Enhancing Multithreaded Performance of Asymmetric Multicores with SIMD Offloading

Jeckson Dellagostin Souza[1], Madhavan Manivannan[2], Miquel Pericàs[2] and Antonio Carlos Schneider Beck[1]
[1]*Universidade Federal Do Rio Grande do Sul*, Porto Alegre, Brazil
[2]*Chalmers University of Technology*, Gothenburg, Sweden
jeckson.souza@inf.ufrgs.br, madhavan@chalmers.se, miquelp@chalmers.se, caco@inf.ufrgs.br

***Abstract*—Asymmetric multicore architectures with single-ISA can accelerate multithreaded applications by running code that does not execute concurrently (i.e., the serial region) on a big core and the parallel region on a larger number of smaller cores. Nevertheless, in such architectures the big core still implements resource-expensive application-specific instruction extensions that are rarely used while running the serial region, such as Single Instruction Multiple Data (SIMD) and Floating-Point (FP) operations. In this work, we propose a design in which these extensions are not implemented in the big core, thereby freeing up area and resources to increase the number of small cores in the system, and potentially enhance thread-level parallelism (TLP). To address the case when missing instruction extensions are required while running on the big core we devise an approach to automatically offload these operations to the execution units of the small cores, where the extensions are implemented and can be executed. Our evaluation shows that, on average, the proposed architecture provides 1.76x speedup when compared to a traditional single-ISA asymmetric multicore processor with the same area, for a variety of parallel applications.**

***Index Terms*—functional unit sharing, offloading, SIMD, heterogeneity, multicore**

I. Introduction

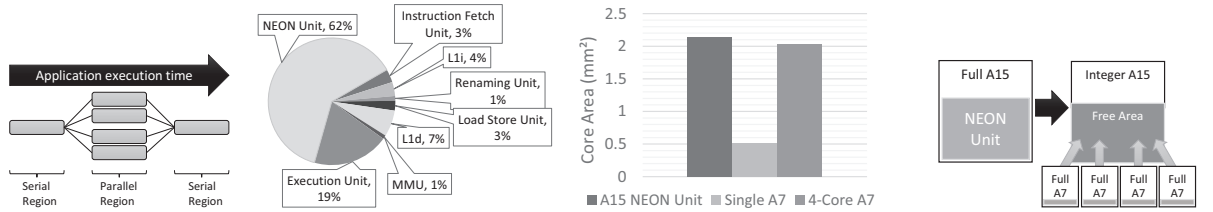
Asymmetric Multi-Core (AMC) architectures with single-ISA [1] can efficiently deliver high-performance at tight constraints and are widely used in the embedded domain. In such architectures, the availability of multiple cores can increase the throughput by exploiting thread-level parallelism (TLP), while heterogeneity can increase energy efficiency by enabling execution migration to energy efficient cores when high-performance is not required. This design strategy has been adopted in the big.LITTLE [2] architecture, and - more recently - in the DynamIQ [3], from ARM, wherein two distinct cores with different performance and energy characteristics co-exist in the same die. Since the cores implement the same ISA, threads can transparently be migrated between them, thereby enabling the scheduler to map threads to match the needs of the application. Such architectures are especially effective when running multithreaded applications since having multiple cores permit TLP exploitation, which is decisive for performance when running parallel region, while having big core(s) with better computational capability permits ILP exploitation, which in turn is useful to accelerate serial regions and critical sections. Figure 1a represents the behavior of a typical parallel application, which forks several threads and eventually joins for synchronization.

While Chip Multi-Processor (CMP) architectures are adopting heterogeneity in different levels [4], the ISA is also undergoing evolution: the processors are being incrementally tailored to increase the performance of emerging applications by extending the number of available instructions. These ISA modifications are mostly incorporated in the form of extensions thereby resulting in an increase in the complexity of the micro-architecture [5]. One example is the NEON instruction extension in the ARM ISA, responsible for executing Floating-Point (FP) and Single Instruction Multiple Data (SIMD) operations.

Although these instruction extensions are only used in specific scenarios (e.g. scientific computing and highly vectorized programs) and not extensively executed, the associated functional units add considerable area and resource overhead. Figure 1b, provides the area breakdown of components for the ARM A15 processor, clearly illustrating the overhead of the NEON extension. Our experiments indicate that the A15 core, an out-of-order processor, has two NEON pipelines and spans 62% of the core area. In comparison, Figure 1c shows that this same NEON unit in the A15 is larger than the total area of four A7 cores (including smaller NEON units) combined. Such difference in area exists mainly because of the characteristics of the NEON unit in the A15, which process wider words, has a larger pipeline, a larger FP register file and a dual execution lane.

We propose an AMC architecture, in which we trade-off area assigned to instruction set extensions and utilize it instead to improve TLP. This is done by removing the SIMD/FP from the big core to make room for extra small cores that can further exploit TLP, as pictured in Figure 1d. This design decision is motivated by the following observations: (1) SIMD/FP operations are less common than integer and memory instructions, (2) for parallel applications, SIMD/FP operations are mostly executed inside the parallel regions, and seldom in the serial region for which the big core is utilized and (3) parallel benchmarks typically have a considerably large parallel region in comparison to the serial region and consequently benefit more from TLP exploitation. In the case when SIMD/FP instructions are encountered in the serial region (running on the big core), we also propose the **t**ightly co**u**pled i**n**struction offload**e**r (TUNE) mechanism as an effective way to offload these operations from the feature-light big core to the SIMD/FP units of the small cores, thereby **ensuring ISA compatibility**

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, Brasil (CAPES), the Fundação de Amparo à Pesquisa do Estado do RS (FAPERGS) and the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).



(a) Multithreaded application forking (b) A15 out-of-order processor (c) A15 NEON against full A7 cores (d) Removing NEON in favor of A7s

Fig. 1: TUNE motivation and design.

of the entire system, without requiring code recompilation.

In summary, the contributions of this work are the following: (1) We propose to trade resources once used to implement Data Level Parallelism (DLP), in the form of SIMD instructions, to increase the Thread Level Parallelism (TLP) in order to improve performance of multithreaded workloads. (2) We present TUNE, an instruction offloading mechanism, which is responsible for assigning SIMD operations to cores that are able to execute them.

Furthermore, we evaluate TUNE using a full system simulation environment. Our results show that the TUNE mechanism can potentially increase performance on average by 1.76x when compared to a traditional AMC of same area.

II. TUNE Architecture

In the TUNE architecture, the expensive SIMD/FP pipelines are traded from the big core to make room for additional smaller cores. Our approach for removing these structures follows a similar methodology as in [6], where the SIMD and FP instruction extensions (along with additional extensions) are removed from an A15 processor. In this work, we only remove the functional units, leaving the decoder, the renaming logic, the issue queue, and the register file unaltered. This results in a design that is easier to implement and simplifies the offloading strategy. Although removing ISA extensions from a processor micro-architecture may introduce additional design effort, considering the modularity of some popular ISAs, we expect this to be minimal. This is because in ARM architectures, the NEON extensions are quite modular and can be removed. These extensions are even considered optional in some ARM processor families, such as the A9. In other recent architectures, such as the RISC-V, ISA extensions are also designed to be modular to ease the process of customizing the processor. Furthermore, the feasibility of sharing the SIMD/FP unit has already been demonstrated in commercial designs like the AMD Bulldozer, that shares a single SIMD/FP unit between two integer modules, and the Sun Niagara (Ultrasparc T1), that shares an FP unit between eight cores.

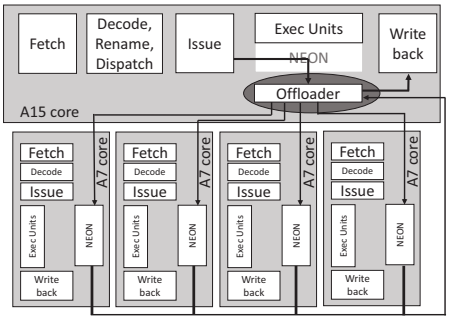
Figure 2a shows an overview of how the ARM A15 and the A7 cores can share the SIMD/FP units using the TUNE offloading strategy. A traditional A15 core implements two 128-bit wide NEON issue lanes. In our approach, these lanes are removed, and an **Offloader** is implemented in its place as shown in the figure.

The Offloader is tightly coupled to the NEON units of the A7 cores, allowing direct sharing of these units with the A15 core. As instructions reach the Offloader ready to execute from the issue queues, there is no need for additional operation decoding or identification in its logic. The Offloader is a simple splitter circuit, responsible for routing the operands (and the operation) of the SIMD/FP instruction from the vector registers of the big core to many NEON units in the little core. Figure 2b shows the 128-bit wide vector registers of the A15 core being split into two 64-bit wide registers, which are used to feed the A7 NEON units. Although both processors have vector registers of 64bits, the typical implementation of the A15 combines two registers to achieve operands of 128bits [7]. When offloading instructions, the 128bits registers are split back into 64bits, as expected by the A7 NEON unit. Using this approach, each of the two NEON issue lanes in the A15 core will be tightly coupled to two other A7 cores, offloading the same instruction for these two cores. Thus, the offloading is always done to a fixed pair of A7s. However, if the given operation is a scalar FP instruction, then the Offloader will only use a single A7 NEON unit (per lane), without splitting the registers. As the A15 core offloads the same instruction to a pair of A7s with all its dependencies and operands resolved, both A7 cores will finish their operation at the same cycle. This operations will be redirected back to the Offloader, which recomposes the original vector register and proceeds to the writeback stage normally, updating the reservation stations and re-order buffer. TUNE thereby keeps the original execution flow of the A15 core, while only incurring extra latency for SIMD/FP operations.

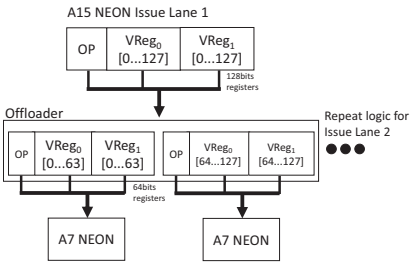
It is important to notice that an A15 core will only offload instructions when executing a serial region. During these execution phases, the A7 cores are always idle, which prevents the possibility of a conflict in the usage of the NEON units. In a multi-programmed scenario, with many different applications running in parallel, a scheduling policy would be required to manage the sharing of resources. We leave detailed exploration of this scenario for future work.

III. Methodology & Results

Modeling and Simulation: In order to obtain area estimates, we have modeled the A15 and A7 processors in McPAT [8] using $28nm$ technology node. Our models consider the entire core (including MMU and instruction and data L1 caches) without L2 caches. Although McPAT designs its components according to an A9 processor, we have used an approach similar to the one proposed in [9] to model the



(a) NEON instruction offloading from a A15 core to A7 cores. The NEON units from A7 cores are shared with the A15.



(b) The vector registers from the A15 core lanes are split into shorter registers and distributed over the A7 cores, along with the operation.

Fig. 2: Overview of the NEON instruction offloading system.

A7 and A15. The authors show that this approach results in models very close to the real processors. McPAT also allows for configurations without FP and SIMD units (by simply setting the FP related tags in the template to zero), which also triggers the exclusion of the FP instruction window, the FP Register File (RF) and the FP register renaming structures. However, in TUNE system, these structures are still needed for decoding and offloading instructions, thus we modified McPAT to include such components in the model. To extract the performance of our system, we have used the gem5 simulator [10] in Full System (FS) mode. The FS mode emulates an entire platform, including the Operating System (OS). Thus, the simulations include all the typical overheads of parallel programming a real system would have. The Offloader was also modeled to introduce the extra latency in the execution of SIMD and FP instructions. In our experiments, we have considered an extra ten cycles of latency for each SIMD and FP instruction, which is approximately the same latency of the L2 cache and the same amount of data (64 bytes).

Workloads: We have evaluated our TUNE configuration with several benchmarks of different characteristics (table I). The set includes the region of interest in applications of both high and medium ratios (percentage of time spent) of parallelism (Parallel Region - PR) and SIMD/FP (Serial Region SIMD - SRS) usage. Thus, we can analyze the system in different scenarios. To compile our workloads, we have used the GCC arm cross compiler arm-linux-gnueabihf-gcc version 7.3.0 with -O3 optimization flag, which includes flags to generate vectorized and floating-point NEON instructions.

Results: We start our analysis with table II, which shows two sets of results that help to understand the behavior of TUNE. The column $\frac{SingleA15}{SingleA7}$ shows how faster the A15 is compared to the A7 for each application. These results were obtained by simulating the single-threaded version of each application in both core types. Results show that the A7 slowdown varies from low 1.36x (*gemm*) to huge 5.70x (*gramschmidt*). Moreover, the column $\frac{8CoreA7}{SingleA7}$ shows the speedup achieved by an 8-core A7 processor when compared to a single A7, which represents how much performance the application can extract from parallel execution (8x means perfect parallel exploitation). Results show, as expected, that applications with smaller PR (from Table I), such as *correlation* and *covariance*, can achieve lower parallel speedups. Nonetheless, some applications that previously showed high coverage of the PR, such as *bicg*, which region of interest is virtually 100% covered by the PR (Table I), do not present the same expected speedup (only 4.86x). This is because many dynamic factors can influence the execution of the parallel region, such as shared memory accesses, data-synchronization, and bandwidth saturation [11]. Table II also shows the overhead data for the simulation of a TUNE system in column *Offloading Overhead*. This data is obtained measuring the cycles taken to execute each application with and without using the offloading strategy. The data shows that the overhead is usually low, and is only significant in applications that have smaller parallel regions and larger amounts of SIMD/FP in their serial parts (Table I).

TABLE I: Application **region of interest** characterization in terms of parallel region size and SIMD/FP ratio in the serial region.

	Parallel Ratio	Serial SIMD/ FP Ratio		Parallel Ratio	Serial SIMD/ FP Ratio
bicg	100.00%	0.00%	mvt	98.82%	1.11%
fdtd-apml	99.99%	0.00%	gesummv	98.76%	0.46%
convolution-2d	99.99%	0.00%	3mm	98.68%	1.66%
gemm	99.97%	0.06%	doitgen	98.41%	0.83%
symm	99.95%	0.00%	cholesky	96.56%	0.88%
syrk	99.90%	0.04%	trmm	82.25%	6.74%
syr2k	99.89%	0.04%	correlation	81.95%	10.10%
atax	99.73%	0.06%	gramschmidt	78.70%	11.72%
2mm	99.27%	1.03%	covariance	77.40%	17.33%
			lu	68.71%	0.09%

Figure 3 shows the results of two heterogeneous systems compared against a single A15 core. Both configurations have **the same area**, but in one of them (the TUNE system), we replace the NEON units of the A15 by four extra A7 cores and an offloader. The figure shows, both the speedups (left Y-Axis, represented by bars) of each configuration normalized by a single A15 core as baseline and the parallel ratio and SIMD/FP ratio of each application (right Y-Axis, represented by the background areas). It is important to notice that we use the A15 as baseline to show the speedups of using heterogeneous multicore systems, but the parity of area exists only between the traditional configuration (A15 (Full) + 4 A7) and the TUNE system (A15 (No NEON) + 8 A7).

There are applications that show a considerable amount

TABLE II: Simulated application characteristics data.

Benchmark	$\frac{SingleA15}{SingleA7}$	$\frac{8CoreA7}{SingleA7}$	Offloading Overhead
correlation	1.51x	3.91x	14%
covariance	1.70x	3.38x	26%
2mm	5.04x	6.35x	2%
3mm	4.45x	6.38x	3%
atax	2.04x	5.58x	0%
bicg	2.41x	4.86x	0%
cholesky	1.51x	5.39x	1%
doitgen	1.79x	7.37x	1%
gemm	1.36x	6.36x	0%
gesummv	1.73x	6.50x	1%
mvt	1.88x	6.31x	1%
symm	4.07x	6.31x	0%
syr2k	3.11x	7.44x	0%
syrk	2.02x	7.44x	0%
trmm	1.99x	5.61x	12%
gramschmidt	5.70x	3.68x	15%
lu	1.84x	3.02x	0%
convolution-2d	1.87x	7.80x	0%
Fdtd-apml	2.04x	7.62x	0%

of NEON instructions in their longer serial regions (e.g. *correlation, covariance* and *gramschmidt* have 10-17%, in table I,). These applications also have a smaller parallel speedup (as seen in table II), which results in the worst-case scenarios for the TUNE configuration, as more instructions will require offloading and the extra A7 cores will not be optimally used. The lack of parallelism in these applications will affect both the speedup of the traditional system and TUNE, but with the extra cores, TUNE is able to extract more performance from the application. This behavior is clearer in the application *lu*, which has longer serial regions, but almost no NEON operations. In this case, with no offloading overheads to hold TUNE back, our system is almost 2x faster than the traditional heterogeneous processor. The results show that in all these four applications, the TUNE system is faster than the traditional one, showing that the extra cores are able to compensate for the slowdown of offloading instructions, even when the application is not highly parallel.

The application *gramschmidt* has similar parallel characteristics, however, both configurations show a slowdown in performance when compared to the single A15, due to the huge performance loss of the A7 cores (5.70x in table II) being larger than the parallel speedup. One possible solution for the traditional configuration at this scenario would be to migrate all work to the A15 core, leaving the A7 cores idle and maintaining the higher baseline performance. The TUNE system could also adopt this policy, but in this case it would need to offload every NEON instruction of the application to the A7 cores, adding a considerable overhead. We have simulated this particular scenario, running the single-threaded application in the A15 core with TUNE, and found that there is an increase in 30% of the execution time when compared to a full A15 core. This is virtually **the same slowdown** we have observed in the multithreaded execution of this application in figure 3.

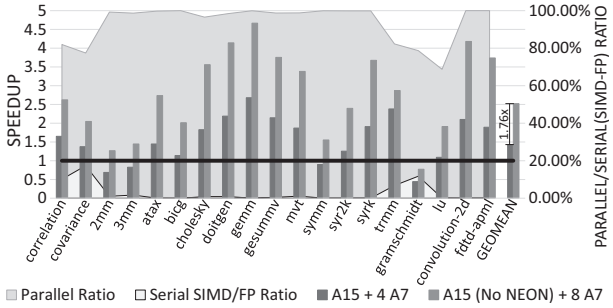


Fig. 3: Speedup according to the proposed system simulation. Bars are the speedup over a single A15 core and areas are the % of PR and SRS of each application.

Applications such as *2mm*, *3mm* and *symm*, have high parallel speedup (about 6x in table II) and small ratios of NEON instructions in the serial region (about 1% in table I). However, the slowdown of executing in the A7 cores is huge(4-5x in table II), at the point that executing these applications in the traditional full core configuration is actually slower than in a single A15 core. This is seen in figure 3 as these three applications are under the normalized performance of the baseline (constant black line). The proposed TUNE system, on the other hand, can overcome the A7 slowdown where the traditional system could not, by exploiting more parallelism with the extra cores. In the case of the *2mm*, the traditional system performance is of 0.69x of a single A15, while the TUNE system achieves 1.27x speedup, as seen in figure 3.

The applications that benefit the most from the extra cores are those that show smaller A7 slowdown, lower ratio of NEON instructions in the serial region, and higher parallel speedups, such as *doitgen*, *gemm*, and *convolution-2d*. These applications show high speedups in the traditional configuration of one A15 core and four A7 cores (up to 2.68x), but even higher in the TUNE configuration (up to 4.67x). As TUNE can provide more cores in the same area, it delivers more thread-level parallelism to these applications, without suffering from the performance loss of offloading instructions. Finally, when one considers the geometric mean of all applications, the traditional system shows an average speedup of 1.43x, while the TUNE system is 2.52x faster than the baseline. Thus, in average TUNE is 1.76x faster than the traditional heterogeneous system.

References

[1] R. Kumar *et al.*, "Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction," *MICRO-36.*, 2003.

[2] "big.LITTLE." https://developer.arm.com/technologies/big-little, 2019.

[3] "DynamIQ." https://developer.arm.com/technologies/dynamiq, 2019.

[4] A. C. S. Beck *et al.*, *Adaptable Embedded Systems*. New York, NY: Springer New York, 2013.

[5] B. Lopes *et al.*, "Shrink: Reducing the ISA Complexity Via Instruction Recycling," *ISCA'15*, pp. 311–322, 2015.

[6] W. Lee *et al.*, "Exploring Heterogeneous-ISA Core Architectures for High-Performance and Energy-Efficient Mobile SoCs," *GLSVLSI'17*.

[7] L. Mallia, "ARM Dev Conf 2007 Qualcomm High Performance Processor Core and Platform for Mobile Applications," tech. rep.

[8] S. Li *et al.*, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *MICRO'19*.

[9] F. Endo *et al.*, "Micro-architectural simulation of embedded core heterogeneity with gem5 and McPAT," *RAPIDO '15*, pp. 1–6, 2015.

[10] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, p. 1, aug 2011.

[11] A. F. Lorenzon *et al.*, "Aurora: Seamless Optimization of OpenMP Applications," *IEEE TPDS*, 2019.