Early Results from ERA – Embedded Reconfigurable Architectures

Stephan Wong, Anthony Brandon, Fakhar Anjam, Roël Seedorf
Delft University of Technology
Delft, The Netherlands
{J.S.S.M.Wong, A.A.C.Brandon, F.Anjam, R.A.E.Seedorf}@tudelft.nl

Roberto Giorgi, Zhibin Yu, Nikola Puzović
University of Siena
Siena, Italy
{giorgi, zhibin, puzovic}@dii.unisi.it

Sally A. Mckee, Magnus Själander
Chalmers University of Technology
Göteburg, Sweden
{mckee, magnus.sjalander}@chalmers.se

Luigi Carro
UFRGS
Porto Alegre, Brazil
carro@inf.ufrgs.br

Georgios Keramidas
Industrial Systems Institute
Patras, Greece
keramidas@isi.gr

***Abstract*—The growing complexity and diversity of embedded systems — combined with continuing demands for higher performance and lower power consumption — place increasing pressure on embedded platforms designers. To address these problems, the Embedded Reconfigurable Architectures project (ERA), investigates innovations in both hardware and tools to create next-generation embedded systems. Leveraging adaptive hardware enables maximum performance for given power budgets. We design our platform via a structured approach that allows integration of reconfigurable computing elements, network fabrics, and memory hierarchy components. Commercially available, off-the-shelf processors are combined with other proprietary and application-specific, dedicated cores. These computing and network elements can adapt their composition, organization, and even instruction-set architectures in an effort to provide the best possible trade-offs in performance and power for the given application(s). Likewise, network elements and topologies and memory hierarchy organization can be selected both statically at design time and dynamically at run-time. Hardware details are exposed to the operating system, run-time system, compiler, and applications. This combination supports fast platform prototyping of high-efficient embedded system designs. Our design philosophy supports the freedom to flexibly tune all these hardware elements, enabling a better choice of power/performance trade-offs than that afforded by the current state of the art.**

***Index Terms*—adaptive embedded platform; benchmarking; ρ-VEX VLIW processor;**

I. Introduction

Continuing innovation in the embedded systems market drives the need for increased usability via improving existing functionalities and inventing new ones. Product design cycles grow ever tighter, currently average about a year. Time-to-market commonly necessitates the use of standardized IP blocks and the re-use of existing designs. These practical approaches reduce design and verification time and effort, but they preclude niche-market optimizations that are not foreseen at design time. Such platforms quickly evolve from innovative to obsolescent in the face of emerging standards (e.g., for communication, audio and video processing). These kinds of platforms deliver the benefits of general-purpose systems at the cost of inherent inefficiencies with respect to specific functionalities. Rapidly changing application needs instead recommend support for adaptability over the lifetime of a product. FPGAs and eFPGAs provide such adaptation, but their programming times preclude dynamic, application-based reconfiguration, and they tend to have limited opportunity to influence memory and communication. Furthermore, embedded systems rely on limited energy supplies, and thus to best leverage the underlying technologies, such reconfiguration and optimization must likely be performed in hardware. The inherent complexity of this adaptation requires innovative solutions, both to address multiple markets and to address multiple applications within a market.

Providing a configurable fabric allows adaptation as necessary, but this innovation requires several problems to be addressed. First, power consumption of reconfigurable devices is generally high, given that programming them requires reading from an external memory. Second, support for reconfiguration necessarily requires additional wiring. Finally, design flows must remain sufficiently simple to hide optimization bottlenecks from the user. The ERA project addresses these problems by introducing a reconfigurable fabric that can adapt at design time, application deployment, and even during execution over the product's lifetime [1]. The ERA project has the following objectives:

- to define and develop a dynamically reconfigurable platform that is composed of a parametrized VLIW processor connected by a NoC with a memory subsystem;
- to provide support for flexible and fast platform reconfiguration by using hardware support and partial reconfiguration;
- to provide the required hardware monitoring and low level OS support for controlling the hardware reconfiguration;

978-1-4244-8396-9/10/$26.00 © 2010 IEEE

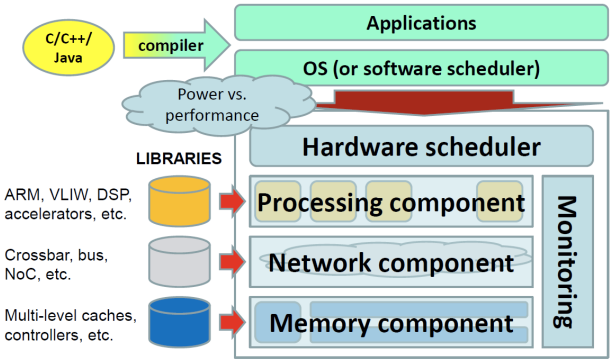

Fig. 1. High-level view of the ERA system.

- to benchmark a set of existing mobile applications in order to extract measurable parameters;
- to design a supervisor for monitoring the reconfiguration of the hardware when the application changes requires so.

II. Project Overview

To cope with the power problems that reconfiguration presents, our research emphasizes accelerator development within a coarse-grain reconfigurable fabric. The ERA family of architectures combines the ρ-VEX reconfigurable VLIW processor, flexible memory organizations, and a configurable interconnection network that provides better power management by distributing routing resources (Figure 1). The software stack comprises a compiler and an OS that can drive both static and dynamic reconfiguration decisions according to application characteristics and user power and performance objectives.

Choosing to use a single adaptable processor avoids creating a different accelerator for each new application, which is costly in terms of a products design time and time-to-market, and adapting processor organization on-the-fly avoids these pitfalls while offering flexible power and performance management. VLIW processors represent an excellent design point with respect to this management. Their issue width and dimensionality can be varied to meet size and power design constraints. This variable issue width can increase ILP at the cost of heavier pressure on the memory subsystem, thus we adapt it, as well. Knowing specific application behaviors and requirements — along with user specifications — allows us to precisely tailor the memory hierarchy organization and management to better match the processor's data-consumption needs. On-chip memory can be effectively re-sized, drowsy and decay policies can reduce power consumptions, shared-memory coherence communication can be minimized, and data placement and replacement can be controlled in hardware and software. Such malleable memory systems leverage information from the application and the compiler together with hardware monitoring and profile-directed feedback. Just as memory needs change from phase to phase and from application to application, so do communication requirements. We thus implement a reconfigurable NoC to manage the changing communication needs.

III. The Delft Reconfigurable VLIW Processor

We present the Delft ρ-VEX processor [2], an extensible and reconfigurable softcore that is based on the *V*LIW Example (VEX) ISA [3]. Hewlett-Packard (HP) and STMicroelectronics developed the VEX ISA as a 32-bit clustered VLIW that can be scaled and customized to individual application domains. Parameters include number and type of Functional Units (FUs), supported operations, memory bandwidth, and register file size. The VEX ISA is loosely modeled on that of the HP/ST Lx (ST200) family of VLIW embedded cores. The VEX trace-scheduling C compiler is a parameterized ISO/C89 industrial strength compiler. A programmable machine model determines the target architecture, which is then provided as input to the compiler. The VEX software toolchain (including the VEX C compiler and simulator) is freely available from Hewlett-Packard Laboratories. The ρ-VEX processor bridges the gap between application-specific and general-purpose processing. Advantages include simplified hardware and the ready availability of powerful tools.

One of the main advantages of VLIW processors is that their hardware design is relatively simple compared to that of dynamic-issue super-scalar processors, since there are no complex instruction decoders (e.g., for out-of-order execution): the compiler dictates all instruction scheduling. The hardware FPGA implementation is thus simplified, which permits higher clock frequencies. Adding more issue slots or functional units to the micro-architecture allows execution of the parallelism that the compiler can extract.

With respect to tools, VLIW compilers are readily available, and they continue to improve. Moreover, for the VEX ISA on which we base our designs, a simulator is available to investigate the performance gains for different architectural instances of the VEX processor. We can thus exploit existing compilers (and simulators), including future advancements to those technologies, without the need to first dedicate excessive effort to their development.

When the sizes of the reconfigurable hardware structure and the available hardware area are known a priori, one or several pre-configured VLIW softcore(s) can be instantiated on the FPGA. Trade-off studies with a simulator or model can quickly determine the best suited parameters of the available hardware for execution of the target applications. This scenario is most suited for the embedded design environment as the requirements and the platform are usually well-known and fixed.

Sharing of resources between multiple VLIW processors can also be dictated statically, but when neither the application nor the precise characteristics of the attached reconfigurable hardware is known at design time, resources may be shared dynamically. Enough resources must be instantiated to allow for sharing among the multiple VLIW processors which run on the same chip. In the same vein, new resources can be

instantiated on-the-fly, and when they are no longer needed, space can be freed to dedicate to other applications.

Properties of VLIW architectures that previously prevented mainstream adoption are easier to overcome in reconfigurable hardware. For instance, varying instruction word widths allows different phases or applications to exploit different levels of parallelism. But using wider or narrower instruction words requires different encoding schemes. Fortunately, the reconfigurable and parameterizable nature of VLIW makes it possible to instantiate different instruction decoders, either with or without reconfiguring the issue slots. Furthermore, unused issue slots can be shared among other softcores.

Traditionally, the fixed nature of VLIW implementations means that their organizations may not match inherent application parallelism, resulting in many scheduled NOPS. This leads to low utilization of resources (sometimes under 50%). Instead of inserting NOPs, the ρ-VEX can reconfigure the issue slots either by switching them off or by allowing other cores that need more performance to use the unused slots.

We can compile whole applications or threads for our ρ-VEX cores, or we can accelerate specific kernels. Code need not be rewritten, complex tools like C-to-VHDL translators need not be employed, and accelerator design need not be manual. Obviously, manual (re)design of both code and accelerators can be useful, as can complex tools, but ERA provides the option of a less complex, more automated solution that can often meet power, area, and performance requirements.

IV. Microarchitecture

The ρ-VEX processor allows design-time configuration of number and types of Functional Units (FUs), number of multi-ported registers, number and type of accessible FUs per sub-instruction (*syllable* – a VEX instruction consists of multiple syllables), issue-width, number of ALUs, number of multiply (MUL) units, number of General-Purpose Registers (GRs), number of Branch Registers (BRs), width of memory buses, and the architectural latencies of FUs. Figure 2 depicts the organization of a 32-bit, four-issue ρ-VEX VLIW processor. The processor's pipelined is divided into *Fetch*, *Decode*, *Execute 0*, *Execute 1*, and *WriteBack* stages. The Fetch stage reads VLIW instructions from instruction memory, splits them into syllables, and passes them to the Decode stage, which decodes the syllables and fetches register operands with the decoded register identifiers. In essence, this stage decodes each syllable to an operation, access registers and performs control transfers (in the *Branch* Unit). The accessed registers are send as operands to the Execute 0 stage. This stage performs arithmetic and logic operations (in the ALU and MUL units) on its operands. The Execute 1 stage performs a pre-selection of functional results for the commit stage and does the store or load phase of data memory operations in the *load/store* unit. The WriteBack stage performs all write activities and avoids read-after-write data hazards that are created by the decode stage. The register file write targets can be in the *General-Purpose Register* file (GR) and/or in the *Branch Register* file (BR). The data memory of the processor is implemented by utilizing the Block RAM (BRAMs) that are provided by the FPGA.

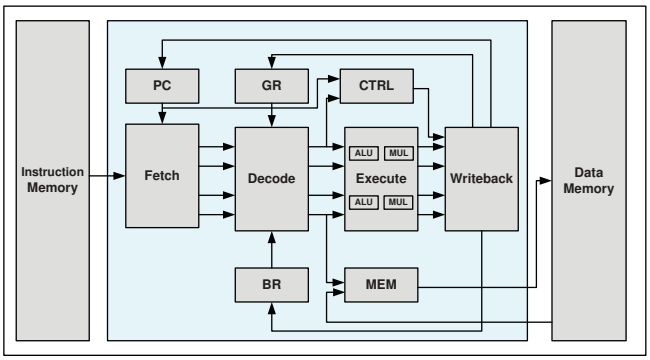

Fig. 2. A 4-issue ρ-VEX VLIW Processor instance.

V. Toolchain

The ρ-VEX toolchain consists of the HP VEX compiler, a port of the gnu binutils package with an assembler and a linker, and a tool that converts object files into synthesizable VHDL code for the instruction ROM and data memory. A machine description configures the HP compiler to generate assembly for different ρ-VEX instantiations. For the examples here, the machine description describes either a two-issue or four-issue core without forwarding. The assembler generates elf object files that are linked to form the executable, and a custom tool extracts the text and data segments to convert them to VHDL.

VI. ERA Software Stack

The software stack comprises not only the compiler and application APIs that are needed to guide reconfiguration, but it contains the OS and run-time support that manage the transformation process. We modify the OS to receive the task distribution in advance; it then manages the reconfiguration, decides when to reconfigure, and schedules work in concert with making reconfiguration decisions. The programmer can completely control reconfiguration via the provided API, or the platform's adaptivity can be left to the OS and the underlying hardware. Hardware provides key monitoring information on application behavior (e.g., functional unit utilization, memory intensity, or temperature), and the OS then decides the best application or the task placement to save power and/or deliver more performance.

The use of variable-issue-width VLIW processors prevents us from using existing VLIW compilers. Our compiler must be aware of the run-time reconfiguration of the architecture, generating multiple code versions (for different widths) and fast schedules for each. We are developing advanced optimizations to induce the most suitable architectural reconfiguration that match the application requirements.

VII. Benchmark Characterization

In order to understand the platform's potential to exploit reconfigurability, we first profile our target application suite, and characterize its behavior on whole programs, and program

intervals. This will guide micro-architecture configuration between applications/tasks. Analyzing a program's intervals guides more fine-grained reconfiguration which will be utilized for adapting the platform to different phase behaviors. Phase detection algorithms estimate the differences in the observed behavior between distinct intervals of execution [4]. To reflect the most common usage scenarios in systems like smartphones, ERA applications comprise image and video processing applications, security programs, and object and image recognition software (e.g., for augmented reality) [5].

Previous work has revealed several pitfalls of using microarchitecture-dependent measurements [6]. For instance, using hardware performance counters may be misleading, and even if they show similarity between two programs, the underlying program behaviors may be quite different. Additionally, small changes in the microarchitecture may trigger big differences in the program behavior. Since we are designing reconfigurable systems, we rely on a number of microarchitecture-independent metrics.

The behavior of programs with respect to memory system is characterized using the working set size, access strides (to determine spatial locality), and reuse distances [7] (to understand cache miss rates). The impact of the processor architecture on execution is determined by analyzing the degree of register use, register dependency distance [8] (to determine whether techniques like data forwarding would be effective) and instruction mix (to estimate the need for different functional units).

VIII. Initial Results

We developed the ρ-VEX processor to be technology independent. For instance, we implemented and tested versions using the different FPGA chips from Altera and Xilinx. Here we report measurements on the Xilinx Virtex-6 *XC6VLX240T-1FF1152* FPGA, which is available on the ML605 development board. We present the implementation results for a 2-issue and a 4-issue ρ-VEX processor instance. The processors are pipelined without forwarding logic. The two-issue core includes two ALUs and two MULs, while the four-issue core has four ALUs and two MULs. Both processor instances have one branch unit and a single load/store unit. For comparison, we choose the single-issue Microblaze processor that is the standard softcore on Xilinx FPGAs and we implemented both a speed-optimized and an area-optimized version in the same FPGA. We compare the performance of ρ-VEX processor with that of the Microblaze processor because it will function as the host processor within the multi-core ERA platform. Table I shows results for the Xilinx ISE 12.4 and the Virtex-6 *XC6VLX240T-1FF1152* FPGA.

The register file for the single-issue Microblaze core has two read ports and a single write port and it requires very few resources for its implementation. The ρ-VEX processor register files have to be more multi-ported. The two-issue ρ-VEX core's RF has four read ports and two write ports, and the four-issue ρ-VEX core's RF has eight read ports and four write ports. These multi-ported register file implementations consume considerable resources: more than 50% of the ρ-VEX hardware is consumed by the register file alone. At 64 times 32 bits, the ρ-VEX's RF is twice the size of the Microblaze's.

In contrast to the ρ-VEX designs, the single-issue Microblaze only needs one ALU and one MUL. Furthermore, it is a proprietary processor optimized to specifically deliver performance on the Xilinx FPGAs, and hence it is not possible to implement its design on a different fabric. On the other hand, the ρ-VEX is designed to be portable across different FPGAs. This means that we can further reduce its required hardware resources by optimizing the ρ-VEX for a specific FPGA family.

We select two kernels from the applications in the previous section. Most applications require either floating point support or system calls, neither of which are yet supported in our current ρ-VEX instantiations. The two kernels are the x264_pixel_sad_16x16 function from H.264, and the jpeg_fdct_islow from cjpeg. H.264 calculates the sum of absolute differences over several arrays. The cjpeg kernel performs a Forward Discrete Cosine Transform (FDCT) calculation. We added four other benchmarks to our experimental suite: matrix multiplication multiplies two 10×10 matrices, the ADPCM uses a Pulse Code Modulation (PCM) application to encode and decode data, Soma benchmark recursively computes the squares of the first 10 integers, and the DFT benchmark performs a Discrete Fourier Transform.

We run these six benchmarks on four different platforms: the Microblaze optimized for speed, the Microblaze optimized for area, a four-issue ρ-VEX, and a two-issue ρ-VEX. The speed optimized Microblaze deploys a 5-stage pipeline, achieves higher clock frequency, but requires more FPGA resources. On the other hand, an area-optimized Microblaze has a 3-stage pipeline. The Microblaze provides a baseline against which to measure the performance of instances of the ρ-VEX processor. All four processors operate at 100MHz. The benchmarks are compiled for the Microblaze with the -O3 optimization flag (including inlining). ρ-VEX applications are compiled with -O3 and -autoinline.

Figure 3 depicts results for ρ-VEX compared to the speed-optimized Microblaze, and Figure 4 depicts results compared to the area-optimized Microblaze. The figures show that for the soma and cjpeg applications the ρ-VEX is significantly faster for both the two-issue and four-issue implementations. These benchmarks exhibit large amounts of ILP in comparison to

TABLE I
Implementation Results.

Core	Slice Registers	Slice LUTs	DSP48E1s	LUTRAMs
two-issue ρ-VEX	2734	7372	8	0
four-issue ρ-VEX	3096	16955	8	0
Microblaze (speed optimized)	1041	1231	3	149
Microblaze (area optimized)	722	977	3	148

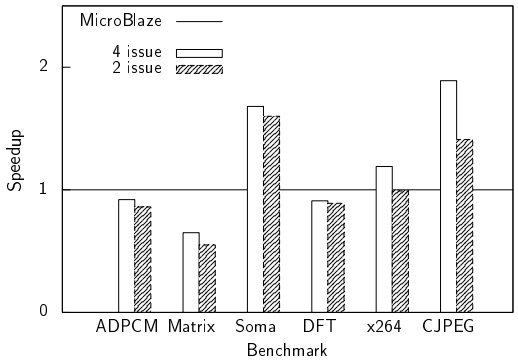

Fig. 3. Performance comparison with speed-optimized Microblaze.

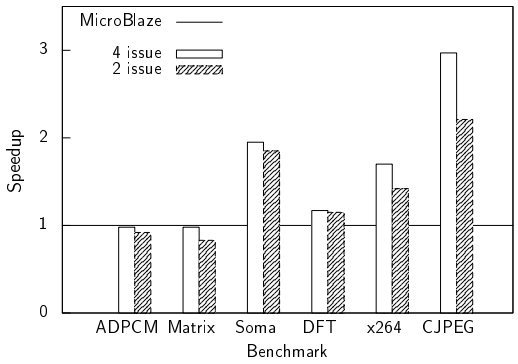

Fig. 4. Performance comparison with area-optimized Microblaze.

the others. For DFT, x264, and ADPCM our results resemble those of the Microblaze. These benchmarks have less ILP, which together with the lack of forwarding negates potential speedups. Finally, the Microblaze performs significantly better on the matrix benchmark than the two-issue and four-issue ρ-VEX cores. Here, the speed-optimized version of the Microblaze has a MUL that performs a 32×32 bit multiplication with a single-cycle latency. The ρ-VEX, on the other hand, requires three cycles, since it needs to compute two 16×32-bit multiplications and a sum of the products.

Comparing results from the four-issue and two-issue ρ-VEX cores shows a consistently small performance drop from four-issue to two-issue, except in the case of the cjpeg benchmark, which exhibits a larger decrease. Lack of forwarding logic in both designs means that the four-issue ρ-VEX contains more NOPs between dependent operations. While the two-issue ρ-VEX processor contains fewer NOPs, since operations are now split over multiple VLIW instructions. The upshot is that both organizations experience approximately similar cycle counts. Finally, Figure 4 compares the two ρ-VEXes to the area-optimized Microblaze. Here, the ρ-VEX performs very closely to the Microblaze in the two worst cases, but delivers a factor of three better performance in the best case.

A closer inspection of Figure 3 and Table I could lead to the conclusion to use two MicroBlaze cores instead of one in order to achieve a speedup factor similar to VEX, and still use less FPGA resources. However, such an implementation has to take care of the communication overhead that comes into play. Furthermore, one has to partition and map the application/kernel to both Microblaze cores. This means that porting an application in general becomes harder, or that of usage of a different programming model. the VEX compiler and the ρ-VEX micro-architecture, does not have these problems. Likewise, we do not compare the ρ-VEX processor with a MicroBlaze based MPSoC, as such a platform in principle presents a limit to the amount of spatial parallelism that its processing cores can execute. This is true, as the cores RISC pipeline contains a single processing lane. Hence, it can perform a limited amount of ILP execution.

At the micro-architectural level, there are three techniques to improve the performance and reduce the required FPGA resources of the ρ-VEX processor:

1) Bypass logic
2) Register Files
3) Functional Units

The goal of the first is to minimize the visible architectural latencies of in flight operations with read-after-write hazards. However, the design of this component must be done with care, as it can decrease the cores clock frequency. Bypass logic requires selection hardware that is located on the critical path of the pipeline. Furthermore, we can utilize register files that are optimized such as to benefit from splitting the decode stage into a decode and register access stage. This will decrease the clock-cycle time even further. The last technique increases the clock-frequency, as it utilizes faster and smaller functional units. We will optimize the design of the MUL and ALU FUs, in order to decrease the critical path of the processors pipeline and its required resources.

A. Application Characterization

To investigate whether switching between different configurations of the ρ-VEX processor at run-time benefits performance or power consumption, we characterize the phase behavior of our benchmark set. This characterization will show whether the phases are long enough to warrant a switch from one configuration to another. For this purpose, we utilize the xSTsim simulator that is configured to simulate the ST231 VLIW core. This processor has a similar micro-architecture as the ρ-VEX cores. The use of the simulator enables us to extract more statistics than we can for the native ρ-VEX processor.

The metric that is observed is the number of cycles needed to commit a VLIW bundle, and its change is measured over time (Figures 5 and 6). We can see that there is a difference in phase behavior between two applications (four phases in AC3 and nine phases in CJPEG). Furthermore, when the observation interval changes (not shown) the number of different phases changes. This can drive the reconfiguration to assign more resources when CPB is low, or to reduce them when CPB is high and performance is not the topmost priority.

B. Power

To determine the power efficiency of our platform, we start with its cores and utilize two simple analytical models that

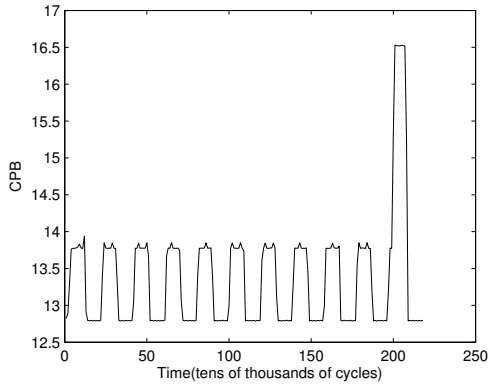

Fig. 5. Cycles Per Bundle (CPB) for AC3 application.

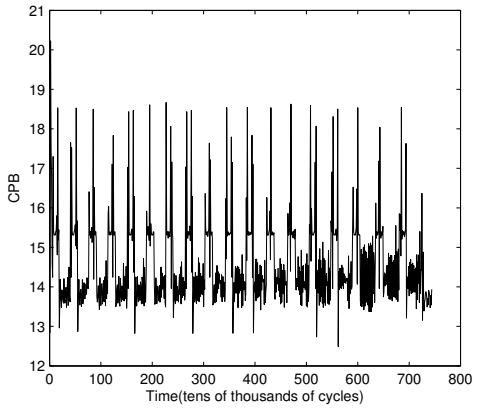

Fig. 6. Cycles Per Bundle (CPB) for CJPEG application.

can accurately predict the performance and energy impact of Dynamic Voltage Frequency Scaling (DVFS) across a wide range of voltage and frequency points. Both models require minimal input. The first is a stall-based model that is fed by an approximation of the stall cycles that are experienced by the processor due to the performance-critical off-chip load accesses. The second, is a miss-based model that improves on this approximation using as input the occupancy of the L2's miss-handling registers (MSHRs). Our experimental results using the SPEC2K suite show 2.1% (stall-based model) and 0.2% (miss-based model). The highly accurate predictions provided by our models can be used in various run-time power optimizations concerning the reconfiguration of the memory hierarchy.

We do not yet have the infrastructure in place to use these models on our ρ-VEX architecture, so we validate these in real machines by performing accurate power and performance measurements on state-of-the-art Intel's i7 processors. Intel Core i7 is a quad-core CMP. Each core supports hyperthreading execution. The Intel Core i7 family is enhanced with a special power-aware feature, called Speedstep technology, which allows run-time voltage and frequency scaling between 9 different steps, from 1.6 to 2.66GHz (i7 920). Furthermore, this core supports various idle states, called Cstates, in which

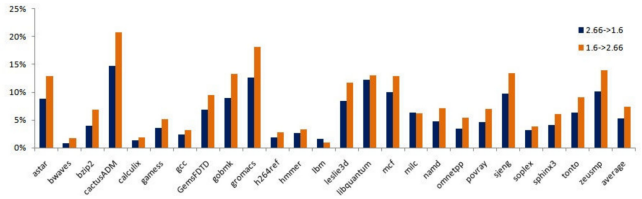

Fig. 7. Error in predicting execution time using the stall based model.

it is possible to completely deactivate the clock and cut-off the power supply for a combination of cores to reduce static and dynamic power consumption. Unfortunately, the Intel Core i7 has no performance counter to account for stalls due to LLC non-overlapping misses. In other words, there is no specific performance counter to measure the Memory-Level Parallelism, and we could not use our highly accurate, miss-based model. Therefore, we use an approximation of the stall-based model in order to predict the performance under different frequency points.

The Intel Core i7 comprises of two main voltage islands: core (execution- and fetch units, OoO and paging logic, L1/L2 caches and branch prediction) and the uncore (L3 caches, memory controller and QPI). In order to isolate the core and the uncore power, we compute core power dissipation by directly measuring voltage and current from the off-chip voltage regulator (ADP4000) residing in the motherboard by identifying two pins of interest: the pin that supplies the voltage to the core and the pin monitoring the total output current of the regulator. By hacking the motherboard (connecting wires to these pins) we were able to measure power using an oscilloscope while the processor was under normal operation. We use a sampling period of 10ms (our target is to provide OS-level optimizations so finer granularities will not provide useful results). The power measurements can be easily fed to the kernel OS using DLP-IO8, a USB analog–to–digital converter. Our future work includes to utilize this information (in the kernel level) to drive application/OS-driven DVFS policies.

Figure 7 shows the absolute error of predicting the execution time using our stall-based model in the Intel Core i7 for a large frequency step: running the program in the nominal frequency and predicting the execution time in the minimum frequency (black bars) – the orange bars represent the reverse scenario. For space reasons, for the benchmarks with multiple inputs, we present in Figure 7 the average error over all inputs. To further

TABLE II
THE INTEL I7 CORE HARDWARE EVENTS SELECTED FOR THIS WORK.

Performance Counter	Description
UOPS_EXECUTED.CORE_STALL_CYCLES	cycles no insts are executed in the processor
L2_RQSTS.LD_MISS	load requests that missed L2 cache
LLC_MISSES	last level cache misses
BRANCH_MISSES.RETIRED	mispredicted branches
UOPS_EXECUTED.PORT015	micro-ops executed in ports 0, 1 or 5
UOPS_EXECUTED.PORT234	micro-ops executed in ports 2, 3 or 4
UNHALTED_CORE_CYCLES	cycles core not halted

analyze the results, we classify the benchmarks into three categories: CPU-bound, memory-bound, and intermediate or mixed category. This categorization is performed as follows: when the frequency is scaled from 2.66 to 1.6 GHz, a purely CPU-bound program will suffer an increase in its execution time of 66.67%, but due to memory accesses this penalty will be smaller. Based on this, a program with performance penalty of more than 55% (scaling the frequency from max to min) is CPU-bound, a program with penalty less than 35% is memory bound while the rest of the benchmarks fall into the intermediate category. In general, the more memory bound a program is, the more the increase in the prediction error. This is an inherent property of the stall based model, since this model ignores the ROB-fill effect.

IX. Conclusions

In this paper, we presented the ERA project that addresses two intertwined problems in the design of embedded systems: higher performance and lower power consumption. We address these problems by investigating hardware and software design aspects of dynamic reconfiguration. The project addresses many issues at the same time and has been running just over a year.

We presented some of the early results that were achieved. First, we identified several benchmarks that should benefit from dynamic reconfiguration in the ERA platform, and we presented the initial results for a couple of them. The results show that applications have the potential for the utilization of reconfigurable features of the platform.

Second, we designed the initial (non-optimized) version of the ρ-VEX processor that constitutes the core processing component of the ERA platform, measured its performance, and compared the results to the performance of the Microblaze processor (both the speed-optimized and area-optimized versions). The results show that the performance is on par (from slightly slower to 3 times faster), but we utilize much more area resources. This is expected and due to the fact that the Microblaze is much more optimized in resource utilization. Currently, we are working on area- and performance optimizations of the ρ-VEX processor in order to decrease its required FPGA resources.

Acknowledgment

This work is supported by the European Commission in the context of the ERA (Embedded Reconfigurable Architectures) collaborative project #249059 (FP7).

References

[1] (2010) ERA – Embedded Reconfigurable Architectures. [Online]. Available: http://www.era-project.eu/

[2] S. Wong, T. van As, and G. Brown, "ρ-VEX: A Reconfigurable and Extensible Softcore VLIW Processor," in *Proceedings IEEE International Conference on Field-Programmable Technologies (ICFPT08)*, Dec 2008, pp. 369 – 372.

[3] J. Fisher, P. Faraboschi, and C. Young, *Embedded computing: A VLIW Approach to Architecture, Compilers and Tools*. Morgan Kaufmann Pub, 2004.

[4] J. Lau, S. Schoemackers, and B. Calder, "Structures for Phase Classification," in *ISPASS '04: Proceedings of the 2004 IEEE International Symposium on Performance Analysis of Systems and Software*, Washington, DC, USA, 2004, pp. 57–67.

[5] N. Puzovic, S. A. McKee, R. Eres, A. Zaks, P. Gai, S. Wong, and R. Giorgi, "A Multi-Pronged Approach to Benchmark Characterization," in *Proceedings of IEEE International Conference on Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS)*. IEEE, Sep 2010, pp. 1–4.

[6] K. Hoste and L. Eeckhout, "Microarchitecture-Independent Workload Characterization," *IEEE Micro*, vol. 27, no. 3, pp. 63–72, 2007.

[7] K. Beyls and E. D'Hollander, "Reuse Distance as a Metric for Cache Behavior," in *Proceedings of the IASTED Conference on Parallel and Distributed Computing and Systems*, vol. 14, 2001, pp. 350–360.

[8] M. Franklin and G. S. Sohi, "Register Traffic Analysis for Streamlining Inter-operation Communication in Fine-grain Parallel Processors," in *MICRO 25: Proceedings of the 25th Annual International Symposium on Microarchitecture*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1992, pp. 236–245.