Heterogeneous Micro-architectural adaptation of Codelets using gem5 and CERE

Nicolas Derumigny

ENS de Lyon

Pablo de Oliveira Castro

Université de Versailles Saint-Quentin-en-Yvelines

Abstract

In recent years, many efforts have been deployed in order to increase the speed and efficiency of computing architectures. Another step to increase their efficiency is to adapt the architectures to the programs they run, this idea is called *codesign*. For example, neuronal networks do not require the full 64 precision bits of recent architectures, and can operate well with only 16 bits. Therefore specific hardware, faster for 16-bit calculus is being developed.

Heterogeneous multicore systems such as the ARM big.LITTLE technology, widely spread on embedded computing, use two different CPUs: the big CPU handles heavy tasks, and the little one, more power-efficient but slower, is used the rest of the time. Sometimes the different regions of an application fit better on some architecture, depending of the type of data and the operations they are dealing with. On the big.LITTLE architecture, they can be dynamically migrated from one CPU to the other.

The codelet approach, based on the use of the software Codelet Extractor and REplayer (CERE), allows to isolate and tune sections of the application individually, showing which improvements on the hardware side leads to better performance.

This report studies the possibility of automatically adapting programs to heterogeneous micro-architectures. It considers programs written in the x86 instruction set and simulated by gem5. The fit of a code region to a particular CPU is measured through a performance-consumption ratio, calculated with McPAT. Experiments have been performed on both sequential and parallel applications on one simulated monocore CPU, using the NAS and PARSEC benchmark suites.

1 Introduction

1.1 Generalities

Developing a new CPU architectures is a compromise between several parameters. The common approach is to minimise both the manufacturing cost and the average execution time, measured on a suite of benchmark representative of the user's most frequent tasks. But in High Performance Computing (HPC), the computational requirements can change a lot depending on the application: for physicist calculus, a great precision is needed, so developing 128-bit compute units is important. Whereas in deep learning, fast 16-bit computation is enough.

Thus, architectural adaptations can greatly improve the performance of computational unit executing only a fixed task. For instance, the boost technology, increasing the frequency during high loads while the chip is under a given temperature (or does not reach a thermal threshold) can also be taken as an on-the-fly tuning of some chips. Another example of this specialisation is GPGPUs and CPUs: GPGPU are efficient in highly-scalable parallel applications, and CPUs are faster on sequential ones.

CERE is a software that decomposes an application into a set of pieces of codes, called codelets, corresponding to a the important program regions. Extracted codelets present themselves as another application that can be run out-of-the-box and recreate the behaviour of the original selected region.

The gem5 simulator[3] is used to quantify micro-architecture impacts on softwares: it allows to fine-tune parameters without using different processors. As it emulates a whole linux system, the measurement can be really slow, that is why running only codelets gives a serious advantage over measuring the entire application. For example, running the NAS Integer Sort (IS) codelet is four times faster in SE mode that running the full application.

1.2 Experimental Protocol

Four x86 CPUs are considered, one based on the Cortex A-15[6], the i5-3550 (with turbo and non-turbo frequency), the i5-3770U, a low-power mobile CPU and a Q9100. All these CPUs are simulated as one-core CPU, using the one-core values for non-shared caches and real value for shared caches.

The codelets used were extracted using CERE[5] from the NAS[1] sequential benchmark suite and the PARSEC[2] benchmark suite. We chose NAS IS sequential as a simple serial application sorting integers and pthread-disabled x264 for a more complicated one demonstating state-of-the-art video compression. Blackscholes and Freqmine, two OpenMP applications from PARSEC suite, were chosen to test multithread performance.

The energy consumption was computed using McPAT[7] and taken as a measure of the efficiency of each simulated CPU. Indeed, power consumption is a crucial measurement for architecture tuning, as it limits on one side the maximum computational power of the CPU (because higher frequencies increase the power consumption) and on the other side the power cost to run it.

Section 2 presents the related work and situates the paper contributions in the current research. Section 3 describes how codelets extracted by CERE were ported to be compatible with the gem5 simulator, along with the performance and energy models and the values and the applications chosen for the simulations. Section 4 demonstrates how CERE coupled with gem5 can be used to quickly tune heterogeneus architecture for each codelet.

2 Background

2.1 On the use of codelets

Previous studies [5] show that source code isolation is a reliable way to reproduce the behavior of applications.

CERE is a software that creates *codelets*, small parts of an application that can be run separately, from one loop or OpenMP parallel region of the application. The loop run inside the application is called the *in-vivo* codelet. CERE creates a copy of the original application running only this loop and recreates its execution (cache state especially should be as similar as possible). This process is called *extraction* and leads to the *in-vitro codelet*. The codelet running time is the time taken by the loop to be executed; it does not include the memory restore stage and the cache warm-up sequence. The complete sequence of codelet extraction is illustrated on figure 1.

CERE extracts codelets from a C/C++/Fortran application using the LLVM compiler. It operates at the Itermediare Represemtation (IR) level, and thus is more flexible than source-code isolation (isolating the codelet at the source-code level, then recompiling it) or assembly isolation (isolating the codelet at the binary file level)[5].

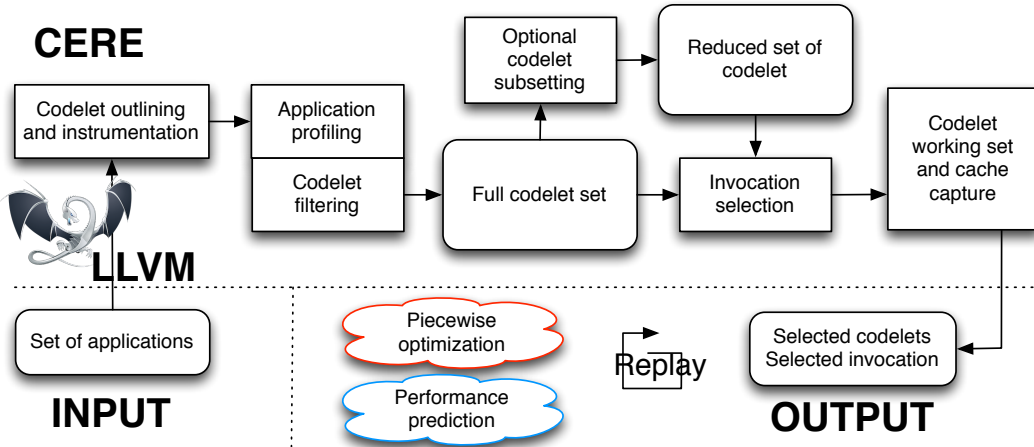At this time, CERE targets only loops and openMP parallel *for* loops.

Figure 1: Functional scheme of CERE (LLVM logo courtesy of Apple Inc.)

On sequential NAS IS benchmark, the selected codelet covers more than 98% of the total execution time, but is 7.3 to 46.6 times faster than running the full NAS.B suite.

CERE captures the memory context at a page-granularity level, which is lighter than a full dump and then faster to replace in memory when replaying inside a simulator. The memory capture allows to restore the codelet working set and cache-state before it is replayed.

2.2 On the simulators

The gem5 simulator

The gem5 simulator is a cycle-accurate simulator. Its accuracy has been demonstrated on ARM simulation on both in-order and out-of-order processor, comparing real and simulated Cortex-A8 and Cortex-A9[6]. These comparisons reveal an average absolute error of only 7%.

Replaying SPLASH benchmark on gem5 shows an error on the execution time from 1.39% to 17.94%, explained by an inaccurate simulation of the DDR memory. Nevertheless, the gem5 simulator now handles different memory types, including modern DDR3, DDR4 and GDDR5, which should be more accurate than the tested DDR memory used in SPLASH tests[4].

The McPAT simulator

McPAT is the first integrated power, area, and timing simulator. It can emulate both multicore and many-core systems with various technology, including in-order and out-of-order cores, caches, network chips, and several manufacturing processes. It can be used with several simulators thanks to its XML-interface (detailed in figure 2). McPAT computes both static (average power consumption of the CPU) and dynamic power consumption (peak power used when executing a specific task). We use dynamic power consumption in this paper, as we aim at finding the codelet with the highest performance/power ratio. McPAT can also compute the die size of a simulated CPU, but this functionality has not been used in this paper.

In previous studies, McPAT accuracy has been validated against four real CPUs[7] and shows a relative error from 10.84% to 22, 61%. The simulated power is always under-evaluated, as some components present in the CPU are sometimes not taken into account. Similarly, the computed die size differs up to 27.3% from the real processor.

3 Simulation framework

3.1 The gem5 simulator

The gem5 simulator can be run in two different modes: syscall emulation (SE) and fullsystem mode (FS). The Syscall emulation mode simulates only the behaviour of the CPU inside a Linux operating system, and therefore cannot efficiently simulate multi-threaded application, as no scheduler has been implemented. Besides, SE mode requires a static linkage of all the application libraries.

On the contrary, the FullSystem (FS) mode emulates a full CPU; as the OS is emulated, the simulation is really slow (about fifteen minutes to boot Linux on an x86 AtomicSimpleCPU). Nevertheless, FS mode is more accurate and more flexible.
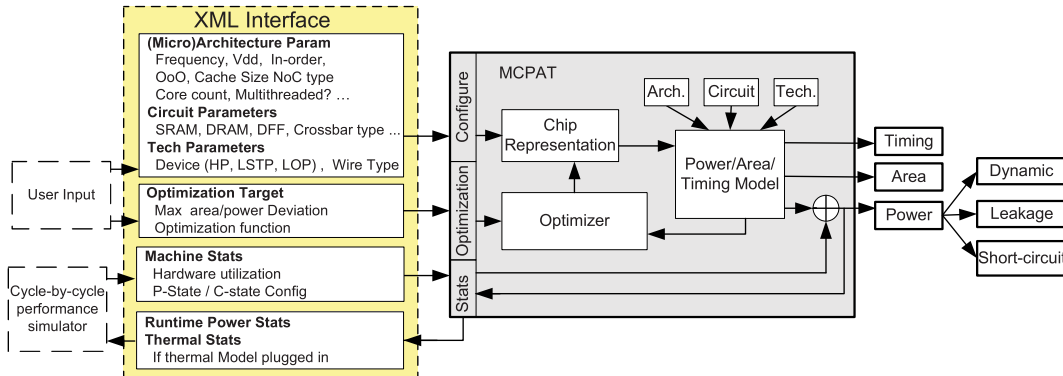

Figure 2: Functional scheme of McPAT

Indeed, FS mode can handle dynamic libraries, assuming that they are well installed in the virtual disk image. Moreover, gem5 features a checkpoint functionality which avoids booting again when the CPU is changed.
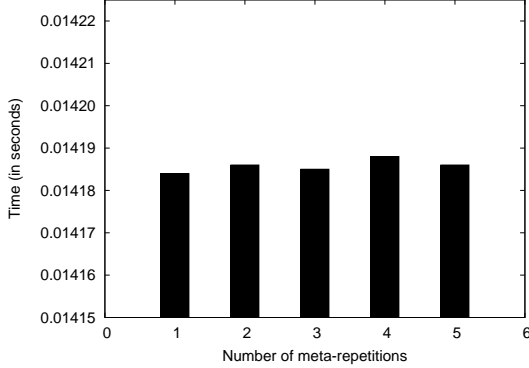

Figure 3: Variations of the codelet loop execution time on NAS IS benchmark using a quad-core i5-3550 configuration.

3.1.1 Syscall emulation mode

Gem5 required two code patches before CERE sequential codelets could be run in SE mode. First, the *getdents* syscall has been implemented (cf 6.3.1), which is called inside the *readdir* function, used in the codelet memory mapping function. The second change concern a bug occurring when reading EOF with the syscall *read* while providing an invalid pointer (cf 6.3.2): according to POSIX specifications it should work and write nothing, but caused a page fault in gem5. Both patches were submitted to the gem5 community.

In order to statically compile codelets with CERE[1], the argument *-static* must be provided at the link time. Noting that this could provoked some relocation error when specifying the entry point of the program: offset 0x60000000 (used as default start point in CERE) is used to place the standard C library in sequential NAS IS. Experimentally, offset 0x40000000 did not cause any trouble when compiling sequential codelets. This issue has been fixed in the last version of CERE.

OpenMP parallel codelets cannot be run on SE mode due to the lack of real pthread implementation in the SE susbsystem. Indeed, statically linking with *libiomp.a* results in a forced exit because of the missing pthread management syscalls.

With these changes, and assuming that all the syscalls have been implemented inside the gem5 simulator, any sequential codelet should work in SE mode. Given that there is no scheduler in gem5 SE mode, and given that there is no proper implementation of pthread implementation in SE mode[2], parallel codelets cannot be realistically replayed on SE mode.

Using the region __cere__is_ranked_475 with five meta-repetitions (six total runs of the loop, as one is used to warm the cache up), we observed 4x speedup, comparing to running the full benchmark[3].

			L2		
Name	Frequency[4]	L1D and L1I associativity[5]	Size	Assoc.	L3
Cortex-A15	1 GHz	2	1 MB	16	No
i5-3550	3,3-3,7 GHz	8	256 kB	8	Yes[6]
i5-3337U	1,8-2,7[7] GHz	8[8]	256 kB	16	Yes[9]
Q9100	2,26 GHz	8	8 MB[10]	16	No

Figure 4: Parameters used for CPU simulations.

---

[1] CERE version 0.2 was used in this paper.

[2] M5thread has not been tested due the lack of scheduler and the miss of important syscall implementations in SE mode.

[3] Class W inputs were used for all the results.

[4] Non-turbo - Turbo frequency when turbo techology is implemented

[5] The L1D and L1I size is always 32 kB.

[6] The size of the L3 cache is set to 8MB and its associativity to 16-way due to gem5 limitations, it should be 6MB and 12-way.

[7] Only the non-turbo frequency has been simulated

[8] It should be 6 MB.

[9] It should be 3 MB.

[10] It should be 6 MB and 12-way.

All the data analysis is done on the median of these five meta-repetitions, but as the fluctuation is at most 0,03% (figure 3)[11], we can imagine running a codelet with only two or three meta-repetitions without significant bias.

3.1.2 Fullsystem mode

To run codelets in FS mode, only a few changes have been done: a manually prepared recent image based on Ubuntu-core 14.04 was used instead of the standard Ubuntu 7.04 image available on gem5 site. The kernel used is version 3.2.40 with default gem5 configuration.

To run OpenMP applications, just putting the *libiomp5.so* and *libomp.so* in */usr/lib*[12] works. Therefore setting KMP_affinity to **scatter**, which should assign each thread to a different core if available, results in a segmentation fault when starting the codelet. As a consequence, codelets on small inputs shows inexploitable behaviour (figure 5) on four cores. Moreover, gem5 seems to hang when simulating more than one x86 CPU in FS mode, that is why all the applications (including multicore's one) were run using a one-core configuration.

3.1.3 Chosen codelets

We chose three codelets to efficiently reproduce different usages:

- NAS IS sequential: region __cere__is_ranked_475 which check whether the computed array is sorted or not.

- PARSEC[13] x264: region __cere__encoder_analyse_block_residual_write-_cabac_745 used in the CABAC encoding of the video[14].

- PARSEC blackscholes: region __cere__blackscholes_m4__Z9bs_threadPv_first, an OpenMP region calculating the option value based on the Black & Scholes's equation.

- PARSEC freqmine: region __cere__tree8scan1_DBEP4Data_first which generates a hash from the tree dataset.

3.2 Simulation models

3.2.1 Hardware configuration

The chosen configurations are detailed in figure 4. All systems are set with 8 GB of 1600 MHz DDR3, the cacheline size is always kept at 64 B, and all CPUs are quad-core without hyper-threading.

McPAT simulates all the CPUs with a 44nm manufacturing process, to observe the impact on the power consumption of the architectural tuning only. The power consumption data correspond to the *dynamic power* output of McPAT.
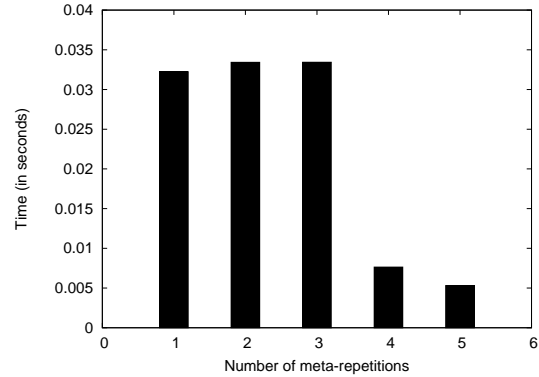

Figure 5: Variations of the codelet loop execution time on PARSEC Freqmine benchmark using the Cortex-A15 configuration with four AtomicSimpleCPU and simtest input.

4 Results

The power consumption of each CPU is calculated roughly using gem5 output file by the McPAT software: such value should not be taken absolutely but relatively to other CPU simulated. A power consumption of 10W is indeed too small for an one-core X86 desktop CPU running a benchmark.

One have to keep in mind that the CPUs use only using a generic single-core x86 model, without hyper-threading or other recent technological improvements (pipeline aside). The absolute measures are therefore quite irrealistic. Nevertheless, the ratio between performance points provide interesting feedback to evaluate the fitness of a codelet to a given architecture.

---

[11]This is due to the deterministic routine of the codelet, as the region __cere__is_ranked_475 is verifying that a give array is well-sorted. Such tight results were harder to get on non-deterministic or multithreaded code, see section 3.1.2.

[12]Taken from linux mint MATE 17.3 64 bits

[13]Version 3.0-beta-20150206

[14]This codelet reveals to be unmeasurable, see 4

Unfortunately, X264 codelet execution is too fast to be measured with gem5 (only 0,0000001 or 0,0000002 seconds are calculated), so this codelet will only be analysed for its power consumption.

4.1 Individual codelet results

4.1.1 IS: A serial applications

IS codelet results are unsurprising and match quite well what one would expect when ranking the tested CPUs (see figure 6): the most recent CPU, the i5-3550, is the fastest, either with or without boost. Boost only gives a small performance improvement. The Q9100 consumes less power, but is also less powerful. The lowest performance is given by the i5-3337U, a mobile processor, and the Cortex A-15.

As seen in figure 7, the instructions used are mostly integer one (60, 9%), following by load instructions (30, 4%), and store (8, 7%) instructions. IS does not use floating instructions at all, which explains the unsurprising linear performance variation of each CPU according to frequency: all of them use the same pipeline and integer compute units.
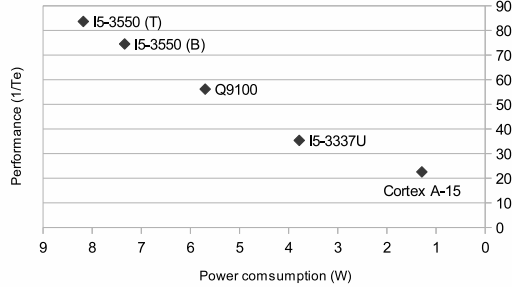

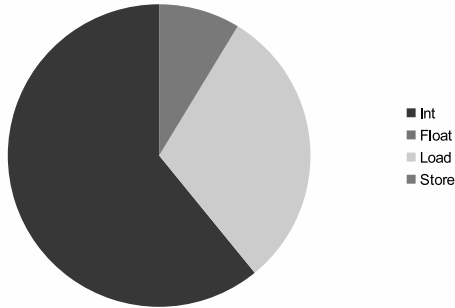Figure 6: Power-Performance graph for the IS codelet.

Figure 7: Repartition of the instructions during the execution of the IS codelet.

4.1.2 Parallel applications

This parallel applications are run on a one-core configuration: these results are only bound to show the single-core performance-consumption differences, and not the manycore scaling. In future work, such measures could be replicated on ARM systems thanks to 6.1.

Freqmine

Freqmine was tested with a small data input that fits fully in cache. The Q9100 L2 cache is faster than the i5-3350 L3 cache, so the former is slightly faster than the latter at stock frequency on this benchmark - and consumes the same power (figure 8). With the turbo enabled, the i5-3550 stays first. Then follows the mobile i5-3337U, and finally the Cortex-A15

On the instruction repartition (figure 9), freqmine is quite similar to IS: 56, 5% of integer instructions, 31% of loading ones and 12, 5% of storing ones. As IS, freqmine does not use floating point units.
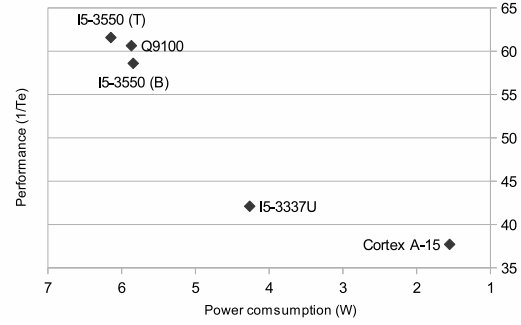

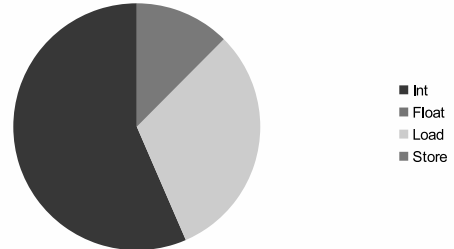Figure 8: Power-Performance graph for the Freqmine codelet.

Figure 9: Distribution of the instructions during the execution of the Freqmine codelet.

Blackscholes

Contrary to the other codelets, blackscholes uses floating operations. This application uses still mostly integer instructions $(56, 7\%)$, then floating operations $(23, 8\%)$, then $13, 8\%$ of loading instructions, and finally $5, 8\%$ of storing ones.

Here too, the CPUs follow the same performance ranking as for IS codelet: the i5-3550 first, then the Q9100, then the i5-3337U, ended by the Cortex-A15
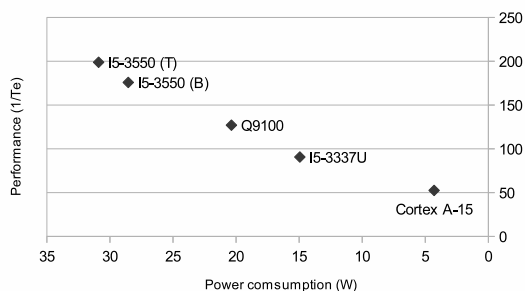

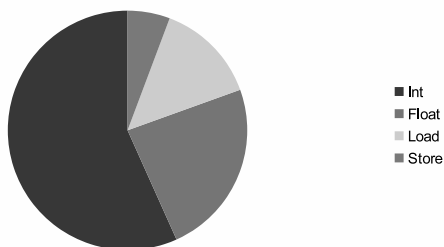Figure 10: Power-Performance graph for the Blackscholes codelet.

Figure 11: Distribution of the instructions during the execution of the Blackscholes codelet.

4.2 Performance

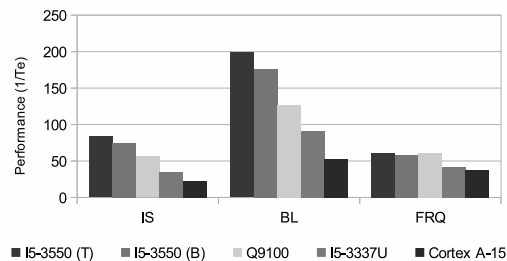The performance of all tested CPU on IS, blackscholes and freqmine is represented figure 12.

Figure 12: Performance of all tested CPUs on differents codelets.

IS and freqmine have a close execution time (figure 12), even if freqmine codelet was extracted running the simtest input. Besides, the relative gap between the Cortex-A15 and the high-end X86 CPUs is particularly vivid on IS and blackscholes codelets. On freqmine, the performance difference becomes really smaller. Once more, this is due to the simtest input, smaller, which fits in the 1MB L2 cache of the cortex-A15.

4.3 Power consumption

When the power consumption is on the x axis, the former is graduated backward to keep a *higher-is-better* index.
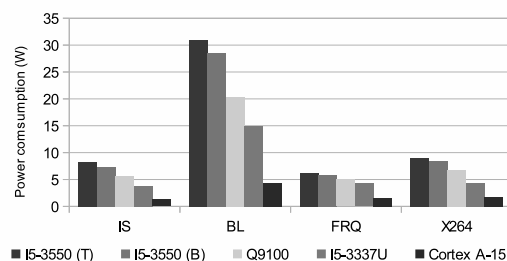

Figure 13: Power consumption for each CPU.

The blackscholes codelets seems to consume more power than all the other codelets (figure 13). This is due to floating point operations, used significantly only in this codelet (see 4.1.2).

Even if every simulation treats the Cortex A-15 as an X86 CPU, its power consumption is from $4, 0$ to $7, 2$ times smaller than the i5-3550 with turbo. This tends to show than the power consumption is mostly an architectural parameter and not a characteristic of the instruction set.

4.4 Power/performance measurement

4.4.1 Definition of the power/performance index

The performance of a CPU is measured by the execution time of the selected codelet. To keep an higher-is-better index, only $1/t_e$ values are used (where t_e is the execution time in seconds). The power-consumption ratio is defined as

$$I = \frac{1}{P.t_e} \qquad (1)$$

With P the power consumption of the CPU on the benchmark in Watt.

A higher ratio means either better performance or less consumption, and so a better choice.

This ratio can also be seen as follow:
$C = k.P.T_e$ is the cost to run an application with:

- k is the cost of power.

- P and T_e the same notation as before.

So $I = \frac{k}{k.P.T_e}$ is only the inverse of the normalized price paid to run the benchmark. We keep an higher-is-better index, as we aims at minimizing the operating cost.

4.4.2 Results on the test panel

The power/performance ratio for each codelet is compared on figure 14. The ARM Cortex-A15 is by far the best option for every codelet, if one wants to minimise the power/performance ratio.
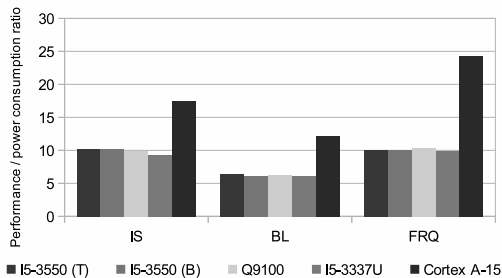

Figure 14: Performance-Power consumption ratio for each codelets.

This CPU aside, the classic X86 ones do not show some great relative differences. Therefore, blacksholes seems to limit the advance of the Cortex-A15: the small associativity of L1D and L1I probably limits it on floating operations.

4.5 Advantages

The use of codelets gives a powerful tool to fine-tune the architecture and codelet co-design. Because of CERE limitations, only codelets replay *for* loops or *omp parallel for* loops are supported, but this is much lighter than running the full application. Moreover, codelets can improve individually each region of an application, which enables per-region mapping when running on a big.LITTLE system, or any heterogeneus computer server.

The gem5 simulator is a precise tool to reproduce the behaviour of a specific machine on a general computer. As the simulation is quite slow, the use of codelets could be really useful in term of time and resources in the conception of heterogeneous servers. As CERE operates as an IR-level, the operation does not need any additional work on the source code, and can be run in C, C++ or Fortran programs. Besides, the compilation of the codelets uses the power of the host machine and not the simulated one, which is really useful in term of compute time: the emulated system is used only when it is truly needed.

4.6 Inconvenients

Gem5 is a simulator, and therefore cannot be entirely trusted. CERE too cannot exactly replay a whole execution of an application ; and the measures output by McPAT are only an estimation of the effective power consumption of the CPUs: that's why the results could not fit exactly to the real-life experiments. Nevertheless, these simulations are bound to give an overview of the gains that could be archived by improving a specific part of a CPU, not to give absolute results.

The gem5 simulator has not been validated yet on x86 accuracy, and could be quite biased, as it emulates a generic x86 processor, whereas state of the art CPUs are even more complex.

Besides, the power consumption is not exactly the right parameter to adapt, as one prefers usually a fast machine consuming a bit more than a really slow one.

Moreover, the manufacturing costs has never been reckon with in this paper, whereas architectural tuning needs a material support for further implementations.

5   Conclusion

This paper presents a novel approach on a hardware-to-software adaptation using codelets to speed up gem5 simulations.

The use of codelets is faster and nearly as efficient as regular benchmarks; and the combined use of gem5 along with McPAT offers a powerful and flexible tool to simulate any architecture modification on compute units.

Our primary results validate the fact that ARM architecture unsurprisingly really does consume less power than x86 standard one, unsurprisingly. But its power-performance ratio is up to 7 times better than that of a regular i5. On the hardware side, the cache size plays an important role in the performance, so one can imagine that some specific architecture could be developed, fitting the cache size (or type) to the data used for example.

In heterogeneous compute clusters, this method can be used to fine-tune the placement policy. Indeed, one can imagine a flexible machine composed by several compute units with various cache size, frequency, floating points and integer compute power. Simulating applications on these compute units individually can thus estimate which codelets fit better on which compute unit. During a real execution, the scheduler can base its decisions on these simulations to move on-the-fly an application to the compute unit which maximizes the performance-power ratio.

Even if only four CPUs are simulated, these results encourage to wisely choose the right architecture when selecting the hardware. That is why a conjoint development of future architectures, based on their efficiency for a specific use only, would be a great step forward in the conception of new architectures and lead to lower power consumption.

References

[1] David H. Bailey, Eric Barszcz, John T. Barton, D. S. Browning, Robert L. Carter, Leonardo Dagum, Rod A. Fatoohi, Paul O. Frederickson, T. A. Lasinski, Robert Schreiber, Horst D. Simon, V. Venkatakrishnan, and Sisira Weeratunga. The nas parallel benchmarks. *IJHPCA*, 5(3):63–73, 1991.

[2] Christian Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[3] Nathan L. Binkert, Bradford M. Beckmann, Gabriel Black, Steven K. Reinhardt, Ali G. Saidi, Arkaprava Basu, Joel Hestness, Derek Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[4] Anastasiia Butko, Rafael Garibotti, Luciano Ost, and Gilles Sassatelli. Accuracy evaluation of GEM5 simulator system. In Leandro Soares Indrusiak, Guy Gogniat, and Nikolaos S. Voros, editors, *7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), York, United Kingdom, July 9-11, 2012*, pages 1–7. IEEE, 2012.

[5] Pablo de Oliveira Castro, Chadi Akel, Eric Petit, Mihail Popov, and William Jalby. CERE: llvm-based codelet extractor and replayer for piecewise benchmarking and optimization. *TACO*, 12(1):6, 2015.

[6] Fernando A. Endo, Damien Couroussé, and Henri-Pierre Charles. Micro-architectural simulation of in-order and out-of-order ARM microprocessors with gem5. In *XIVth International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS 2014, Agios Konstantinos, Samos, Greece, July 14-17, 2014*, pages 266–273. IEEE, 2014.

[7] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In David H. Albonesi, Margaret Martonosi, David I. August, and José F. Martínez, editors, *42st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42 2009), December 12-16, 2009, New York, New York, USA*, pages 469–480. ACM, 2009.

[8] ARM ltd. Arm information center, 2015.

6 Annexe

6.1 ARM Simulation

As multicore benchmarks seem to hang on x86 simulation, some work has been done to adapt the codelet on ARM simulation. The multicore simulation works with linaro-minimal.

As capturing on ARM required an ARM machine, a few cross-compile instructions has been added in CERE. Capturing on gem5 may indeed take several days, and require moreover an ARM version of CERE pre-installed on the virtual machine, which was not ready at the time of this work. The goal is then to use an x86 memory dump and replay it on ARM systems. The following changes were made in CERE:

- Changed objdump to aarch64-linux-gnueabi-objdump.

- Added clang cross-compile option.

After those changes, the compilation ran successfully, but the execution outputs *"Killed"*, even before the main() starts.

We found that the aarch64 architecture limits the application space to the address 0x00000000_00000000 to 0x0000ffff_ffffffff[8]. But the stack is placed on x86 linux at address 0x07fda3c7_b0000000 so the kernel kills the application as soon as the x86 stack region is reserved. To avoid this issue, the stack has manually been moved to address 0x000003c7_b0000000.

NAS IS (codelet __cere__is_ranked_475) has been successfully replayed on a juno board (linaro-image-minimal-genericarmv8 system) using this trick. Nevertheless, further adaptations have to be done in order to safely convert x86 dumps to ARM dumps[15].

6.2 About the UVSQ laboratory

My internship took place at the University of Versailles Saint-Quentin, at Versailles, inside the LiPaRAD laboratory, under the directory of Pablo de Oliveira Castro.

My first weeks were mostly occupied by bibliography and primary setup of different tools, gem5 and CERE especially. I have spend most of my time adapting gem5 and modifying CERE to work with each other (including patches, improvements, bug reports, library adaptations and modifying a linux virtual drive image). The simulations were quite long (about one day to calculate all the results), and were repeated several time due to incorrect parameters. The conversion from gem5 output to McPAT input was done at the very end of my internship.

I have worked mostly with Mihail Popov (Ph.D. student), Chadi Akel (Engineer), both involved in the development of CERE, and Yohan Chatelain, a M2 intern in charge of the RAM capture when replaying an OpenMP region. I have also been helped by Lavania Ramapantulu, an intern from Singapore, who worked as me on PARSEC benchmarks. At the end of my work, William Jalby came to me in order to see if all went right. Pablo was always there when I needed some precisions or to validate each step that I have presented to him.

I spent two days during my internship at Polytechnique attending to the Teratec forum. It was really interesting, as I could talk to other researchers and discover what they are working on.

I liked the atmosphere, as I was really free in my research, but I could easily ask somebody when I encountered hardships. I particularly enjoyed having a clear line of work while being free in my timing schedule and my way to tackle this subject.

---

[15]Especially on pointers of stack address which need to be updated to the new stack position at the codelet compilation.

6.3 Gem5 patches

6.3.1 Getdents syscall:

```
SyscallReturn
getdentsFunc(SyscallDesc *desc, int num, LiveProcess *p, ThreadContext *tc)
{
    int index = 0;
    int sim_fd = p->sim_fd(p->getSyscallArg(tc, index));
    if (sim_fd < 0)
        return -EBADF;
    Addr buf_ptr = p->getSyscallArg(tc, index);
    int nbytes = p->getSyscallArg(tc, index);
    BufferArg buf_arg(buf_ptr, nbytes);

    int nread = syscall(SYS_getdents, sim_fd, buf_arg.bufferPtr(), nbytes);
    int errno_after_call=errno;

    if (nread > 0)
        buf_arg.copyOut(tc->getMemProxy());

    return (nread == -1) ? -errno_after_call : nread;
}
```

6.3.2 Read syscall bugfixe:

```
SyscallReturn
readFunc(SyscallDesc *desc, int num, LiveProcess *p, ThreadContext *tc)
{
    int index = 0;
    int tgt_fd = p->getSyscallArg(tc, index);
    Addr bufPtr = p->getSyscallArg(tc, index);
    int nbytes = p->getSyscallArg(tc, index);
    BufferArg bufArg(bufPtr, nbytes);

    int sim_fd = p->getSimFD(tgt_fd);
    if (sim_fd < 0)
        return -EBADF;

    int bytes_read = read(sim_fd, bufArg.bufferPtr(), nbytes);

    if (bytes_read != -1)
        bufArg.copyOut(tc->getMemProxy());

    return bytes_read;
}
```