A Reconfigurable Heterogeneous Multicore with a Homogeneous ISA

Jeckson Dellagostin Souza[1], Luigi Carro[1], Mateus Beck Rutzig[2], Antonio Carlos Schneider Beck[1]

[1]Universidade Federal do Rio Grande do Sul, Instituto de Informática, Porto Alegre, Brazil
[1]{jeckson.souza, carro, caco}@inf.ufrgs.br
[2]Universidade Federal de Santa Maria, Departamento de Eletrônica e Computação, Santa Maria, Brazil
[2]mateus@inf.ufsm.br

Abstract— Given the large diversity of embedded applications one can find in current portable devices, for energy and performance reasons one must exploit both Thread- and Instruction Level Parallelism. While MPSoCs are largely used for this purpose, they fail when one considers software productivity, since it comprises different ISAs that must be programmed separately. On the other hand, general purpose multicores implement the same ISA, but are composed of a homogeneous set of very power consuming superscalar processors. In this paper we show how one can effectively use a regular fabric to provide a number of different possible heterogeneous configurations while still sustaining the same ISA. This is done by leveraging the intrinsic regularity of a reconfigurable fabric, so several different organizations can be easily built with little effort. To ensure ISA compatibility, we use a binary translation mechanism that transforms code to be executed on the fabric at run-time. Using representative benchmarks, we show that one version of the heterogeneous system can outperform its homogenous counterpart in average by 59% in performance and 10% in energy, with EDP improvements in almost every scenario.

Keywords—reconfigurable system, multiprocessor, embedded systems, heterogeneous systems

I. INTRODUCTION

Current embedded systems have become popular with the dissemination of smartphones, wearables and many other smart gadgets. Such systems execute many kinds of applications: from web browsers to video decoders, which cover a wide and heterogeneous environment. To deal with this matter, multicore systems became very common on embedded environments. Having multiple cores allows the system to exploit the thread level parallelism (TLP) of applications. Homogeneous systems are usually comprised of superscalar processors [1], which allow instruction level parallelism (ILP) to be exploited as well.

However, current multicore systems that implement the same Instruction Set Architecture (ISA) are mostly homogeneous (all the cores have exactly the same resources), which is inefficient when running threads with heterogeneous workloads. In homogeneous systems, threads with lighter loads have to run on cores with many resources and more suitable for heavier threads, which results in resource wasting and high energy consumption. Heterogeneous multicore systems have cores that exploit different levels of instruction parallelism each, allowing these systems to distribute the applications threads more efficiently. Heterogeneous multicore processors that implement the same ISA rely on superscalar organizations, as initially proposed in [2]. The big.LITTLE technology [3] from ARM is an example of system that comprises two different superscalar processors (i.e. organizations) to execute the same ISA. However, superscalar processors are inherently inefficient: the dependency check must be done even if the same block of instruction is reexecuted (e.g. in loops). Moreover, the technique is restricted to fixed and very few distinct cores, because of the associated costs of designing different superscalar organizations.

Therefore, an alternative to superscalar processors are reconfigurable organizations. These systems can adapt themselves to the application at hand, reconfiguring their datapaths to improve execution. Implementation strategies are adopted to optimize data dependency and maximize the ILP exploitation, providing huge performance improvements and energy saving over classic processors [4], [5]. By replicating reconfigurable cores, it is also possible to build multicore systems, exploiting both TLP and ILP (the latter more efficiently). However, another advantage emerges when using reconfigurable fabric: as it is highly regular, there is a huge flexibility when it comes to delivering heterogeneous cores, by just varying the size of the reconfigurable system or number of different configurations supported. We take advantage of this fact and propose HARTMP: Heterogeneous Arrays for Reconfigurable and Transparent Multicore Processing.

To support total binary compatibility with legacy code and across the cores, HARTMP is also composed of a binary translator. It transforms, at run-time, the code to execute on the reconfigurable system at hand, regardless the amount of existent reconfigurable logic. We simulated different versions of HARTMP (having distinct sized cores) using a predictive scheduler and compared to its homogeneous counterpart, considering versions of equal area. Six applications that represent typical embedded environments were chosen as benchmarks. We show that HARTMP can be 59% faster on applications with low TLP, while still consuming 11% less energy.

This paper is organized as follows. Section II shows a review of existing work on reconfigurable organizations. Section III demonstrates the HARTMP organization. Section IV presents the simulation environment and the results. Finally, the last section draws conclusions.

978-3-9815370-6-2/DATE16/ ©2016 EDAA

II. Related Work

There are many proposed reconfigurable architectures in the literature. One can find different environments that apply some kind of adaptability to improve the performance of applications[6]–[8]. Considering only reconfigurable architectures built upon multicore systems, Watkins [9] presents a procedure for mapping functions in the ReMAPP system, which is composed of a pair of coarse-grained reconfigurable arrays that is shared among several cores. As an example of a system with homogeneous architecture and heterogeneous organization, one can find Thread Warping [10]. It extends the Warp Processing [11] system to support multiple threads executions. In this case, one processor is entirely dedicated to execute the operating system tasks needed to synchronize threads and to schedule their kernels in the accelerators. KAHRISMA [12] is another example of a totally heterogeneous architecture. It supports multiple instruction sets (RISC, 2- 4- and 6-issue VLIW, and EPIC) and fine and coarse-grained reconfigurable arrays. Software compilation, ISA partitioning, custom instructions selection and thread scheduling are made by a design time tool that decides, for each part of the application code, which assembly code will be generated, considering its dominant type of parallelism and resources availability. A run-time system is responsible for code binding and for avoiding execution collisions in the available resources.

CReAMS [5] (Custom Reconfigurable Arrays for Multiprocessor Systems) couples a homogeneous reconfigurable matrix of functional units with a main processor to form each core that comprises the system. CReAMS maintains binary compatibility, like the superscalar multiprocessors do, but using a special translation circuit. This hardware translator transforms sequences of instructions in configurations for the reconfigurable logic, which can be stored and reused in the future, avoiding repetitive code analysis – which is one of the main drawbacks of superscalar processors.

As just discussed, reconfigurable multicore architectures can be both homogeneous or heterogeneous, considering their architecture (i.e. what ISA is implemented) and organization (i.e. if the processors that comprise the system are the same or not). HARTMP is homogeneous in architecture, allowing binary compatibility, and heterogeneous in organization, which gives flexibility for thread scheduling. Therefore, the advantages of using HARTMP for heterogeneous cores over the architectures reviewed above include:

- Unlike KAHRISMA and Thread Warping, HARTMP is homogeneous in architecture. It employs a binary translation system implemented in hardware that eases the software development process since a well-known toolchain (i.e. gcc) is used for any of its versions. Neither source code modifications nor additional libraries are necessary as the binary translator will handle the process of generating configurations.
- KAHRISMA and ReMAPP rely in special and particular tool chains to extract Thread-Level Parallelism and to prepare the platform for execution. HARTMP does not change the current development flow; so well-known application programming interfaces (e.g. OpenMP) can be used. This way, the programmer can extract TLP using the same APIs used in typical systems.
- In contrast to ReMAPP and Thread Warping, HARTMP employs a coarse-grained reconfigurable fabric instead of a fine-grained one. Fine-grained architectures may provide higher acceleration levels, but their scope is narrowed to applications that have few kernels responsible for a large part of the execution time. Coarse-grained reconfigurable architectures are capable of accelerating the entire application as they have reduced reconfiguration time.
- Even though CReAMS can cover most of the drawbacks discussed, it is a homogeneous organization: all cores have the same resources and can exploit the same levels of instruction parallelism, which leads to inefficiency when it comes to execute applications with low or unbalanced TLP.

Heterogeneity on HARTMP is completely transparent to the programmer or the operating system. The instructions are evaluated and dispatched to the reconfigurable logic by the binary translator implemented on chip, so no additional modifications in code are necessary.

III. HARTMP Organization

A general overview of HARTMP is given in Figure 1(a). The thread-level parallelism is exploited by replicating the number of Dynamic Adaptive Processors (DAPs) [4], [13] (in the example of the Figure 1(a), by four DAPs). Each DAP is a transparent single-threaded reconfigurable architecture coupled to the processor. The communication among DAPs is done through a 2D-mesh Network on Chip using an XY routing strategy. HARTMP also includes an on-chip unified 512 KB 8-way set associative L2 shared memory, illustrated as SM in the Figure 1(a). We divided the DAP into four blocks to better explain it, as illustrated in Figure 1(b).

Block 1 in Figure 1(b) shows the structure of the reconfigurable datapath. It is composed of registers for input and output context and a matrix of functional units. The matrix is a combinational block with ALUs (Arithmetic and Logic Units), Multipliers, Memory access ports and multiplexers. The matrix is composed of levels that run in parallel with the General Propose Processor (GPP) (block 2). Each level has columns and rows. The columns have units that can run in parallel, executing instructions that do not have data dependency. As the multiplier stands as the critical path of the level, it is possible to align three ALUs in each row and keep the base frequency of the processor unchanged. Thus, each ALU row has three columns and can execute three data dependent ALU instructions in the same level. During the reconfiguration process, a basic block is mapped to the matrix, in order to execute the whole block in a combinational fashion.

Block 2 shows the processor coupled with the matrix. In this work, we use a SPARCV8 architecture running at 600MHz. The processor has a five-stage pipeline, which reflects a traditional RISC design (instruction fetch, decode, execution, memory access and write back). In block 3, the necessary storage components are illustrated. Apart from the usual L1 caches, two

other memories are used. The address cache holds the address for each basic block decoded by the dynamic detection hardware (block 4) and is used as an index (and to check existence) for the datapath configurations. The reconfiguration memory holds the bits necessary to reconfigure the datapath into a basic block indexed by the address cache.

The Dynamic Detection Hardware (DDH), represented in block 4, does the binary translation and data dependency check of the instructions in a basic block. DDH is four-stage pipelined circuit that runs in parallel to the GPP being out of the critical path of the system. Instruction Decode (ID) stage is responsible for decoding the operands in the base processor instruction to datapath code, while Dependence Verification (DV) checks if these operands have any dependency with the instructions already stored in the configuration being built. Resource Allocation (RA) stage uses the DV analysis to determine the optimal functional unit for the given operation inside the array. Finally, Update Tables (UT) stage saves the new allocation in the reconfiguration memory for future use. Every time a jump or an incompatible instruction is detected, a new configuration is started by the DDH and a new entry is created in the address cache. Further details of the DDH and how it interacts with the reconfigurable array can be found in [4], [13].

During the Instruction Fetch (IF) stage of the base processor, the Program Counter (PC) is compared to the values in the address cache. A hit on this cache means that the following sequence of instructions was already translated to a configuration. In this case, the processor's pipeline is stalled and the configuration is executed in the reconfigurable datapath, greatly exploiting the ILP of the application.

In HARTMP, each DAP has a different reconfigurable datapath. This allows for some cores to be bigger than others by having more resources. In other words, they are more efficient to execute threads that can exploit higher levels of instruction level parallelism. Similarly, smaller DAPs would be allocated to run threads with low ILP. Figure 1(a) shows a HARTMP processor with four cores where two of them are small, one is medium and the last is large.

IV. Results

A. Methodology

We have created two versions of HARTMP and compared them with other three configurations of CReAMS (homogeneous multicore reconfigurable system) in both execution time, energy consumption and Energy-Delay Product (EDP). CReAMS has already proven to be always better than a Multicore composed of superscalar processors considering the same transistor budget in different scenarios [5]. The goal is to measure which benchmarks take advantage of the heterogeneous environment, considering their resources and number of cores.

We have named the homogeneous configurations as Ho1, Ho2, and Ho3. Similarly, the heterogeneous versions are He1 and He2. Each heterogeneous version is composed of 50% large cores, 25% medium cores, and 25% small cores. For example, a 4-core configuration of He1 has two large, one medium, and one small core, while an 8-core configuration has four large, two medium and two small cores. These sizes vary across the configurations, to match the area parity. Thus the large core in He1 is different from the large core in He2, and so on. Table I shows the details of the reconfigurable logic (block 1 in Figure 1(b)) of each configuration. For example, the Ho1 configuration has a total of 27 ALUs distributed along 3 levels (which is 9 ALUs per level) and can hold up to 32 array configurations on its 32KB configuration cache. Table II contains the ratio of area between the heterogeneous and homogeneous versions. For instance, the intersection of Ho1 and He1 shows a ratio of 2:1, meaning that the area of two cores of Ho1 is approximately the

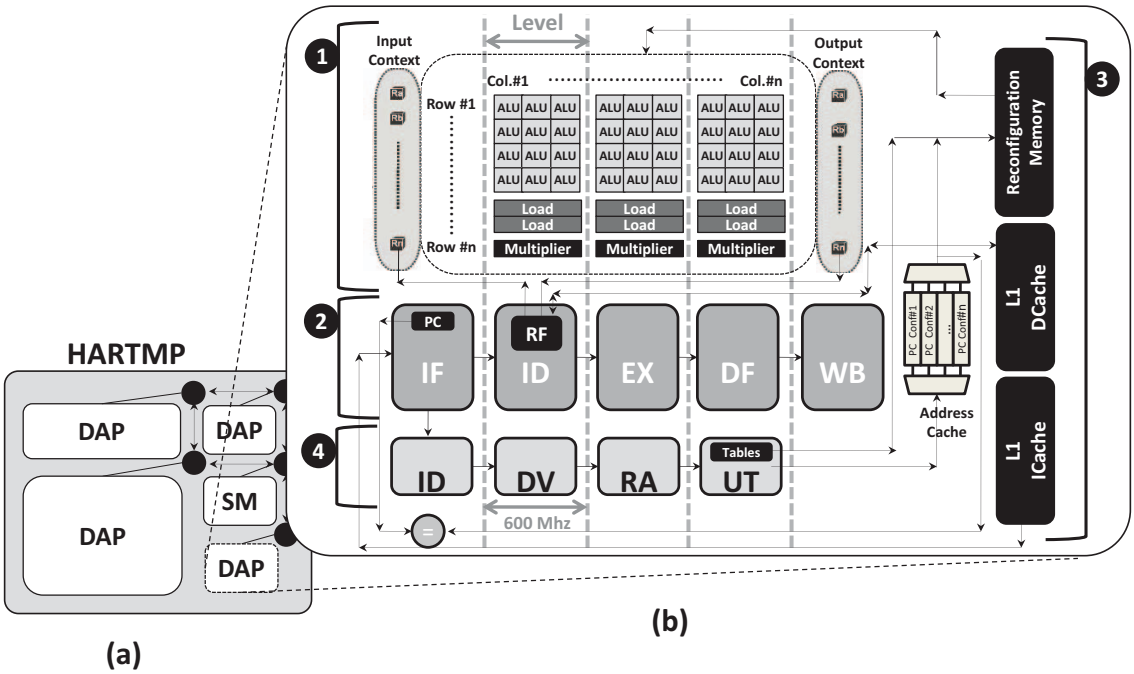


Figure 1(a) HARTMP: A multicore system composed of DAPs. (b) DAP: A processor tightly coupled to a reconfigurable logic and a binary translator.

same as one core of He1. We will use these parities of area to compare the processors.

All circuits are implemented in VHDL and synthesized to CMOS 90nm technology using Synopsys Design Compiler. The reconfiguration cache parameters were extracted using Cacti-P[14]. This work uses the Simics simulator [15] to generate the instruction trace from a set of applications. Each benchmark was executed using the same number of threads as the number of available cores. The trace was split according to its threads and each of these threads was allocated to an instance of a DAP simulator. We have also created a predictive scheduler for the threads. It is capable of deciding which thread and core size combination results in the fastest execution and is of great value to determine the potential of the proposed system.

Benchmarks of different suites were chosen to cover a wide range of applications in terms of available parallelism (i.e. TLP and ILP). We have selected benchmarks from parallel suites such as OpenMP[16], MiBench suite [17] and SPEC OMPM2001[18]. The latter is originally composed of single-threaded applications that we have ported to take advantage of multi-threaded environments. The benchmarks *susan_s, fft,* and *swaptions* have good load balancing between their threads, making them fine examples for TLP exploitation. Additionally, *susan_e, susan_c,* and *susan_s* have big mean basic block sizes. Therefore, they have more instructions in between control ones, which make them potential candidates to take advantage of ILP exploitation. Finally, *equake* has neither good load balancing nor big basic blocks.

B. Performance and Energy Evaluation

Tables III and IV show, respectively, the performance (in milliseconds) and the energy consumption (in Joules) for each of the simulated versions. The benchmarks are ordered accordingly to their TLP levels, being *equake* (top most) the one with least TLP and *swaptions* (bottom most) the one with most. The arrows numbered as 1-6 indicate the area parity between HARTMP and CReAMS versions (e.g.: arrow 1 indicates that 4He1, 2Ho3, 4Ho2 and 8Ho1 are all approximately the same size).

From Table III it is possible to notice that HARTMP best scenarios come from applications that have low TLP or applications with high ILP using manycore configurations. For instance, through arrow 1 we see that the 4He1 processor has a performance improvement of up to 33.5% over the 8Ho1 processor on the *equake* application. As *equake* has low TLP, the extra processors from the homogeneous configuration do not improve performance, while the heterogeneous processor has large cores that are used to execute threads that demand more ILP. On the other hand, in *swaptions* (an application with high TLP) the 4He1 processor performance is 89% worse than the 8Ho1. This almost 2x performance difference is due to the number of cores, which is twice higher on the homogeneous version. However, most applications have a limit for TLP exploitation: a point where extrapolating the number of threads does not increase the application's performance. *swaptions* TLP limit is reached at 16 threads, which is the point where the He1 configuration performs better than the Ho1: as seen in arrow 3, the 16He1 processor is 5.74% faster than the 32Ho1. On average, the He1 configuration is 59% faster than the Ho3 configuration and 22% faster than all the homogeneous versions, while the He2 is 17% faster than the Ho1, but 0.9% slower in overall.

TABLE I. FOR EACH PROPOSED ARRAY CONFIGURATIONS, THE TABLE SHOWS THE NUMBER OF LEVELS, MULTIPLIERS, MEMORY ACCESS, ALUS, RECONFIGURABLE CACHE AND INPUT CONTEXT.

Homogeneous - CReAMS						
Config	**Levels**	**Mult**	**Load/ Store**	**ALU**	**Reconf Cache**	**Input Context**
Ho1	3	3	6	27	32 Conf 32Kb	8
Ho2	5	10	20	75	64 Conf 64Kb	16
Ho3	8	32	40	144	128 Conf 256Kb	24
Heterogeneous - HARTMP						
He1						
Small (25%)	3	3	6	27	32 Conf 32Kb	8
Medium (25%)	5	10	10	60	64 Conf 64Kb	14
Large (50%)	8	16	16	96	128 Conf 128Kb	20
He2						
Small (25%)	3	6	12	54	32 Conf 32Kb	16
Medium (25%)	5	20	20	120	64 Conf 128Kb	28
Large (50%)	8	32	32	192	128 Conf 256Kb	32

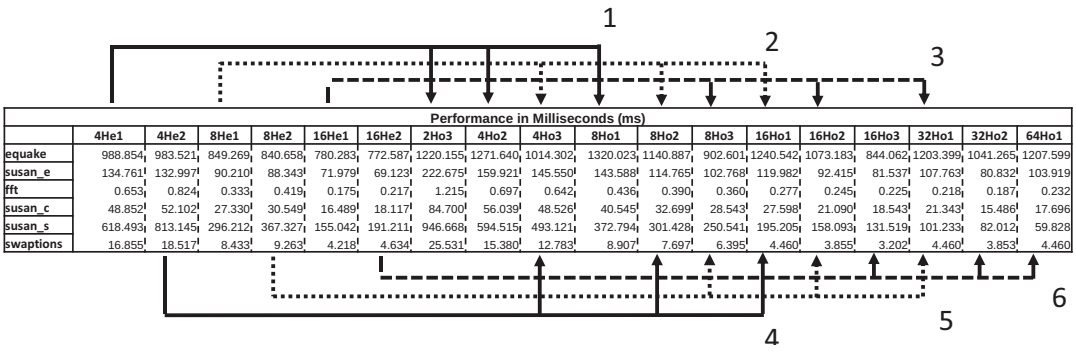
TABLE II. AREA RATIO

	Ho1	**Ho2**	**Ho3**
He1	2 Ho1 : 1 He1	1 Ho2 : 1 He1	1 Ho3 : 2 He1
He2	4 Ho1 : 1 He2	2 Ho2 : 1 He2	1 Ho3 : 1 He2

Energy consumption in HARTMP is affected not only by the size of the reconfigurable logic, but also by the size of the reconfiguration cache. As shown in Table I, larger cores in both HARTMP and CReAMS have also larger reconfigurable caches because they can hold more basic blocks and, as the array is bigger, each configuration needs more bits to set the datapath. Consequently, the reconfiguration cache in larger cores is more energetically expensive to access. In Table IV it is possible to analyze this cost with the comparisons of He1 and Ho2. Both versions have nearly the same transistor count, as seen in their junctions in arrows 1, 2 and 3, with the same number of cores. However, Ho2 (which has smaller caches) consumption is smaller than He1 in all the cases. A similar situation is observed with He2 and Ho3. On the other hand, when comparing He1 with Ho1, the heterogeneous version is more efficient in every scenario. For example, the arrow 1 joining 4He1 with 8Ho1 shows energy savings of 9% in *equake* and 10% in *susan_s*. Although 4He1 has larger cores and caches, the 8Ho1 has double the cores, which means double the number of SparcV8 processors, which are also responsible for energy consumption.

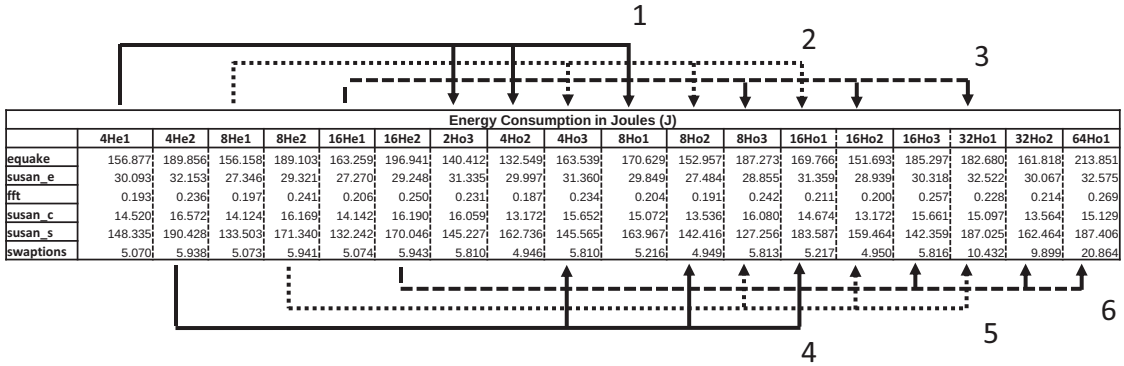
Results in energy and performance show that HARTMP has potential for efficient execution over the homogeneous counterpart depending on the characteristics of the application (unbalanced/saturated TLP). However, it is important to also

TABLE III. PERFORMANCE IN MILLISECONDS FOR EACH CONFIGURATION. ARROWS INDICATE THE PARITIES OF AREA BETWEEN HARTMP AND CREAMS VERSIONS.



Performance in Milliseconds (ms)																		
	4He1	4He2	8He1	8He2	16He1	16He2	2Ho3	4Ho2	4Ho3	8Ho1	8Ho2	8Ho3	16Ho1	16Ho2	16Ho3	32Ho1	32Ho2	64Ho1
equake	988.854	983.521	849.269	840.658	780.283	772.587	1220.155	1271.640	1014.302	1320.023	1140.887	902.601	1240.542	1073.183	844.062	1203.399	1041.265	1207.599
susan_e	134.761	132.997	90.210	88.343	71.979	69.123	222.675	159.921	145.550	143.588	114.765	102.768	119.982	92.415	81.537	107.763	80.832	103.919
fft	0.653	0.824	0.333	0.419	0.175	0.217	1.215	0.697	0.642	0.436	0.390	0.360	0.277	0.245	0.225	0.218	0.187	0.232
susan_c	48.852	52.102	27.330	30.549	16.489	18.117	84.700	56.039	48.526	40.545	32.699	28.543	27.598	21.090	18.543	21.343	15.486	17.696
susan_s	618.493	813.145	296.212	367.327	155.042	191.211	946.668	594.515	493.121	372.794	301.428	250.541	195.205	158.093	131.519	101.233	82.012	59.828
swaptions	16.855	18.517	8.433	9.263	4.218	4.634	25.531	15.380	12.783	8.907	7.697	6.395	4.460	3.855	3.202	4.460	3.853	4.460

TABLE IV. ENERGY CONSUMPTION IN JOULES FOR EACH CONFIGURATION. ARROWS INDICATE THE PARITIES OF AREA BETWEEN HARTMP AND CREAMS VERSIONS.



Energy Consumption in Joules (J)																		
	4He1	4He2	8He1	8He2	16He1	16He2	2Ho3	4Ho2	4Ho3	8Ho1	8Ho2	8Ho3	16Ho1	16Ho2	16Ho3	32Ho1	32Ho2	64Ho1
equake	156.877	189.856	156.158	189.103	163.259	196.941	140.412	132.549	163.539	170.629	152.957	187.273	169.766	151.693	185.297	182.680	161.818	213.851
susan_e	30.093	32.153	27.346	29.321	27.270	29.248	31.335	29.997	31.360	29.849	27.484	28.855	31.359	28.939	30.318	32.522	30.067	32.575
fft	0.193	0.236	0.197	0.241	0.206	0.250	0.231	0.187	0.234	0.204	0.191	0.242	0.211	0.200	0.257	0.228	0.214	0.269
susan_c	14.520	16.572	14.124	16.169	14.142	16.190	16.059	13.172	15.652	15.072	13.536	16.080	14.674	13.172	15.661	15.097	13.564	15.129
susan_s	148.335	190.428	133.503	171.340	132.242	170.046	145.227	162.736	145.565	163.967	142.416	127.256	183.587	159.464	142.359	187.025	162.464	187.406
swaptions	5.070	5.938	5.073	5.941	5.074	5.943	5.810	4.946	5.810	5.216	4.949	5.813	5.217	4.950	5.816	10.432	9.899	20.864

evaluate the tradeoff between the execution time and savings in energy.

C. Energy-Delay Product Evaluation

Energy consumption and performance are tightly-coupled parameters, as to increase performance, one usually needs to increase the complexity of the organization. In the Figure 4, we analyze the tradeoff between energy and performance using the Energy-Delay Product metric. In the Figure 4, we have compared the EDP of configuration He1 with all homogeneous versions. As in Tables III and IV, the configurations are compared using area parity, however in the Figure 4 we have grouped all He1 and Ho1 core variations in the first chart, all He1 and Ho2 in the second and all the He1 and Ho3 in the third. Results above 0% means that the HARTMP version had better EDP, while those below 0% means a better EDP for CReAMS.

When comparing He1 with Ho1, it is noticeable that HARTMP has better EDP in the applications with low TLP. These are the best environments for heterogeneity, as threads do not have the same load balance. For the high TLP applications, the Ho1 version performs better until the application reaches its TLP limit, as already discussed in the previous sub-section. From that point on, having more cores do not provide much performance gains and actually increases the energy consumption, as seen in Table IV. This happens because the extra cores introduce more SparcV8 processors that stay active, but without actually performing any jobs. For instance, the EDP ratio in *susan_c* goes from negative to positive when Ho1 has 16 cores, while in *swaptions* this happens at 32 cores.

As seen in Table IV, the He1 spends more energy than the Ho2 (due to bigger caches in He1), however the heterogeneous configuration performs better (due to efficient core allocation and ILP exploitation in He1), as shown in Table III. This is reflected in the EDP results of the Figure 4 when comparing He1 with Ho2. The EDP gains rise with the number of cores because the performance increases and the energy consumption keeps almost constant, as both configurations have the same number of cores. EDP results for the He1 and Ho3 comparison are in accordance with Tables III and IV, where the heterogeneous version has better performance in all cases and better energy consumption in a few.

On average, the EDP ratio of He1 version is 33% better than the homogeneous configurations. When analyzing the He2 EDP

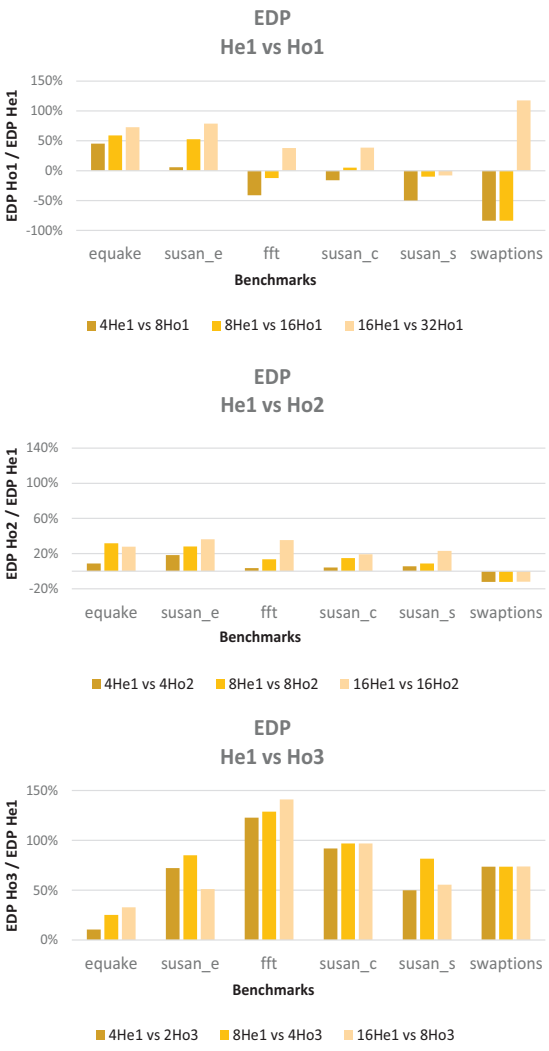


Figure 2: Energy-Delay Product relations between He1 HARTMP and all CReAMS configurations. Positive percentages means better EDP for HARTMP, while negative percentages represent better EDP for CReAMS

results (not shown in figures), the heterogeneous version has better EDP in only 2 of the six benchmarks tested. Although when analyzing energy and performance separately the homogeneous version is sometimes superior, the heterogeneous version always has scenarios with better EDP efficiency.

V. Conclusions

This work proposes HARTMP: a reconfigurable multiprocessor system with cores capable of exploiting different levels of instruction parallelism, aiming to offer efficient execution of applications with distinct thread load balance. Our solution provides a heterogeneous multicore organization with homogeneous ISA, which maintains code compatibility. Considering designs with same code area, our proposed heterogeneous processors offer improvements in both performance and energy consumption.

References

[1] J. L. Henessy and David A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed. Morgan Kaufmann, 2011.

[2] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction," *Proceedings. 36th Annu. IEEE/ACM Int. Symp. Microarchitecture, 2003. MICRO-36.*, 2003.

[3] "big.LITTLE Technology," 2015. [Online]. Available: http://www.arm.com/products/processors/technologies/bigLITTLEprocessing.php. [Accessed: 16-Oct-2015].

[4] A. C. S. Beck, M. B. Rutzig, G. Gaydadjiev, and L. Carro, "Transparent Reconfigurable Acceleration for Heterogeneous Embedded Applications," in *2008 Design, Automation and Test in Europe*, 2008, pp. 1208–1213.

[5] M. B. Rutzig, A. C. S. Beck, and L. Carro, "A Transparent and Energy Aware Reconfigurable Multiprocessor Platform for Simultaneous ILP and TLP Exploitation," *Des. Autom. Test Eur. Conf. Exhib. (DATE), 2013*, 2013.

[6] A. C. Schneider Beck Fl. and L. Carro, *Dynamic Reconfigurable Architectures and Transparent Optimization Techniques*. Springer-Verlag, 2010.

[7] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," *ACM Computing Surveys*, vol. 34, no. 2. pp. 171–210, 2002.

[8] A. C. Schneider Beck, C. A. Lang Lisbôa, and L. Carro, *Adaptable Embedded Systems*, 1st ed. New York, NY: Springer New York, 2013.

[9] M. a Watkins and D. H. Albonesi, "Enabling Parallelization via a Reconfigurable Chip Multiprocessor," *Pespma 2010-Workshop Parallel Exec. Seq. Programs Multi-core Archit.*, vol. 2, pp. 59–68, 2010.

[10] J. Lee, H. Wu, M. Ravichandran, and N. Clark, "Thread tailor," *ACM SIGARCH Comput. Archit. News*, vol. 38, no. 3, p. 270, Jun. 2010.

[11] R. Lysecky, G. Stitt, and F. Vahid, "Warp Processors," *ACM Transactions on Design Automation of Electronic Systems*, vol. 11, no. 3. pp. 659–681, 2006.

[12] Bingfeng Mei, A. Lambrechts, D. Verkest, J. Mignolet, and R. Lauwereins, "Architecture Exploration for a Reconfigurable Architecture Template," *IEEE Des. Test Comput.*, vol. 22, no. 2, pp. 90–101, Feb. 2005.

[13] A. C. S. Beck, M. B. Rutzig, and L. Carro, "A transparent and adaptive reconfigurable system," *Microprocess. Microsyst.*, vol. 38, no. 5, pp. 509–524, Jul. 2014.

[14] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "CACTI-P: Architecture-level Modeling for SRAM-based Structures with Advanced Leakage Reduction Techniques," in *Proceedings of the International Conference on Computer-Aided Design*, 2011, pp. 694–701.

[15] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer (Long. Beach. Calif).*, vol. 35, no. 2, pp. 50–58, 2002.

[16] A. J. Dorta, C. Rodriguez, F. de Sande, and A. Gonzalez-Escribano, "The OpenMP Source Code Repository," in *13th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, 2005, vol. 2005, pp. 244–250.

[17] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench : A free , commercially representative embedded benchmark suite The University of Michigan Electrical Engineering and Computer Science," *Proc. Workload Charact.*, pp. 3–14, 2001.

[18] K. M. Dixit, "The SPEC Benchmarks," in *Computer Benchmarks*, Elsevier, 1993, pp. 149–163.