clock speed (by placing 100% load on the CPU), a user can still execute another task before the governor starts reducing CPU frequency. This can eliminate the frequency bouncing discussed earlier. Furthermore, this governor is better prepared to utilize intermediate clock speeds that range between the minimum and maximum CPU frequencies thanks to its timer-based approach.

The differences in core and frequency management between the two SoCs makes it difficult to analyze in further detail the performance of the two architectures. Accordingly, we disable those mechanisms and study the individual core performance in the following subsection.

C. Individual Core Performance

To gain control, we override the default configurations of the two SoCs. In the Exynos7, we set the frequency governor to userspace. Thereby, we can control from the user space the frequency of each core by modifying the scaling_setspeed file. Fore example, the following command:

```
$ echo 220000 > /sys/devices/system/cpu/cpu0/
cpufreq/scaling_setspeed
```

sets core number zero to 220 MHz. In the HelioX20, we first disable the hps with the following command:

```
$ echo 0 > /proc/hps/enabled
```

Once hps is disabled, we can can set the frequency governor to userspace and wake up the individual cores (e.g., core number zero) with the following command:

```
$ echo 1 >  /sys/devices/system/cpu/cpu0/online
```

Accordingly, we wake up all the cores of the HelioX20. Furthermore, we set all the cores in both SoCs to their maximum clock frequency. We then use taskset to pin the program to the desire core and execute single-threaded PARSEC programs on each core type. Figure 3 shows the resulting execution times normalized with respect to the HelioX20 LITTLE cores.

The HelioX20 big cores are ARM Cortex-A72, which is a more modern out-of-order architecture than the ARM Cortex-A57 of the Exynos7 big cores. Furthermore, the HelioX20 big cores are clocked at a maximum frequency of 2.31 GHz while the Exynos7 big cores can only reach 2.1 GHz. Thus, one should expect the HelioX20 big cores to outperform the Exynos7 big cores, which is what the geomean shows in Figure 3: the HelioX20 big cores are 2.20 times faster than the HelioX20 LITTLE cores, while the Exynos7 big cores are 2.12 times faster. However, this is not the case for all the programs: the Exynos7 big cores are significantly faster in the execution of streamcluster and ferret. The reason being that, despite the superiority of the HelioX20 big core architecture, the Exynos7 SoC includes a much faster main memory (24.8 GB/s vs 14.9 GB/s), which leads to faster executions of memory bounded programs.

All the remaining cores have the same in-order ARM Cortex-A53 architecture and only differ in their maximum clock frequency: 1.39 GHz in the HelioX20 LITTLE core, 1.5 GHz (+7%) in the Exynos7 LITTLE core, and 1.85 GHz (+32%) in the HelioX20 Medium core. In average, the Exynos7 LITTLE cores and the HelioX20 Medium cores are 5% and 13% faster than the HelioX20 LITTLE cores.

IV. Energy Consumption and Thermal Evaluation

In this section, we concentrate our evaluation on the HelioX20 SoC, as we consider its increased heterogeneity, which goes one step beyond the traditional big.LITTLE architectures, of greater interest to study energy consumption and thermal behavior.

We use two OpenMP programs of the Rodina benchmark suite [12], namely heartwall and lud. We disable the hps as shown in the previous section and pin the OpenMP threads to cores with the GOMP_CPU_AFFINITY environment variable. We also select the dynamic OpenMP scheduler with the OMP_SCHEDULE environment variable. Note that a static OpenMP scheduler will be highly inefficient in our heterogeneous multicore cluster architectures: since the work is distributed evenly between all threads, the big cores will finish their work faster and need to wait idly for the slower LITTLE cores to finish their work. Instead, the dynamic scheduler distributes dynamically the work between threads so that the threads pinned to faster cores will be able to do more work.

A. Measuring Power and Temperature

We use the Agilent Technologies (now KEYSIGHT) N6705B DC Power Analyzer to measure the current consumption of the board. We configure the power analyzer in *Current Measure* and connect it in series between the board and its power supply. The power supply gives 12 Volts and limits the current to 2 Amperes, which results in a maximum power delivery of 24 Watts. We configure the power analyzer to capture 100 samples per second. The captured current is then multiplied by the 12 Volts of the supply voltage to obtain

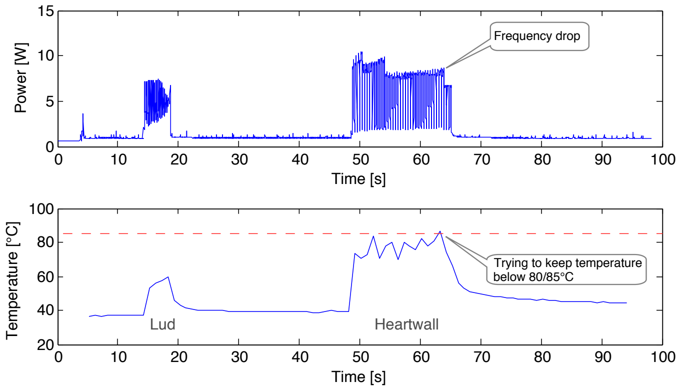

Fig. 4. Power consumption (top) and SoC temperature (bottom) of the execution of lud and heartwall on the HelioX20 SoC.

power. The top graph of Figure 4 shows the power consumption measured on the HelioX20 SoC while executing lud and heartwall with ten OpenMP threads, each pinned to a different core.

We measure the SoC temperature by using a file interface provided by the Linux kernel. The following command prints the current temperature of the SoC:

```
$ cat /sys/class/thermal/thermal_zone1/temp
```

Accordingly, we wrote a lightweight script that samples the SoC temperature every second. The bottom graph of Figure 4 shows the temperature evolution of the HelioX20 SoC while executing lud and heartwall. We observe that the execution of lud triggers a lower power consumption than the execution of heartwall. As a result, the SoC temperature barely reaches 60°C during a lud execution while it rapidly grows up to 85°C during a heartwall execution. Thus, we conclude that the heartwall program triggers a more intensive use of the SoC microarchitecture and becomes a good example application to study the limits of such a heterogeneous SoC in terms of power consumption and temperature.

B. Energy-Performance tradeoffs

In this subsection, we study the tradeoffs between energy consumption and performance enabled by a 3-cluster heterogeneous multicore architecture.

Energy-Performance Pareto. In a first experiment, we execute the two Rodinia programs under the following cluster configurations:

- L: Only the LITTLE cluster is active (4 threads).
- M: Only the Medium cluster is active (4 threads).
- B: Only the big cluster is active (2 threads).
- B+M: The big and Medium clusters are active (6 threads).
- B+L: The big and LITTLE clusters are active (6 threads).
- M+L: The Medium and LITTLE clusters are active (8 threads).
- B+M+L: All clusters are active (10 threads).

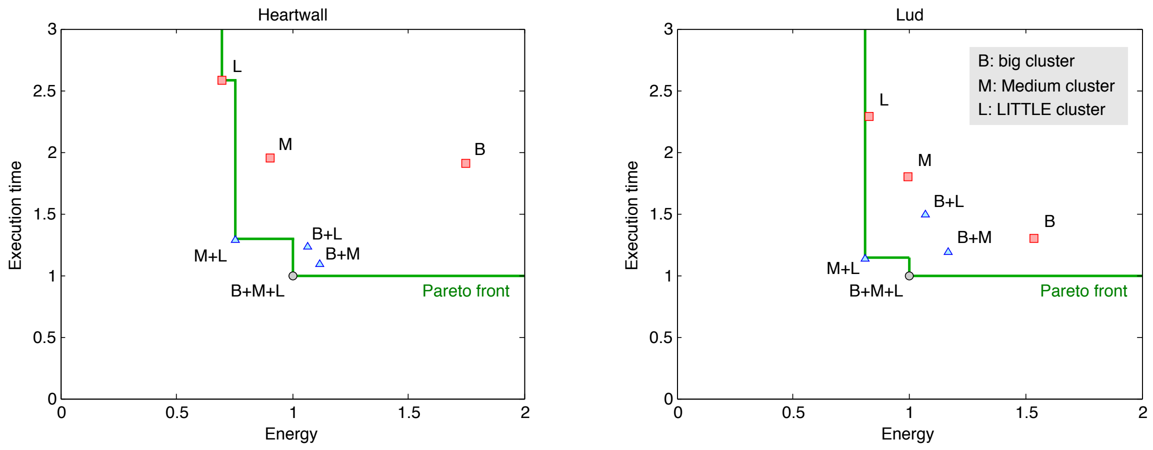

Fig. 5. Execution time vs energy consumption Pareto plot of running `heartwall` (left) and `lud` (right) on different `HelioX20` cluster configurations.

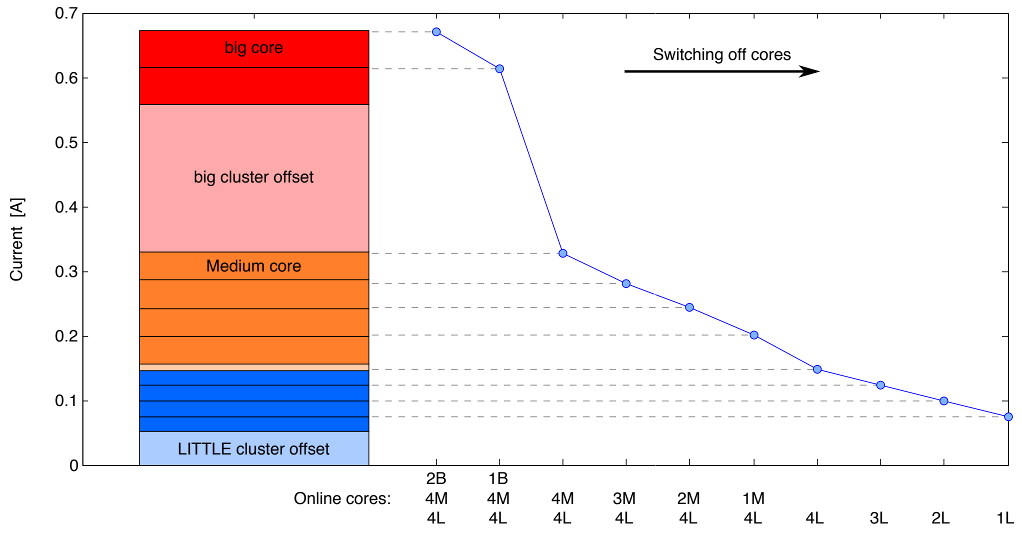

Fig. 6. Average current consumption of executing `heartwall` as function of the online `HelioX20` cores. A 2B4M4L configuration indicates that two cores in the big cluster, 4 cores in the Medium cluster and 4 cores in the LITTLE cluster are online.

The execution of `lud` (right) and `heartwall` (left) under the different cluster configurations is projected on the execution time vs energy consumption Pareto space in Figure 5. Using only the big cluster for these type of parallel workloads is highly suboptimal as it is the most energy-hungry configuration and only the 4th or 5th fastest configuration. Using only the LITTLE cluster is the most energy efficient configuration for running `heartwall` and the second most energy efficient configuration for running `lud`. However, in both cases, their execution times are around 2.5 times slower than those of the fastest configurations. Using only the Medium cluster achieves the same performance as using the big cluster for running `heartwall` but consumes about half of the energy. Instead, `lud` runs sensibly faster using only the Medium cluster than using only the LITTLE cluster and consumes significantly less energy than using only the big cluster.

Considering the use of two clusters, the use of the Medium and the LITTLE cluster is always a Pareto optimal configuration. The reminding two-cluster configurations, namely big+Medium and big+LITTLE, are always Pareto dominated by the superior three-cluster configuration, which is both faster and more energy efficient. Finally, we observe that using the three clusters is faster but slightly more energy-hungry than the best use of two clusters (i.e., M+L).

Switching off cores. In this experiment, we start with the execution of `heartwall` on the `HelioX20` with all the cores online and study the effect in average current consumption of switching off the fastest cores one at a time. Figure 6 illustrates the reduction in average current as a function of the number and type of online cores. Running with all the cores switched on consumes 670 mA. Switching off one of the big cores saves 60 mA, which is just 9% of the original consumption. Instead, if we switch off the two big cores (i.e., the entire big cluster) the current consumption drops to 330 mA, which corresponds to 48% of the original consumption. Switching off a Medium core saves 48 mA, which is 7% of the original consumption. Note that the graph shows a remarkable linear behavior as Medium cores are switched off. After switching off the Medium cluster, the current consumption drops to 150 mA, which corresponds to 22% of the original consumption. Switching off a LITTLE core saves 24 mA, which is 4% of the original consumption. Having only one LITTLE core online consumes 75 mA, which is about 9 times less than the original current consumption.

Based on the experiment, we can conclude that the use of the big cluster is considerably expensive in terms of current/power consumption. Accordingly, it will rarely lead to higher energy efficiency as not many applications will achieve the speed up necessary to compensate for the extra power consumption (i.e., more than a two times speed up). Furthermore, we can see that the `HelioX20` has been designed to provide excellent power scalability, with a factor 9 between the least and the most performing configuration. We have also measured the standby current (i.e., current consumed when no application is executed) of the different configurations: 80 mA when the three clusters are online, 71 mA when only the big cluster is online, 58 mA when only the Medium cluster is online, and 48 mA when only the LITTLE cluster is online. Thus, the LITTLE cluster standby current consumption is 14 times lower than that of running `heartwall` on all the cores.

But power scalability does not necessarily translate into energy scalability. Figure 7 shows the energy consumption vs execution time of the configurations presented in Figure 6. The configuration with the lowest current consumption turns out to be 11.5 times slower than the fastest configuration and thus becomes the more energy-hungry configuration. The latter is two times more energy-hungry than the most energy efficiency configuration. Based on these results, we can conclude that data parallel applications run more efficiently in clusters having all their cores online. This is particularly evident in the big and the LITTLE cluster, in which switching off cores only leads to slower and more energy-hungry configurations.

Finally, we use the Linux `perf` tool to gain access to the core performance counters and study the utilization factor of each core. In concrete, we read the number of cycles that each core is active and divide that value by the execution time. The latter results in the effective frequency of operation of that core and the ratio between this effective frequency and the actual core frequency gives the core utilization factor. Based on that metric, we can assess that the LITTLE cluster configuration (i.e., 4L) achieves a utilization factor of 91% while the three-cluster configuration (i.e., 2B4M4L) achieves a utilization of 53% for the big cores, 57% for the Medium cores and 89% for the LITTLE cores. This indicates that, despite using the `dynamic OpenMP` scheduling, the fastest cores still spend a sizable amount of time idle.

C. Thermal Evaluation

In this subsection, we study how the cluster configuration affects the `HelioX20` SoC temperature. The execution of `heartwall` shown in Figure 4 starts with the big cores configured at their maximum frequency of 2.31GHz, however, the frequency drops to 1.5 GHz at the end of the execution.

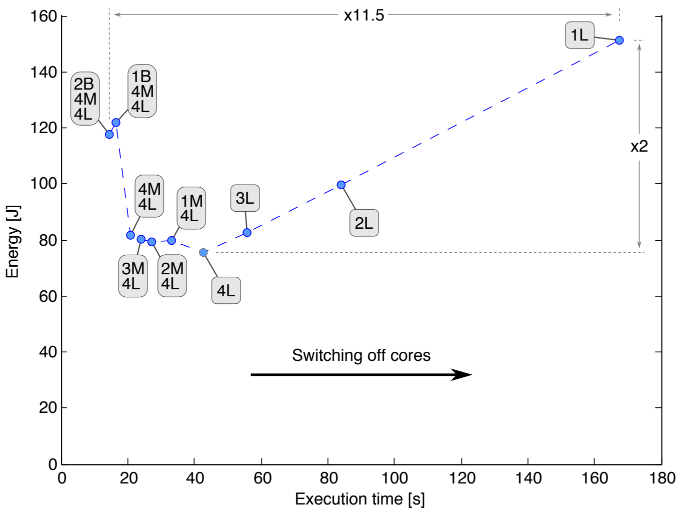

Fig. 7. Energy vs execution time of executing `heartwall` as function of the online `HelioX20` cores.

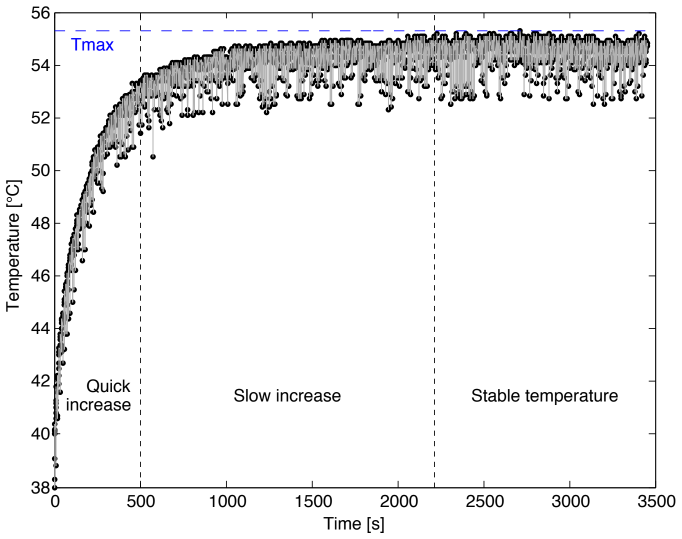

Fig. 8. Heating up process of a Medium core running `heartwall` at 1.09 GHz. A T_{max} of 55°C is reached after more than 2000 seconds.

As one can see in the bottom graph, a thermal supervisor is overruling our setting of maximum frequency to keep SoC temperature below a thermal threshold that seems to be set at around 80–85°C. Accordingly, thermal management is becoming paramount as mobile platforms keep on using increasingly powerful cores.

To better understand the heating process and how this is influenced by the core clock frequency, we run multiple consecutive `heartwall` executions on a Medium core configured to an intermediate clock frequency of 1.09 GHz. Figure 8 shows the resulting temperature evolution. We identify three different phases: (1) quick temperature increase, (2) slow temperature increase, and (3) stable temperature of T_{max}. The latter is a function of the application, the core architecture, the core frequency, the type of heat spreader (see the picture in Figure 11) and the ambient temperature.

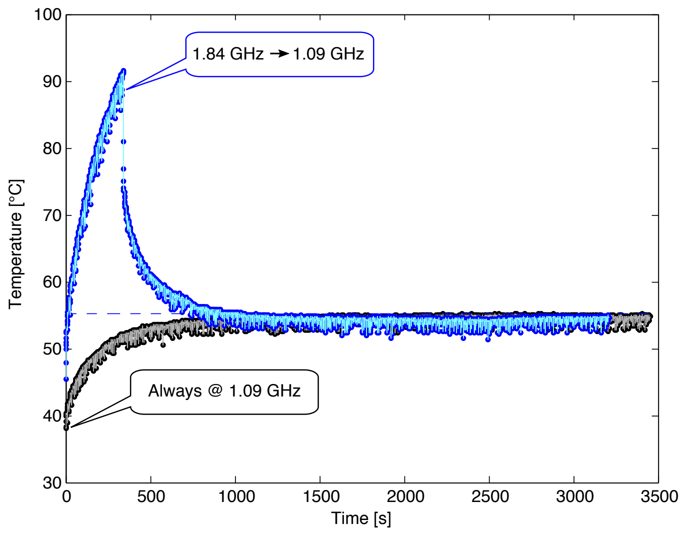

Fig. 9. Medium core switches clock frequency from 1.84 GHz to 1.09 GHz while executing several instances of `heartwall`. Immediately after reducing the frequency, the SoC temperature quickly descends until meeting the T_{max} of the second frequency.

We overrule the default thermal configuration and set the new threshold to 120°C by running the following command:

```
$ thermal_manager /etc/.tp/.ht120.mtc
```

Then, we repeat the same experiment but starting now at the maximum clock frequency of the Medium core (i.e., 1.84 GHz) to change back to 1.09 GHz after 340 seconds. Figure 9 shows how the temperature quickly shoots up to more than 90°C while running at maximum frequency. Once the frequency is reduced, the temperature quickly decreases and slowly reaches the T_{max} of the final frequency. The figure also shows that the execution starting with the maximum frequency finishes executing all the `heartwall` instances faster than the run at a constant frequency. That acceleration comes at the expense of a temperature increase and higher instantaneous power consumption. While the latter does not necessarily translate into a higher energy consumption, provided that the achieved acceleration is large enough, the temperature increase can be fatal: excessive temperature may harm users and even damage permanently the SoC.

Figure 10 extends the experiment to include the thermal behavior of the LITTLE cores. We observe that the LITTLE cores are more thermal friendly than the Medium cores and remain well below 60°C when running at their maximum frequency (i.e., 1.39 GHz). Interestingly, we can see the thermal difference between the Medium and LITTLE cores when running both at 1.09 GHz. Since they both have the same core architecture and frequency, the execution time is exactly the same. However, the Medium core reaches a T_{max} of 53°C while the LITTLE core remains below 45°C. When running at the minimum frequency (i.e., 220 MHz), the LITTLE core remains below 35°C but requires more than 2900 seconds to complete the execution.

Considering that the threshold for thermal throttling (i.e., reduction of frequency to avoid overheating) is set around 80°C by default, we sweep different clock frequencies of the Medium core to find the maximum frequency that can be used without overheating above the 80°C. Figure 11 shows the T_{max} of different frequencies of the Medium core running `heartwall`. To avoid thermal throttling, the frequency should not exceed the 1.5 GHz. Note that the maximum frequency of a LITTLE core is 1.39 GHz, so the frequencies that lead to thermal throttling in Medium cores are basically the ones making those cores superior in performance. But thermal throttling will only be necessary after several minutes of continuous intense processing. For instance, a Medium core can process `heartwall` at its maximum frequency during 190 seconds before overheating above the 80°C.

The big cores are particularly susceptible to thermal throttling due to their more complex architecture (e.g., deeper

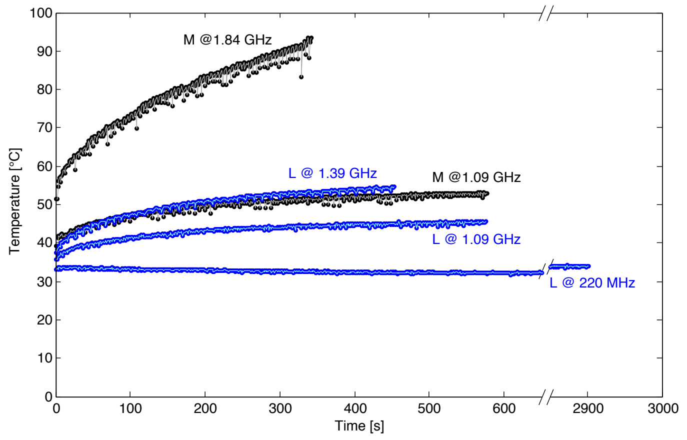

Fig. 10. Thermal behaviour of LITTLE (L) and Medium (M) cores running the same `heartwall` load at different frequencies. When running at 1.09 GHz, a Medium core heats up 8°C more than a LITTLE core.

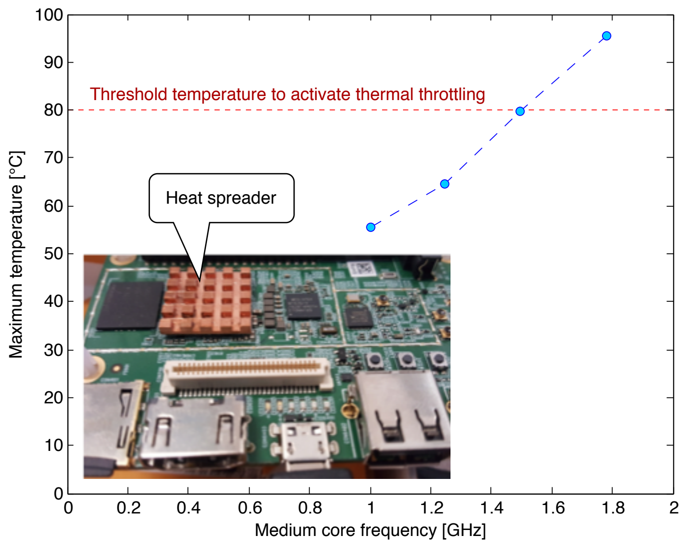

Fig. 11. Variation of the T_{max} reached by a Medium core while running `heartwall` as a function of the clock frequency. Thermal throttling will be needed for frequencies above 1.5 GHz.

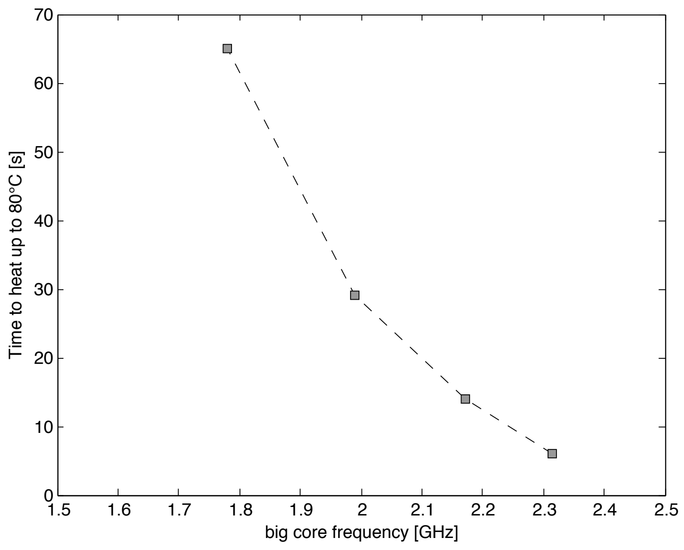

Fig. 12. Time that a big core can sustain `heartwall` processing before overheating above 80°C. At its maximum frequency (i.e., 2.31 GHz), a big core can only run 6 seconds before activating thermal throttling.

pipeline, higher parallelism, etc.) and higher frequencies. Figure 12 shows the time that a big core can sustain `heartwall` processing before overheating above 80°C depending on the core frequency. At its maximum frequency, a big core can only run 6 seconds before activating thermal throttling.

V. Conclusions

In this paper, we have presented a study of two modern heterogeneous multicore architectures based on real measurements. We have compared the performance of a typical 8-cores big.LITTLE `Exynos7` architecture with a more exotic 10-core `HelioX20` architecture that extends the concept of heterogeneous clusters from two to three.

Performance-wise, we do not see an advantage in increasing the number of cores from 8 to 10. On the contrary, the `HelioX20` is faster in 3 out of the 5 programs in the single-threaded execution. However, for the 2-threads execution, the `Exynos7` catches up and achieves a faster execution in all the programs. When studying the individual cores, the superior A72 big cores of the `HelioX20` outperform the A57 big cores of the `Exynos7` for the compute intensive programs. However, the trend is reversed in memory intensive workloads that take advantage of the superior LPDDR4 of the `Exynos7` SoC, which is almost two times faster than the `HelioX20` LPDDR2.

We have further analyzed the energy and power consumption of the `HelioX20` 3-cluster architecture. We have found that running on the 3 clusters is a Pareto optimal use of the architecture in the energy consumption vs execution time design space. Running on the Medium and LITTLE cluster is also Pareto optimal, saving in energy consumption at the expense of a longer execution time. We have also found a factor 9 power scalability between the fastest (i.e., all cores online) and slowest (i.e., only on LITTLE core online) core configuration. However, this remarkable power scalability only translates to a limited energy proportionality when running a parallel program. For example, switching off the big cluster during the execution of `heartwall` can save 25% of the energy at the expense of 30% longer execution time.

The high-performance configurations of the `HelioX20` consume around 8 Watts of power, which critically affects SoC temperature. Our experiments show that a Medium core will activate thermal throttling when running at frequencies higher than 1.5 GHz during a certain time. For example, thermal throttling will be activated after running the Medium core at its maximum frequency for a period of 190 seconds. The big cores are even more sensitive to overheating and will activate thermal throttling after only 6 seconds of processing at its maximum frequency. Accordingly, we conclude that the big cores of such an architecture are not meant to support sustained processing but rather to assist processing in sporadic cases.

VI. Acknowledgement

The research leading to these results has received funding from the H2020 RIA Framework under the Mont-Blanc 3 project: http://www.montblanc-project.eu, grant agreement No. 671697.

References

[1] M. J. Flynn and P. Hung, "Microprocessor design issues: thoughts on the road ahead," *IEEE Micro*, vol. 25, no. 3, pp. 16–31, 2005.

[2] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-ISA heterogeneous multi-core architectures for multithreaded workload performance," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2004, pp. 64–75.

[3] R. Kumar, D. M. Tullsen, and N. P. Jouppi, "Core architecture optimization for heterogeneous chip multiprocessors," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2006, pp. 23–32.

[4] J. C. Mogul, J. Mudigonda, N. Binkert, P. Ranganathan, and V. Talwar, "Using asymmetric single-ISA CMPs to save energy on operating systems," *IEEE Micro*, vol. 28, no. 3, 2008.

[5] K. Van Craeynest and L. Eeckhout, "Understanding fundamental design choices in single-ISA heterogeneous multicore architectures," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 9, no. 4, p. 32, 2013.

[6] S. L. Xi, H. Jacobson, P. Bose, G.-Y. Wei, and D. Brooks, "Quantifying sources of error in McPAT and potential impacts on architectural studies," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 577–589.

[7] MediaTek, "MediaTek helio X20," https://www.mediatek.com/products/smartphones/mt6797-helio-x20/, [Accessed: May-2018].

[8] Samsung, "Exynos 7 Octa (7420)," http://www.samsung.com/semiconductor/minisite/Exynos/Solution/MobileProcessor/Exynos_7_Octa_7420.html, [Accessed: May-2018].

[9] ARM, "big.LITTLE," https://developer.arm.com/technologies/big-little, [Accessed: May-2018].

[10] Princeton University, "PARSEC 3.0," http://parsec.cs.princeton.edu/parsec3-doc.htm, [Accessed: May-2018].

[11] MediaTek, "CorePilot 3.0. Max.Mid.Min (Tri-Cluster) technology to maximize power efficiency with extreme computing performance," MediaTek White paper, Tech. Rep., 2015.

[12] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, 2009, pp. 44–54.