Reconfigurable processors for handhelds and wearables: Application analysis

Rolf Enzler, Marco Platzner, Christian Plessl, Lothar Thiele, and Gerhard Tröster
Swiss Federal Institute of Technology (ETH) Zurich, CH-8092 Zurich, Switzerland

ABSTRACT
In this paper, we present the analysis of applications from the domain of handheld and wearable computing. This analysis is the first step to derive and evaluate design parameters for dynamically reconfigurable processors. We discuss the selection of representative benchmarks for handhelds and wearables and group the applications into multimedia, communications, and cryptography programs. We simulate the applications on a cycle-accurate processor simulator and gather statistical data such as instruction mix, cache hit rates and memory requirements for an embedded processor model. A breakdown of the executed cycles into different functions identifies the most compute-intensive code sections – the kernels. Then, we analyze the applications and discuss parameters that strongly influence the design of dynamically reconfigurable processors. Finally, we outline the construction of a parameterizable simulation model for a reconfigurable unit that is attached to a processor core.

Keywords: benchmarking, handhelds, wearables, processor simulator, dynamically reconfigurable processor

1. INTRODUCTION
The vision of reconfigurable computing is to combine the flexibility of programmable processors with performance characteristics of dedicated hardware. The first systems built in the early 1990s were composed of SRAM-based field-programmable gate arrays (FPGAs). These systems were quite loosely coupled to a host’s external memory or I/O buses, e.g. the PCI bus. Since then, reconfigurable units and processors have been coupled tighter in order to minimize the communication bottleneck. The current trend goes toward so-called hybrids that integrate a processor core with a reconfigurable unit or array on a system-on-chip. A selection of commercial hybrids includes Triscend’s E5 and A7, Chameleon Systems’ CS2000, Altera’s Excalibur, and Atmel’s FPSLIC. Examples for research hybrids are Garp,1 NAPA,2 REMARC,3 MorphoSys,4 Chimaera5 and OneChip.6 The variety of existing research processors points to the complexity that the design of dynamically reconfigurable processors entails. A multitude of design parameters exists that characterize the reconfigurable unit and its integration with the processor core.

This paper presents work in progress of the ZIPPY project at the Swiss Federal Institute of Technology (ETH) Zurich. ZIPPY’s mission is to develop reconfigurable processor technology for the domain of handheld and wearable computing and, specifically, to investigate new trade-offs between performance, power consumption, and system cost. While many of today’s research efforts target general-purpose computing, ZIPPY intends to design a domain-specific, hybrid, dynamically reconfigurable processor. Although reconfigurable hardware might be considered to be truly generic, research shows that the achieved performance strongly depends on the application. Therefore, we believe that a domain-specific approach to design a reconfigurable processor is preferable.

As a consequence, ZIPPY’s design methodology consists of two key components: (1) a set of applications representing the targeted domain of handheld and wearable computing, and (2) a parameterizable model of the reconfigurable processor. Our starting point is a set of complete applications that characterize the handheld and wearable application domain. By means of a processor simulation environment, we investigate the characteristics of this application set. The insights of the application analysis are used to develop a parameterizable processor model. In this way, the processor architecture can be evaluated and optimized according to the targeted application mix.

In this paper, we present our application set-up and results from simulation and analysis. Section 2 surveys related work in benchmarking. Section 3 describes the investigated application mix, discusses the used simulation environment, and presents the results of the application analysis. Section 4 discusses the main design parameters of hybrid reconfigurable processors. Finally, Sect. 5 concludes the paper and lists further work.

Correspondence: Rolf Enzler, E-mail: enzler@ife.ee.ethz.ch. Supported by the Swiss Federal Institute of Technology (ETH), Zurich.
2. RELATED WORK IN BENCHMARKING

There exist several benchmarks for general-purpose computing of which the SPEC family is the most popular benchmark for desktop computing. Benchmarking embedded systems is generally more difficult than benchmarking general-purpose systems. This is mainly due to two reasons: First, system specialization leads to several application domains in the embedded market with quite diverse applications and constraints. The formulation of the one embedded benchmark would thus not be very meaningful. Second, compiler technology for embedded processors is far from being as mature as for general-purpose processors. Many embedded applications are still written in assembly language to exploit special features of the embedded processor at hand. A benchmarking procedure that relies exclusively on *out-of-the-box* code and thus includes the compiler quality in the evaluation does not reflect on the likely use-case for embedded processors.

**Kernel-oriented benchmarks**  Benchmarks for embedded processors usually focus on computation-intensive kernels rather than on full applications. The underlying assumption is that embedded applications spend most of their runtime executing these kernels. Then, the kernel performance is representative for the overall application. Examples for kernel-oriented embedded benchmarks are:

- **BDTImark** The BDTImark is a composite metric based on the performance of an embedded processor on a group of typical digital signal processing kernels. Examples for such kernels are digital filters, dot product, and fast Fourier transform. The benchmarking procedure specifies the functionality of the kernel but allows for assembly-code programming to apply processor-dependent optimizations.

- **EEMBC** The EDN Embedded Microprocessor Benchmark Consortium (EEMBC) targets benchmarking of embedded applications by using kernels of key algorithms. Key algorithms are being defined for several embedded domains such as automotive/industrial, consumer, networking, office automation and telecommunications markets. The benchmarking procedure is clearly defined and the results have to be certified by the EEMBC Certification Labs (ECL).

**Application-oriented benchmarks**  In the last years, collections of complete applications have been used to model workloads in multimedia and communications. These benchmarks are oriented towards SPEC and target high-performance processors with multimedia instruction set extensions and VLIW architectures. Examples for these application-oriented benchmarks are:

- **MediaBench** MediaBench intends to capture the characteristics of today’s embedded multimedia and communication applications. The MediaBench suite is composed of 19 programs selected from publicly available image processing, communications, and DSP applications.

- **CommBench** CommBench was set up for evaluating and designing telecommunications network processors. The CommBench suite contains eight programs, four of them oriented toward packet header processing and four oriented toward processing of data streams.

**Benchmarks for reconfigurable computers**  Recently, benchmarks specifically for the area of reconfigurable computing have emerged. Two examples are:

- **ACS** The Adaptive Computing Systems (ACS) benchmark suite is thought for evaluating a configurable computing system’s architecture and tools. In contrast to traditional approaches that use functional benchmarks, ACS proposes so-called Stressmarks, benchmarks that focus on a specific characteristic of a configurable system such as versatility, capacity, timing sensitivity, scalability, and interfacing.

- **RAW** The Reconfigurable Architecture Workstation (RAW) benchmark suite consists of twelve programs representing general-purpose algorithms including sorting, matrix operations, combinatorial search, and graph problems. Each benchmark can be parameterized to derive instances of different problem sizes.
Benchmarks for handhelds and wearables  Currently, there exist no widely-accepted benchmarks for embedded systems targeting the application domain of handheld and wearables. The existing MediaBench and CommBench come close to this domain from a functional point of view. However, they do not explicitly target embedded systems or even reconfigurable systems. ACS and RAW are explicitly designed for evaluating the performance of a given reconfigurable system in various aspects, but they do not focus on our application domain. In summary, these benchmarks are not well-suited for our purpose of determining the optimal reconfigurable architecture for a specialized domain.

A typical workload for handheld and wearable computers emphasizes applications from multimedia, cryptography, and communications. As MediaBench and CommBench both cover parts of these domains, we select programs from these benchmark suites and augment them with the upcoming cryptography standard Rijndael and the text-to-speech synthesizer SVOX. Overall, this results in 29 programs that form the application pool for the following analyses. For brevity, we denote this set of programs as MCCmix. We plan to add further applications to this pool from high-speed graphics, speech recognition, and communication functions for short-range ad-hoc wireless networks such as BlueTooth.

To investigate how reconfigurable units improve the performance of a hybrid processor, we will integrate a model of the reconfigurable unit with a CPU simulator. This simulation model allows for high-level architectural exploration. We intend to use hand-crafted code fragments utilizing the reconfigurable unit to speed up the hot spots of the applications. The application programs written in C will be linked with a library of such code fragments. For our initial benchmarking effort, we therefore concentrate on complete applications and out-of-the-box code, i.e. unoptimized C code that has not been target-specifically optimized.

3. APPLICATION ANALYSIS

3.1. Application Pool

The following list outlines the investigated MCCmix organized in the three classes multimedia, cryptography, and communications. Many of these applications actually consist of two programs: encoding and decoding or encryption and decryption, respectively.

**Multimedia**
- JPEG\textsuperscript{10,11}: Lossy image compression
- EPIC\textsuperscript{10}: Lossy image compression based on wavelets
- MPEG2\textsuperscript{10}: Lossy video compression
- ADPCM\textsuperscript{10}: Adaptive differential pulse code modulation
- GSM\textsuperscript{10}: European standard for speech transcoding
- G.721\textsuperscript{10}: CCITT voice compression
- RASTA\textsuperscript{10}: Speech recognition
- SVOX\textsuperscript{14}: Text-to-speech synthesizer

**Cryptography**
- Rijndael\textsuperscript{15}: Advanced Encryption Standard (AES) crypto algorithm
- PGP\textsuperscript{10}: Crypto program based on IDEA and RSA algorithms
- Pegwit\textsuperscript{10}: Elliptic curve crypto algorithm
- CAST\textsuperscript{11}: DES-like crypto algorithm

**Communications**
- DRR\textsuperscript{11}: Deficit round robin fair scheduling algorithm
- FRAG\textsuperscript{11}: IP packet fragmentation with checksum computation
- REED\textsuperscript{11}: Reed-Solomon Forward Error Correction
- RTR\textsuperscript{11}: Radix-tree routing table lookup program
- ZIP\textsuperscript{11}: Data compression based on the Lempel-Ziv (LZ77) algorithm

Most of the applications of the MCCmix are also part of the MediaBench and CommBench suites. More details on these applications can be found in the referenced literature. Besides these applications, MCCmix contains the two new applications Rijndael and SVOX.

Rijndael is the winner of the Advanced Encryption Standard (AES) effort initiated by the U.S. National Institute of Standards and Technology (NIST) in order to find a successor for the Data Encryption Standard (DES).\textsuperscript{15}
Therefore, Rijndael can be expected to soon become an extremely popular cryptography algorithm. Rijndael is a block cipher with variable block length and key length. It is stated that Rijndael can be implemented very efficiently on a wide range of processors as well as in hardware.

SVOX\textsuperscript{14} is a commercial text-to-speech program for the German language and consists of three steps: word and sentence analysis, prosody control, and voice synthesis. The word analysis step applies phonetic lexica to yield the phonetic representation of each single word in the text. Prosody control then determines speech melody and sound durations based on statistical models, i.e. neural networks and others. Finally, the speech signal is synthesized by concatenation of small units extracted from natural human speech. SVOX applies special signal processing methods to modify these units such that they achieve the previously determined pitch and duration values.

3.2. Simulation Environment

We use the simulation tool set SimpleScalar\textsuperscript{16} to investigate the characteristics of the MCCmix applications. SimpleScalar is a publicly available tool set offering an environment for detailed simulations of modern microprocessors. The simulated instruction set bases on the MIPS/DLX instruction set architecture.\textsuperscript{17} SimpleScalar provides six simulators that vary in their detail vs. performance focus, ranging from a fast functional simulator to a cycle-accurate simulator with a configurable processor architecture. A key feature of SimpleScalar is its extensibility, which allows to adapt or extend the simulated processor architecture, as well as to derive customized simulators. In our analysis effort, we used the simulators sim-profile and sim-outorder.

- sim-profile is a simulator intended to gather profiling data of the application under investigation. Sim-profile does not consider a specific processor architecture, i.e. the generated profiling data is independent from the processor model.

- sim-outorder is a cycle-accurate simulator that allows to gather detailed execution statistics of out-of-order issue, superscalar processors. The underlying processor model can be configured with parameters such as number of scalar units, branch predictor, memory hierarchy, cache sizes and modes, memory access latency, bus widths, etc. Sim-outorder does not simulate an operating system, but transfers all operating system calls of the simulated applications to the simulator’s host operating system.

For the processor dependent simulations, sim-outorder’s processor model was configured to roughly resemble modern embedded processors such as TinyRISC or StrongARM. The used processor model has the following features:

<table>
<thead>
<tr>
<th>Scalar units</th>
<th>1 integer ALU, 1 dedicated integer multiplier/divider, 1 floating-point ALU, 1 dedicated floating-point multiplier/divider</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction issuing</td>
<td>in-order</td>
</tr>
<tr>
<td>Branch predictor</td>
<td>always ‘not-taken’</td>
</tr>
<tr>
<td>Caches</td>
<td>one level of cache</td>
</tr>
<tr>
<td></td>
<td>32-way set-associative 16 KB instruction cache</td>
</tr>
<tr>
<td></td>
<td>32-way set-associative 16 KB data cache</td>
</tr>
<tr>
<td></td>
<td>both with first-in first-out replacement strategy</td>
</tr>
<tr>
<td>Memory interface</td>
<td>1 system port to the CPU (4 bytes wide) access latency of 8 cycles for the first chunk, 2 cycles for the rest of a burst access</td>
</tr>
</tbody>
</table>

3.3. Application Characterization

We have simulated all applications described in section 3.1 using SimpleScalar. The runtime to process the original test data sets delivered with MediaBench and CommBench is quite different. Therefore, we have chosen our own input data sets, scaled such that each single simulation takes about 2 hours of host CPU time. For reference we have simulated SPECint95 using the provided test input files.
3.3.1. Instruction class mix

The instruction class mix characterizes applications according to the frequency of different instruction types during execution. These frequencies depend only on the application, the compiler, and the processor’s instruction set, but not on architectural parameters of the processor such as the number of execution units, cache sizes, etc. By means of sim-profile the instruction profile of each MCCmix application was generated revealing how many times each instruction was issued during execution. The instructions were then classified according to their functionality into the eight instruction classes:

- **load**, **store**, **branch**, **integer arithmetic**, **logic**, **shift**, **floating-point arithmetic**, and **other**.

Fig. 1 depicts the distribution of the instruction classes over the MCCmix applications. Figures 2(a)–2(h) show the results in more detail by displaying the frequencies for each instruction class individually. The MCCmix applications are prefixed with **mm** for multimedia, **cr** for cryptography, and **co** for communication applications. The averages of each class and the corresponding averages of the SPECint95 benchmark are also shown in the graphs. The comparative analysis between the applications of the MCCmix leads to following results:

- **Multimedia** applications require significantly more integer arithmetic instructions compared to the MCCmix average which is evidenced by Fig. 2(d). That is not astonishing since most multimedia applications perform arithmetic intensive signal processing. Fig. 2(e) shows that logic operations are used rarely.

- **Cryptography** algorithms are characterized by very regular code operating on bit-level data, which is emphasized by Fig. 2(c), 2(e) and 2(f). The regularity of these algorithms reflects in a small number of branch instructions, shown in Fig. 2(c). Since the instruction set under consideration does not provide special instructions for bit manipulation, the bit-level operations are performed using repeated shift and logic operations, as given by Fig. 2(e) and 2(f).

- **Communication** applications are characterized by using a high number of branch instructions, shown in Fig. 2(e). This results from the control-flow dominated nature of communication applications. Arithmetic operations are of less significance for these applications.

These results underline the importance of differentiating applications into the three classes multimedia, cryptography, and communications. Each group has its own characteristics and hence emphasizes different architectural features of a processor. Another interesting observation is that only 6 out of the 29 programs require floating-point operations. Given the fact that these results were generated with out-of-the-box code, i.e. without any hand
Figure 2. Instruction class mix split into the individual classes with averages for MCCmix and SPECint95.
optimization, we can expect optimized programs to use floating point even less. Comparing the overall results of MCCmix to SPECint95, we observe major differences. The most significant differences are related to the instruction classes store (MCCmix: 5.97% vs. SPEC: 16.7%), logic (MCCmix: 7.65% vs. SPEC: 1.67%), and shift (MCCmix: 11.88% vs. SPEC: 5.14%). A surprising result is the small percentage of store instructions in the MCCmix.

3.3.2. Memory requirements

Fig. 3 shows the average memory requirements of the MCCmix, the individual application groups of the MCCmix (multimedia, cryptography, communications), and SPECint95. Program text refers to the size of the executable code, data size is the size of data (data and bss segment), where the initialized data section is also incorporated in an application’s executable. Dynamic memory is the amount of memory that is allocated by the application at runtime. This number has to be treated as an upper bound since the SimpleScalar simulators do not report on memory deallocation.

The MCCmix applications have rather small memory requirements. Fig. 3 shows that the data memory requirements of SPECint95 exceeds the requirements of the MCCmix by a factor of 22. If the RASTA benchmark is omitted from the MCCmix, even a factor of 125 results. The average program text of SPECint95 is four times larger than the average program text of MCCmix.

3.3.3. Instruction and data cache hit rates

We gathered the hit rates for instruction cache and data cache using sim-outorder configured with the processor model outlined in Sect. 3.2. Fig. 4 and Fig. 5 show the results. The MCCmix has higher hit rates for both instruction and data cache, but for the instruction cache the difference is more significant. This confirms the assumption that many embedded applications spend most of their runtime in rather small code sections, the so-called kernels. Fig. 6 shows the code sizes of these kernels. The MCCmix applications contain compute-intensive code parts with high locality. Contrary to general-purpose processors, processors for the MCCmix application domain can work efficiently with rather small instruction caches.

3.3.4. Kernel identification

In order to investigate the structure of the MCCmix and SPECint95 programs, we generated a function breakdown for each application. By means of a Perl program, we merged address information from disassembled binaries with runtime information delivered by sim-outorder to compute the execution time of each function. The function breakdown lists all functions of an application together with the percentage of the overall application runtime. Fig. 7 shows the function breakdown of the MCCmix and SPECint95 applications. The two left-most segments of each bar represent the shares of the two most compute-intensive program functions. The third segment represents the merged shares of the remaining functions. For example, the dominant function of the application DRR is schedule, the most dominant functions of MPEG2 encode is dist1.
The function breakdown facilitates to identify the compute-intensive parts of the applications, the kernels. Fig. 7 expresses that MCCmix and SPECint95 differ strongly. The average share of the most compute-intensive function for MCCmix is 59%, compared to only 19% for SPECint95. This proves that the MCCmix applications are kernel-oriented, while the general-purpose SPECint95 applications consist of many smaller-sized functions. This quantitative information about the MCCmix kernels is essential for the subsequent design of a reconfigurable processor. Kernels form the greatest potential for improvements in performance and power consumption.

### 3.3.5 Conclusions

The application analysis served to gather quantitative data that will drive the design and evaluation of a reconfigurable processor. The results confirm the strategy of selecting such a large number of applications for analysis. The major qualitative conclusions are:

- The MCCmix applications are strongly kernel-oriented – in contrast to general-purpose benchmarks such as SPECint95.
- The memory requirements for the MCCmix applications are one to two orders of magnitude lower than for SPECint95.
- The MCCmix applications achieve a better cache performance than SPECint95 with a rather small instruction cache.
- MCCmix consists of applications from three quite differing groups: multimedia applications are dominated by integer arithmetic, cryptography applications rely on logic and shift operations, and communication applications require many branches.
Figure 7. Function breakdown based on executed cycles for MCCmix and SPECint95. The two left-most segments of each bar represent the shares of the two most compute-intensive program functions. The third segment represents the merged shares of the remaining functions. The average shares of the most compute-intensive functions for MCCmix and SPECint95 are also given.

4. FROM KERNELS TO RECONFIGURABLE UNITS

4.1. Parameters of Application Kernels

The analysis in the previous section confirms that our application mix is strongly kernel-dominated. Consequently, these kernels must be executed efficiently to achieve high performance. The ZIPPY project investigates reconfigurable units integrated with CPU cores to efficiently accelerate kernel execution. The design of the reconfigurable processor entails many design parameters. The corresponding design decisions should ideally be driven by application characteristics. Some of these characteristics can be derived from the simulative analysis discussed in the previous section. Although this is an important first step, it is not sufficient. Simulation statistics and kernel parameters are derived from sequential C code which is targeted to von-Neumann execution models. The major strength of reconfigurable systems is computing in space, massively utilizing parallel resources. This is in sharp contrast to the sequential model of computing in time.

Parallelism is a central kernel parameter that must be extracted by a data-flow analysis. Parallelism involves three issues: type, level, and amount of parallelism. Types of parallelism are SIMD, pipelining (systolic), and MIMD. Parallelism can be found on several hierarchical levels. For example, many operations in image compression...
can be parallelized on macro block, slice, or frame level. The amount of parallelism on each level gives the number of computing elements that can be employed. The required operation types are given by instruction class profiles gathered from C code. The granularities can only be estimated, as processors have fixed operand widths. Applications show typical data access patterns. Multimedia applications, for example, often work on streams and access data continuously.

4.2. Architectural Parameters of Dynamically Reconfigurable Processors

The design of a reconfigurable processor consists of two parts: the design of the reconfigurable unit and the integration of the reconfigurable unit with the CPU core. We consider the reconfigurable unit to consist of an array of logic elements with an interconnect and embedded storage resources. For efficient operation, the parameters of the reconfigurable unit should match the kernel parameters.

- **Operators and Granularity** Choosing operators and granularity involves a trade-off. Reconfigurable elements are classified into fine-grained and coarse-grained. Fine-grained units use logic elements with 2–4 bit inputs and single flip-flops and resemble current FPGA architectures. Such structures have shown to be well-suited for bit-manipulation and random logic. Coarse-grained architectures accommodate 8–16 bit ALUs and registers and are better suited for regular arithmetic operations on byte and word sized data. A parameter strongly related to granularity is configuration size. Given a certain silicon area for the reconfigurable unit, one can implement many fine-grained elements or less coarse-grained ones. A large number of fine-grained elements requires more configuration data than a smaller number of coarse-grained elements.

  The operators should clearly match the kernel’s requirements. Hence, operators for integer arithmetic, logic, and shift operations must be provided. Communication kernels require many branch instructions. Mapped to the reconfigurable array, these kernels require random logic. To support random logic efficiently, fine-granular operators are required.

- **Interconnect** Logic elements are usually placed in a 2-D array. The simplest scheme connects each element to its four neighbors horizontally and vertically. Additional buses may exist that connect all elements in a row and in a column. Sometimes interconnects are additionally hierarchically structured.

  The design of the interconnect is mainly driven by the type and level of parallelism exploited and the data transfers required. For our application mix, we can expect mostly SIMD and pipelining (systolic) types of parallelism. An open issue is the interconnect between logic elements and storage resources. Our analysis shows that the applications are load-dominated rather than store-dominated. Corresponding prefetching mechanisms will therefore be investigated.

- **Reconfiguration** The data required to configure the reconfigurable unit is called context. Main parameters are the size of a context and the time it takes to configure the unit. The reconfiguration time depends on the configuration size and the location from where the configuration data has to be read. Multi-context devices are able to store several contexts in a context memory on-chip.

  As we design dynamically reconfigurable processors, the clear design goal is single-cycle reconfiguration, i.e. the whole reconfigurable unit is reprogrammed in a single clock cycle. The size of a context involves a trade-off. Greater functionality and number of logic elements and interconnect increase the context size, but allow for more flexibility and exploitation of parallelism at a higher degree. The number of contexts stored on chip and the way contexts are loaded depend strongly on the number of context switches.

  The integration of the reconfigurable unit and the CPU core leads to further design decisions in coupling, instruction set extension, and data transfer. The coupling is the relative position of processor core and reconfigurable unit. Generally, a tighter coupling leads to a smaller communication overhead. Couplings can be classified into three categories. In case of attached processing units the reconfigurable unit is located outside the processor and is connected to a memory or I/O bus. In coprocessor coupling the reconfigurable unit is part of the processor and is placed next to the processor core. Examples are Garp, NAPA, REMARC, and MorphoSys. Reconfigurable functional units (RFUs) are integrated into the processor core similar to any other functional unit. Examples are OneChip, and Chimaera.
Both RFUs and coprocessors extend the core’s instruction set with customized instructions. RFU approaches use instructions to initiate reconfiguration of the RFU and to actually execute the RFU function. Coprocessors additionally include data transfer and synchronization instructions. Synchronization is required whenever two computing elements operate concurrently. RFUs can operate concurrently to other functional units, because the core’s control logic synchronizes activities and controls access to the register file. Simple approaches for coprocessors force the core to stall until coprocessor execution has completed. More advanced techniques allow concurrency and synchronize by semaphore-like mechanisms. RFUs access the core’s register file to read and write data. Coprocessors can use several options. First, data may be transferred via dedicated coprocessor registers. Second, coprocessors can access the same memory hierarchy as the core. Third, to increase the overall bandwidth the reconfigurable units can be equipped with dedicated memory ports. While latter certainly increases bandwidth, it can also lead to data consistency problems.

5. FURTHER WORK

The ZIPPY project follows a systematic design methodology to investigate architectures for dynamically reconfigurable processors for the domain of handhelds and wearables. Starting from a set of complete applications, benchmarking is performed by using a cycle-accurate processor simulator, outlined in Fig. 8(a). Parameters such as instruction class mix, memory requirements, and instruction and data cache hit rates are derived. Furthermore, the most compute-intensive parts of the applications, the kernels, are identified. The next steps in the ZIPPY project are:

- The number of benchmarks will be reduced to get a smaller set of applications for further simulations. This compact application mix will still cover the requirements of multimedia, cryptography and communication applications.
- We will develop a behavioral, high-level model of a reconfigurable array that matches the characteristics of our handheld and wearable workload. The model will be parameterizable in terms of number and functionality of the logic elements, interconnection network, number of contexts, integrated memory blocks, etc. to allow for early design space exploration. To enable co-simulation of the reconfigurable array and the CPU core, we plan to integrate the model into the SimpleScalar simulation environment as shown in Fig. 8(b).
- In order to gain information about power dissipation and system cost, we will integrate a power simulator and a chip-area estimator in the simulation process.
The overall simulation environment will facilitate the evaluation of alternative reconfigurable processor designs and to find an optimal architecture for the targeted application mix. As outlined in Fig. 8(b), the results from simulation influence the design parameters for the reconfigurable processor and the programming model. The programming model will be kept simple and will consist of parameterizable hand-crafted library functions. The problem of finding an optimal architecture is a multi-objective optimization problem. We also plan to attack this problem with a more formal approach by using stochastic search procedures, e.g. genetic algorithms, to find Pareto-optimal design points.  

REFERENCES