TransRec: Improving Adaptability in Single-ISA Heterogeneous Systems with Transparent and Reconfigurable Acceleration

Marcelo Brandalero†, Muhammad Shafique*, Luigi Carro†, Antonio Carlos Schneider Beck†
† *Institute of Informatics, Universidade Federal do Rio Grande do Sul (UFRGS).* Porto Alegre, Brazil
* *Institute of Computer Engineering, Vienna University of Technology (TU Wien).* Vienna, Austria
{mbrandalero, carro, caco}@inf.ufrgs.br, {muhammad.shafique}@tuwien.ac.at

***Abstract*—Single-ISA heterogeneous systems, such as ARM's big.LITTLE, use microarchitecturally-different General-Purpose Processor cores to efficiently match the capabilities of the processing resources with applications' performance and energy requirements that change at run time. However, since only a fixed and non-configurable set of cores is available, reaching the best-possible match between the available resources and applications' requirements remains a challenge, especially considering the varying and unpredictable workloads. In this work, we propose *TransRec*, a hardware architecture which improves over these traditional heterogeneous designs. *TransRec* integrates a shared, transparent (i.e., no need to change application binary) and adaptive accelerator in the form of a Coarse-Grained Reconfigurable Array that can be used by any of the General-Purpose Processor cores for on-demand acceleration. Through evaluations with cycle-accurate gem5 simulations, synthesis of real RISC-V processor designs for a 15nm technology, and considering the effects of Dynamic Voltage and Frequency Scaling, we demonstrate that *TransRec* provides better performance-energy tradeoffs that are otherwise unachievable with traditional big.LITTLE-like designs. In particular, for less than 40% area overhead, *TransRec* can improve performance in the low-energy mode (LITTLE) by $2.28\times$, and can improve both performance and energy efficiency by $1.32\times$ and $1.59\times$, respectively, in high-performance mode (big).**

***Index Terms*—adaptive systems, reconfigurable systems, big.LITTLE, accelerators, flexibility, performance, energy, efficiency.**

I. Introduction

Adaptability is a key to modern mobile systems since a wide range of tasks must be executed with changing and sometimes unpredictable run-time performance and energy requirements. This scenario has driven the design of heterogeneous systems composed of microarchitecturally-different General-Purpose Processor (GPP) cores. However, typically only two core choices are available: (1) one GPP optimized for low energy, and (2) another one for high performance. An industrial example of such an architecture is ARM's big.LITTLE [1]. Therefore, the range of architectural solutions is often limited and performance-energy tradeoffs suboptimal, even when Dynamic Voltage and Frequency Scaling (DVFS) is available. With the help of an example from our own experiments, we show in Fig. 1 that an application may execute in a *LITTLE* core, for low energy and slow performance, or migrate to a *big* core, with $3.3\times$ better performance and $2.8\times$ higher energy consumption at nominal voltage. Besides the wide gap between the two operating ranges, both operating points provide nearly the same Energy-Delay Product (EDP).

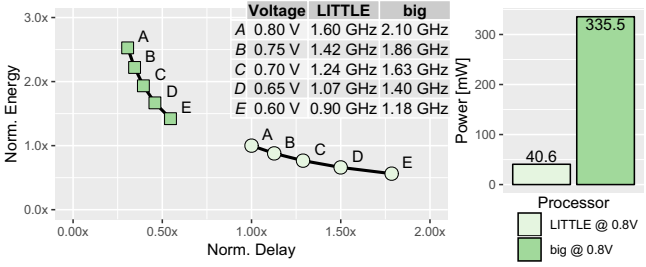

Fig. 1. Energy-delay (left) and power (right) tradeoffs for two cores operating in a heterogeneous system: a single-issue core (*LITTLE*) and a 2-issue OoO core (*big*). Different points in the left chart show the effects of DVFS.

Reconfigurable architectures, on the other hand, have the ability to create customized datapaths at run time, thus providing a nearly-continuous range of architectural solutions [2], [3]. As reconfigurable accelerators are typically coupled to GPPs, they can be used to extend traditional heterogeneous designs and improve their adaptability, potentially achieving better performance-energy tradeoffs.

In this work, we propose coupling reconfigurable logic to a heterogeneous single-Instruction-Set Architecture (ISA) system as means to improve the range of architectural solutions and achieve better Pareto-optimal performance-energy tradeoffs. We do that by extending a system similar to ARM's big.LITTLE design, based on real RISC-V processors and synthesized for a recent 15nm node. The *TransRec Architecture* extends this design with a transparent (i.e. works for existing binaries) Coarse-Grained Reconfigurable Array (CGRA) that is shared among the two cores, providing acceleration capabilities to both cores *without need to recompile the program*. By doing so, this work makes the following contributions.

- We propose a novel architectural improvement to heterogeneous systems where a transparent CGRA is coupled to a big.LITTLE-like architecture and can be used for transparent acceleration, extending the adaptability of heterogeneous systems and improving performance-energy tradeoffs.
- We show that, for nearly the same energy consumption as the *LITTLE* core, TransRec can achieve $2.28\times$ better

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. The authors would also like to thank CNPq and FAPERGS for partial support.

performance, or achieve $1.32\times$ better performance than the *big* core with $1.59\times$ reduced energy consumption.

II. Related Work

Reconfigurable architectures have been extensively investigated in the past years [2], [3]. We restrict the discussion to architectures which can accelerate application binaries already deployed by using run-time (transparent) techniques and architectures proposed in multi-core scenarios.

Most of the previous works have proposed reconfigurable architectures to improve a single form of core microarchitecture, a single-issue core [4]–[7] or a complex OoO one [8]–[10]. By restricting the coupling to a single microarchitecture, important points in the design space are missed, since a fraction of the base application must still execute in the GPP. Even works in multi-core scenarios consider only homogeneous GPP cores [11]–[13].

The TransRec architecture proposed here is also transparent and uses a CGRA for acceleration. However, unlike previous works that improve over homogeneous GPPs, the TransRec Architecture improves adaptability by using two distinct GPPs to best optimize for performance or energy. Compared to the multi-core works where a CGRA is used, this is the first one where a transparent CGRA is coupled to a heterogeneous system with different GPP cores, similar to a big.LITTLE design. Noteworthy, our evaluation methodology uses real RISC-V processor designs synthesized for a recent 15 nm node for more accurate results.

III. The TransRec Architecture

The TransRec Architecture is a single-ISA heterogeneous system where two microarchitecturally-different GPP cores, one optimized for low-energy (LITTLE) and the other for high-performance (big), share a CGRA for on-demand acceleration. A top-down overview is presented in Fig. 2. A *System Tile* consists of a *LITTLE* core, a *big* core, and a TransRec Processing Unit (TRePU), which consists of a Binary Translation (BT) module, a TransRec Configuration Cache (TReCC), and a CGRA.

The process works as follows. An application begins execution in the *big* core. As it executes, the operating system may decide to migrate it to *LITTLE*, in case the task load is low, or keep it in the big otherwise. In case it migrates, the entire architectural state must be copied from one core to another, as described in [1]. When instructions complete their execution either in the *LITTLE* or *big* cores, they are sent to the TRePU and temporarily stored in an instruction queue. The BT module reads from this queue, translating sequences of instructions into CGRA configurations and saving them in the TReCC for accelerating their future executions.

Whenever the base processor attempts to fetch a new basic block from the instruction cache, either in *LITTLE* or in *big*, a lookup also takes place in the TReCC. If the block is not found, then the instruction sequence has not yet been translated and must execute in the base processor. Otherwise, a configuration is loaded from the TReCC and used to configure the operations and interconnections in the CGRA, while input register data is transferred from the base processor to the TRePU. A special instruction (invisible to the programmer) is inserted in the base processor pipeline to mark TransRec execution and to support precise exceptions. The TRePU starts execution starts as soon as the data path has been configured and all input registers transferred. When the previously-generated special instruction reaches the commit stage in the base processor, the TRePU is notified to send the results back to the processor.

A. The CGRA

The CGRA is a matrix of Functional Units (FUs) composed entirely of combinational logic and divided into rows and columns, as shown in the right side of Fig. 2. Data propagates from left to right, so that each FU occupies a row and a sequence of columns depending on its latency. Since the design is highly regular, it is easily configurable for exploiting distinct ranges of Instruction-Level Parallelism (ILP). For the technology under consideration in this work, Arithmetic-Logic Units (ALUs) have the latency of half a processor cycle, due to their simplicity, and correspond to a single column. Loads and stores are constrained by the data cache, with one read and one write port in our design, and take two processor cycles to complete. As shown in the figure, while a single load operation or store is executed, two chains of four data-dependent ALU operations may be executed. The design under consideration has 24 columns, following the pattern depicted in columns 1 to 4 in Fig. 2, supporting up to 60 operations.

FUs communicate via context lines, initially fed by values from the input context. Before each FU, a crossbar selects the context line that will feed the inputs. After each FU, another crossbar selects, for each context line, the value that will be propagated to the next columns. Beside these functions, each crossbars also routes the output values from each column to the results buffer, which holds in-order commit information for the operations just executed.

B. The BT module

The BT module is a 4-stage pipeline that transforms sequences of instructions into CGRA configurations on-the-fly, adapted from [14]. It reads from the instruction queue, and analyzes the data dependencies among them to generate new CGRA configurations for posterior acceleration. A greedy algorithm is used for mapping the incoming instructions to the lowest possible column in the CGRA (where the operation's input values and a functional unit are avaliable), while translation tables are filled with the configuration occupation. This process stops when an unsupported instruction arrives (e.g. floating point), or when no functional unit is available for the incoming instruction. The algorithm exploits control speculation by mapping instructions from multiple basic blocks to the same configuration, increasing inter-block ILP. Moreover, it also exploits memory dependence speculation, reordering loads with preceding stores whenever possible and checking at run time for correctness.

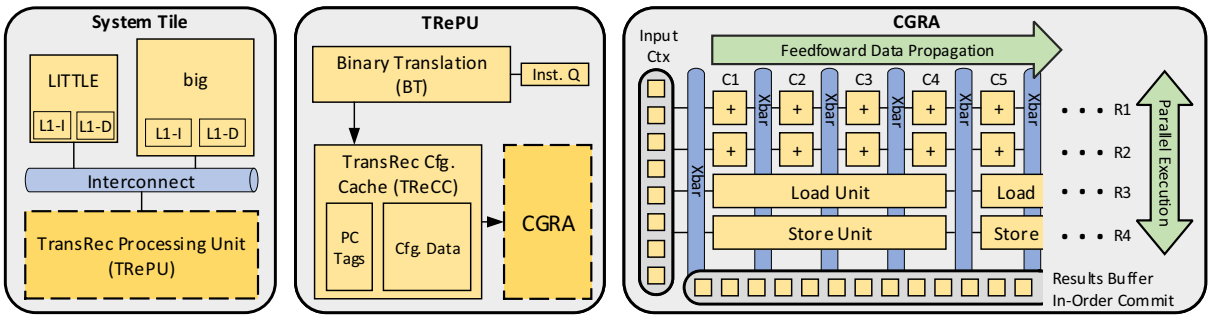

Fig. 2. Top-down overview of the TransRec Architecture.

When the process completes, a CGRA configuration is generated and stored in the TReCC along with the Program Counter (PC) of the first instruction in the translated sequence. The TReCC consists of two tables: (1) a PC tag array to store the memory location of sequences already translated and to quickly check if a configuration exists, and (2) a data array for configurations, accessed only when one must be fetched and executed. Each configuration encodes the FU operations, the crossbar routing and the logical destination registers of each instruction in the sequence (required to update the register file in the base processor later on), and occupies at most 256 bytes. We use a 128-entry data array, for a total of 32kB of storage.

IV. Results and Discussion

A. Evaluation Methodology and Tool Flow

We evaluate the TransRec Architecture in energy-efficiency and area against a baseline heterogeneous system similar to ARM's big.LITTLE. We target the RISC-V ISA (rather than ARM) due to the availability of open Hardware Description Language (HDL) processor designs [15], [16], allowing for high-accuracy area and power estimations.

We model the *LITTLE* core as a single-issue, in-order pipeline, running at 1.6 GHz, and the *big* core as a 2-issue OoO pipeline, running at 2.1 GHz. This operating frequency difference is also found in real big.LITTLE implementations (e.g. Samsung's Exynos 7420 and Qualcomm's Snapdragon 810). We also consider a *bigger* core for comparison, which is a 4-issue OoO pipeline also running at 2.1 GHz. The details of both OoO cores are depicted in Table I. Caches are modeled according to ARM's Cortex-A7 and Cortex-A15 implementations [17]; the TReCC is estimated to be similar to big(ger)'s L1-I cache, since their total size is the same.

For performance evaluation and fast design-space exploration, we extended the gem5 cycle-accurate simulator [18] with our BT and CGRA designs. We used the *TimingSimple* CPU to model a single-issue core and *O3* CPU for an OoO core. 15 benchmarks from mibench [19], typically found in the embedded domain, compiled for the RISC-V ISA with -O3 and running the "small input set" were used in the evaluation.

For area and power, we used logic synthesis with Cadence RTL Compiler and NanGate's 15nm standard cell library [20] for the cores and FinCACTI [21] for the caches. Synthesis considered two configurable RISC-V cores, Rocket (Single-Issue, *LITTLE*) [15] and Boom [16] (OoO, *big*). The CGRA was estimated based on BOOM's FUs.

TABLE I
OoO processors used in the analyzes.

Parameter	Processor	
	big: 2-issue OoO	bigger: 4-issue OoO
Pipeline Depth	12 Stages	
Inst-Q, Ld-Q, St-Q, ROB	24, 8, 8, 96	32, 12, 12, 128
Issue Width	2-issue	4-issue
Issue Ports	2 ALUs	4 ALUs
	1 Mult (3 cycles, pipelined), 1 Ld , 1 St	

B. Improvements in Energy-Efficiency

Fig. 3 presents the EDP improvements in five distinct scenarios: *LITTLE*, *big*, *LITTLE+CGRA*, *big+CGRA*, *bigger*. In all analyses, the CGRA runs at the same frequency as the processor it is coupled to. We group the results per benchmark and normalize EDP in each group with respect to *LITTLE*.

Considering the baseline system, execution in *big* introduces, on average and w.r.t *LITTLE*, $2.53\times$ energy overhead for a performance improvement of $3.27\times$, which results in $1.29\times$ better EDP. Behaviour for each benchmark depends on how effectively *big* can exploit an application's ILP. For instance, in *bitcount*, EDP worsens by a factor of $0.72\times$ when executing in *big* instead of *LITTLE*, while in *susan-e* the EDP improves by $1.92\times$. In these two applications, the average committed ops per cycle in *LITTLE* and *big* are 0.82 and 1.43 (*bitcount*, $1.74\times$ improvement) and 0.33 and 1.04 (*susan-e*, $3.15\times$ improvement). Memory accesses also play a key role: in *susan-e*, with 37% of the operations related to memory reads, caches will be responsible for a significant fraction of energy consumption, with significant higher overheads for the *LITTLE* core than for *big*.

When executing in the CGRA, average results show that *LITTLE+CGRA* improves *LITTLE*'s EDP by $2.21\times$, and *big+CGRA* improves *big*'s EDP by $2.10\times$, reaching energy-efficiency spots unachievable with traditional heterogeneous designs. The improvement is better for *LITTLE* than *big* for two reasons: (1) the power overheads introduced by the accelerator are proportionally larger for the *LITTLE* core than for the *big* one, and (2) switching execution from the complex *big* core to the CGRA allows for more significant power

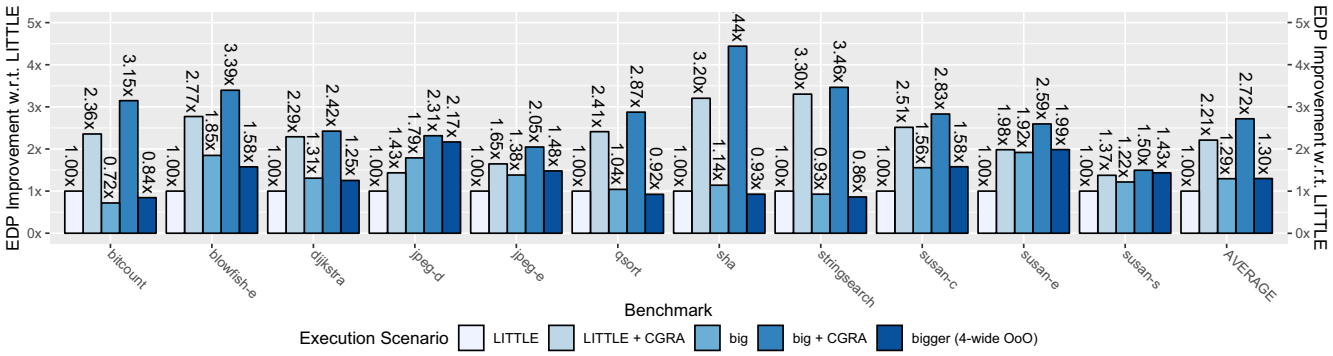

Fig. 3. EDP improvements compared to execution in the *LITTLE* core.

reduction due to lower utilization of several OoO scheduling structures [8], [9].

Finally, it is noteworthy that using the a CGRA for transparent acceleration pushes performance of the *LITTLE* core towards that of the *big* core, enabling *LITTLE+CGRA* to achieve 56% of the speedup provided by *LITTLE* to *big* migration, despite the frequency difference. Likewise, *big+CGRA* pushes performance of the *big* core to $1.04\times$ that of the *bigger* core. This analysis shows that it is typically not enough to have only one base GPP core to which the CGRA is coupled to, since part of the application still executes in the GPP.

C. Area costs

big.LITTLE-like systems often come in different arrangements, not only with different microarchitectures but also with a distinct number of cores (of each type) in the same System-on-Chip (SoC). For comparison, we use a typical scenario of four LITTLE cores and four big cores, as in the Samsung's Exynos 7420 and Qualcomm's Snapdragon 810. Extending this system to TransRec would require 4 additional TRePU tiles, for a total area overhead of 32%. This overhead is smaller than 50%, which would be the cost of replacing the four *big* cores by *bigger* cores, and the choice for TransRec presents better energy benefits with modest performance increases, as demonstrated in section IV.B.

V. Conclusions

This work has motivated the need for improved adaptability in heterogeneous designs. Towards this, we presented TransRec, a hardware architecture where a single-ISA heterogeneous system is extended with a CGRA for transparent adaptability improvements. In particular, TransRec can improve average EDP in a big.LITTLE-like system by $2.10\times$, improving performance of the *LITTLE* core by $2.28\times$ with marginal energy overheads, and improve both performance and energy by $1.32\times$ and $1.59\times$ in the *big* core.

As a future work, we are investigating additional cluster configurations for executing multithreaded applications in TransRec.

References

[1] "big.LITTLE Technology: The Future of Mobile," tech. rep., ARM, 2013.

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

[3] M. Wijtvliet, L. Waeijen, and H. Corporaal, "Coarse grained reconfigurable architectures in the past 25 years: Overview and classification," in *SAMOS'16*, pp. 235–244, 2016.

[4] 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.

[5] N. Clark, M. Kudlur, S. Mahlke, and K. Flautner, "Application-Specific Processing on a General-Purpose Core via Transparent Instruction Set Customization," in *MICRO'04*, pp. 30–40, 2004.

[6] N. Paulino, J. C. Ferreira, J. Bispo, and J. M. P. Cardoso, "Transparent acceleration of program execution using reconfigurable hardware," in *DATE'15*, pp. 1066–1071, 2015.

[7] A. C. S. Beck and L. Carro, "Transparent acceleration of data dependent instructions for general purpose processors," in *2007 IFIP International Conference on Very Large Scale Integration*, pp. 66–71, 2007.

[8] F. Liu, H. Ahn, S. R. Beard, T. Oh, and D. I. August, "DynaSpAM : Dynamic Spatial Architecture Mapping using Out of Order Instruction Schedules," in *MICRO'15*, pp. 541–553, 2015.

[9] M. Brandalero and A. C. S. Beck, "A Mechanism for Energy-Efficient Reuse of Decoding and Scheduling of x86 Instruction Streams," in *DATE'17*, pp. 1468–1473, 2017.

[10] V. Govindaraju *et al.*, "DySER: Unifying Functionality and Parallelism Specialization for Energy-Efficient Computing," *IEEE Micro*, vol. 32, no. 5, pp. 38–51, 2012.

[11] M. A. Watkins and D. H. Albonesi, "Enabling Parallelization via a Reconfigurable Chip Multiprocessor," in *Workshop on Parallel Execution of Sequential Programs on Multi-core Architecture*, 2010.

[12] M. B. Rutzig, A. C. S. Beck, and L. Carro, "A Transparent and Energy Aware Reconfigurable Multiprocessor Platform for Simultaneous ILP and TLP Exploitation," in *DATE'13*, pp. 1559–1564, 2013.

[13] J. D. Souza, L. Carro, M. B. Rutzig, and A. C. S. Beck, "A Reconfigurable Heterogeneous Multicore with a Homogeneous ISA," in *DATE'16*, pp. 1598–1603, 2016.

[14] A. C. S. Beck, M. B. Rutzig, and L. Carro, "A Transparent and Adaptive Reconfigurable System," *Microprocessors and Microsystems*, vol. 38, no. 5, pp. 509–524, 2014.

[15] K. Asanović *et al.*, "The Rocket Chip Generator," tech. rep., EECS Department, University of California, Berkeley, 2016.

[16] C. Celio, D. A. Patterson, and K. Asanović, "The Berkeley Out-of-Order Machine (BOOM): An Industry-Competitive, Synthesizable, Parameterized RISC-V Processor," tech. rep., EECS Department, University of California, Berkeley, 2015.

[17] M. Pricopi, T. S. Muthukaruppan, V. Venkataramani, T. Mitra, and S. Vishin, "Power-performance modeling on asymmetric multi-cores," in *CASES'13*, pp. 1–10, 2013.

[18] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, p. 1, 2011.

[19] M. R. Guthaus *et al.*, "MiBench: A free, commercially representative embedded benchmark suite," in *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization*, pp. 3–14, 2001.

[20] M. Martins *et al.*, "Open Cell Library in 15nm FreePDK Technology," in *ISPD'15*, pp. 171–178, 2015.

[21] A. Shafaei, , Y. Wang, X. Lin, and M. Pedram, "FinCACTI: Architectural Analysis and Modeling of Caches with Deeply-Scaled FinFET Devices," in *ISVLSI'14*, pp. 290–295, 2014.