Mighty-Morphing Power-SIMD

Ganesh Dasika[1], Mark Woh[1], Sangwon Seo[1], Nathan Clark[2], Trevor Mudge[1], and Scott Mahlke[1]

[1]Advanced Computer Architecture Laboratory
University of Michigan - Ann Arbor, MI
{gdasika, mwoh, swseo, tnm, mahlke}@umich.edu

[2]College of Computing
Georgia Institute of Technology - Atlanta, GA
{nate.clark}@gatech.edu

ABSTRACT

In modern wireless devices, two broad classes of compute-intensive applications are common: those with high amounts of data-level parallelism, such as signal processing used in wireless baseband applications, and those that have little data-level parallelism, such as encryption. Wide single-instruction multiple-data (SIMD) processors have become popular for providing high performance, yet power efficient data engines for applications with abundant data parallelism. However, the non-data-parallel applications are relegated to a low-performance scalar datapath on these data engines while the SIMD resources are left idle. To accelerate both types of applications, we propose the design of a more flexible SIMD datapath called *SIMD-Morph*. In SIMD-Morph, code with data-level parallelism can be executed across the lanes in the traditional manner, but the lanes can be morphed into a feed-forward subgraph accelerator to execute scalar applications more efficiently. The morphed SIMD lanes form an accelerator that exploits both instruction-level parallelism as well as operation chaining to improve the performance of scalar code by exploiting the available resources in the SIMD lanes. Experimental results show that the performance impact is a 2.6X improvement for purely non-SIMD applications and a 1.4X improvement for the non-SIMD-ized portions of applications with data parallelism.

Categories and Subject Descriptors

C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures—*Single-instruction-stream, multiple-data-stream processors (SIMD)*

General Terms

Design, Performance

Keywords

Operation chaining, SIMD processing, Data-level parallelism, Instruction-level Parallelism

CASES'10, October 24–29, 2010, Scottsdale, Arizona, USA.

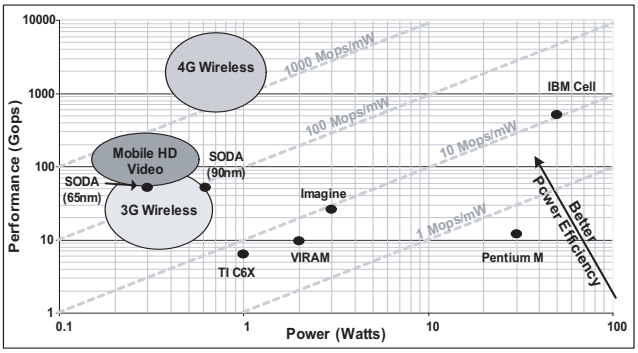

Figure 1: Performance vs. power requirements for various mobile computing applications.

1. INTRODUCTION

In the coming years, the deployment of mobile computers will continue to skyrocket. The prime example today is the smart phone, but in the near future we expect to see the emergence of new classes of such devices. These devices will improve on the smart phone by incorporating advanced functionality such as high-bandwidth Internet access, human-centric interfaces with voice recognition, high-definition video coding, and interactive conferencing. While integrating new capabilities is important for attracting customers, battery lifetime and power dissipation remains paramount.

Untethered devices perform signal processing as one of their primary computational activities due to their heavy usage of wireless communication as well as rendering of audio and video signals. Fourth generation wireless technology (4G) has been proposed by the International Telecommunications Union to increase the bandwidth to maximum data rates of 100 Mbps for high mobility situations and 1 Gbps for stationary and low mobility scenarios like Internet hot spots [23]. This translates to an increase in the computational requirements of 10-1000x over previous third generation wireless technologies (3G), with a power envelope that is limited to increasing only 2-5x [28].

Figure 1 presents the demands of 3G and 4G computing in terms of their peak processing throughput requirements and their power budgets. Conventional processors cannot meet these requirements; low-power laptop processors, such as the Pentium M, operate below 1 Mop/mW, while digital signal processors, such as the TI C6x, operate around 10 Mops/mW. Conversely, 3G wireless protocols, such as W-CDMA and 802.11a, require approximately 100 Mops/mW. The IBM Cell system can provide excellent throughput, but its power consumption makes it infeasible for mobile devices [21]. Research solutions, such as VIRAM [16] and Imag-

ine [1], can achieve the performance requirements for 3G, but exceed the power budgets of mobile terminals. SODA improves upon these solutions and delivers a solution for 3G wireless [18]. Companies such as Phillips [27], Infineon [3], ARM [29], and Sandbridge [11] also propose domain-specific systems that can support 3G.

The design of the next generation mobile platforms must address three critical issues: efficiency, programmability and adaptivity. The inherent computational efficiency of 3G solutions is insufficient and must be increased by at least an order of magnitude as shown in Figure 1. Straight-forward scaling of 3G solutions by techniques such as increasing the number of cores is part of the solution, but is not enough on its own. Programmability provides the opportunity for a single platform to support multiple applications and even multiple standards within each application domain. Further, programmability provides faster time to market as hardware and software development can proceed in parallel, the ability to fix bugs and add features after manufacturing, and higher chip volumes as a single platform can support a family of mobile devices. Lastly, hardware adaptivity is necessary to maintain efficiency as the core computational characteristics of the applications change. 3G solutions rely heavily on the vast amounts of vector parallelism in wireless signal processing algorithms, but lose most of their efficiency when vector parallelism is unavailable or constrained.

The AnySP architecture tackles the first two issues using a combination of wide-SIMD execution (64 lanes), efficient data shuffling, and support for common intrinsic functions [30]. Efficiency and programmability are simultaneously garnered by pushing SIMD execution to new levels, while applying domain specific customizations to the datapath. Adaptivity within vectorizable code is also supported by allowing neighboring lanes to conjoin, creating 32 lanes, each supporting a computation depth of two, or overlaying multiple narrow vector computation threads using independent address generation units. Researchers at ST Microelectronics apply similar generalizations to SIMD datapaths [20]. However, all of these solutions ignore non-SIMD-izable computation. Such code is relegated to execute on a low performance scalar pipeline provided in the design. For inherently scalar applications such as compression or encryption, high computational rates cannot be sustained. Even for highly vectorizable codes, Amdahl's Law will expose the non-vectorizable portions of the code as the eventual bottleneck.

One approach to accelerating the scalar code is to enhance the capabilities of the scalar pipeline. Integrating specialized functional units which more efficiently execute critical portions of an application's dataflow graph with instruction set extensions to utilize the new hardware is a popular approach for designing application specific instruction processors, or ASIPs. Several commercial tool chains design ASIPs with instruction set extensions, including ARM OptimoDE, and ARC Architect. However, this approach fails to take advantage of the vast hardware resources already present in the datapath, namely the SIMD execution units, which are mostly idle when scalar code is executing.

In this paper, we propose a dynamically changeable SIMD datapath, referred to as *SIMD-Morph*, that in the base mode can execute data-parallel code across all the SIMD lanes. However, for scalar code, the datapath is morphed into a subgraph accelerator similar to the configurable compute accelerator (CCA) [6]. A CCA consists of an array of simple functional units interconnected in a feed-forward manner. The CCA executes dataflow subgraphs identified by the compiler as atomic units. Acceleration is provided in two ways. First, instruction-level parallelism is exploited by concurrently executing independent operations in the subgraphs (horizontal compression). Second, operation chaining executes dependent operations in a single cycle by exploiting slack in the pipeline and eliminating conventional register forwarding (vertical compression).

```
1:  for( i = 0; i < length; i++ )
2:  {
3:      x = (unsigned char) ( x + 1 );
4:      a = m[x];
5:      y = (unsigned char) ( y + a );
6:      m[x] = b = m[y];
7:      m[y] = a;
8:      data[i] ^=
9:        m[(unsigned char) ( a + b )];
10:  }
```

Figure 2: A portion of the `rc4` benchmark

```
1:  for (j = 0;  j < data_blk_size;  j++)
2:  {
3:    a = crc_accum >> 24;
4:    b = *data_blk_ptr;
5:    c = a^b;
6:    i = c&0xff;
7:    d = crc_table[i];
8:    e = crc_accum << 8;
9:    crc_accum = e ^ d;
10:   data_blk_ptr++;
11: }
```

Figure 3: A portion of the `crc` benchmark

The challenges of SIMD-Morph that lead to the central contributions of this work are as follows:

- The extensions to the SIMD datapath to enable effective execution of a scalar subgraph where both horizontal and vertical compression are achieved.
- A design space exploration to define the organization, needed capabilities, and connectivity of the subgraph accelerator to maximize utilization of the available functional resources in the SIMD datapath.

2. MOTIVATION

As embedded devices become more and more pervasive, the types of computation they are required to perform becomes more varied. SIMD-ization is one of the most common architectural techniques used to improve the efficiency, but it is not effective for many styles of applications, e.g., some types of data encryption and compression, and many forms of error detection and correction.

To give a specific example, Figure 2 is the critical loop for the RC4 encryption algorithm, the basis of Secure Sockets Layer (SSL) and many other popular streaming, secure protocols.

Note that while the data being encrypted is accessed in a sequential order (line 8), the value being XORed with the data is the result of multiple indirect memory accesses to `m[y]`. This type of pointer-chasing is difficult to perform efficiently on SIMD accelerators, which are typically designed for contiguous accesses.

As another example, Figure 3 is the main loop from a cyclic redundancy check (CRC) algorithm, commonly used to detect errors in signal transmissions.

Similar to RC4, CRC has non-contiguous, pointer-chasing memory accesses (line 7). An additional challenge is that CRC also has a dependence, `crc_accum`, that crosses loop iterations thus preventing any significant form of data-level parallelism. Like RC4,

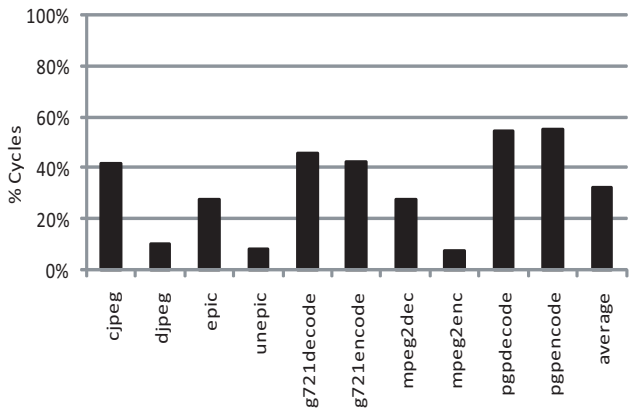

Figure 4: Fraction of cycles spent outside of inner-most loops

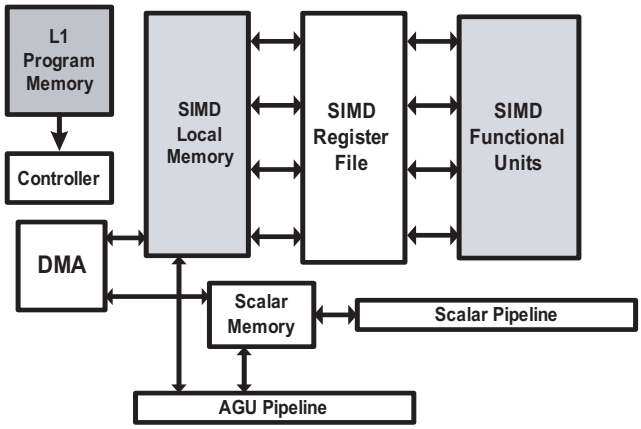

Figure 5: Baseline SIMD+Scalar Processor

CRC would garner little if any benefit from traditional SIMD accelerators. Both of these applications are very important to many embedded domains, and augmenting a SIMD datapath to better support them would prove very beneficial.

Even in applications that are amenable to SIMD acceleration, often times only their inner-most loops are data-parallel and can be SIMD-ized. SIMD datapaths are continually getting wider with Moore's law [7] but as this happens, the non-SIMD-ized portions of the application will begin to dominate execution time as per Amdahl's law.

Figure 4 illustrates the importance of this trend. This figure shows the percentage of time spent outside inner-most loops, i.e. executing non-SIMD-izable code for several SIMD-izable benchmarks from the MediaBench Suite [17]. From this figure we can deduce that even in an ideal system with an infinitely-wide SIMD machine, over 30% of the application's execution time is spent on non-SIMD-ized code. This limits the speedup of these applications to ≈3x in the best case, and clearly represents an important target for acceleration.

Several important application domains are not amenable to SIMD-ization, and even those that are have a significant fraction of non-SIMD-izable code that is important to accelerate. For these reasons, this paper answers the question, "How can the standard narrow-scalar-pipeline-with-wide-SIMD-pipeline architectures be modified to better accelerate a wider variety of applications?"

3. SIMD-MORPH

3.1 Hardware

The baseline SIMD architecture used is shown in Figure 5. This baseline is modified in the manner shown in Figure 6 to create the SIMD-Morph architecture. The 16 SIMD lanes are grouped in 4 Configurable Execution Groups (CEGs) of 4 elements each, named CEG0 (lanes 0 to 3) through CEG3 (lanes 12 to 15). The FUs in all the lanes are capable of executing the same operations as before but now each CEG has an added memory unit capable of executing scalar loads and stores. Each FU may receive its inputs from 8 possible sources: the 4 outputs of the previous CEG, any of the other 3 FUs in its row or from an 8-bit constant register (not pictured). In addition, the memory FUs may receive loaded data from memory.

Register data is transferred into the SIMD datapath via CEG0's 4 inputs directly from the scalar register file. Register data is transferred out of the SIMD datapath via CEG3's outputs. Despite the increased complexity of adding connections from the scalar register file to the SIMD datapath, this method is a better solution than adding extra ports to the SIMD register file. This is because the code that will require this functionality is not SIMD-parallel and would normally be executed on the scalar pipeline so adding these connections to the SIMD register file will require extra cycles to copy data between the two register files, thereby reducing performance.

Figure 7 shows a representation of lines 6-9 of the `crc` benchmark shown in Figure 3 on the SIMD-Morph hardware. The live-in values in this case are the variables `crc_accum`, `c` and the base pointer `crc_table`. This pointer is used to issue the load from `crc_table[i]` (line 7 in the code). The shaded nodes are idle and are not used for any computation. The live-out value is the updated value of `crc_accum`. The `mov` operations are used to transfer operands from where they are generated to where they are read.

3.2 Configuration

A control memory is required in order to store the various configuration bits required for SIMD-Morph. The requirements of the different components are broken down as follows:

- To specify opcodes, each element requires 5 bits to support the various arithmetic and logic operations. The memory units require an additional bit to support various load and store instructions. This is a total of 21 bits per CEG, or 84 bits total.

- Each FU has 2 ports, each of which can receive inputs from 1 of 7 possible sources (requiring a 3-bit mux per port). Further, 1 global bit is required to specify whether the FU receives its inputs in "SIMD-Morph mode" (from other FUs and the scalar register file) or in "normal mode" (from the SIMD register file). Each FU can also receive an 8-bit literal value as input. The total bits required to specify inputs is, therefore 225 bits (1 + 8*16 + 3*2*16).

- The very last row outputs back to the 2 write ports in the scalar register file. Each of these ports may receive its inputs from any of the 4 elements in CEG3, requiring 4 bits of control.

The total number of bits required to configure SIMD-Morph is, therefore, 225 + 84 + 4, or 313 bits.

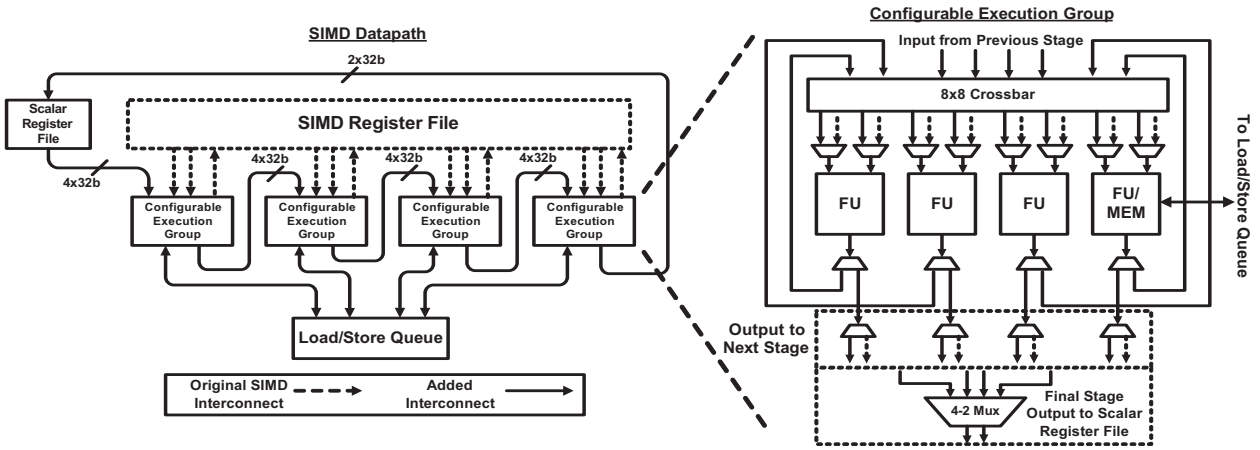

Figure 6: SIMD-Morph Modifications

3.3 Compilation

Effectively utilizing accelerators such as SIMD-Morph requires tool chain support, and so it is important to introduce the compilation strategy used during design space exploration. Compiling an application to make use of computation accelerators boils down to two steps: *enumerating* portions of the application's dataflow graph (DFG) that can be executed on the accelerator, and *selecting* which portions to accelerate.

"Enumeration" consists of generating a set of subgraphs from a given DFG, and determining if they can run on an accelerator. Generating a set of subgraphs is difficult because the number of possible subgraphs grows exponentially with the size of the DFG. Determining if the subgraphs can run on an accelerator, i.e., determining if they perform the same computation, is essentially equivalence checking, which is NP-complete. The problem is further complicated if the accelerators perform a superset of the desired computation (e.g., an accelerator for dot-products could also accelerate multiply-accumulates in an application).

"Selecting" which subgraphs to accelerate is also difficult. Typically, the selection problem is formulated to push as much computation as possible onto the accelerators, while ensuring that there is no overlap between subgraphs. That is, given a set of enumerated subgraphs, find the group that covers the largest portion of the DFG while minimizing the number of nodes appearing in multiple subgraphs. This problem is also NP-complete and is quite similar to the well known technology mapping problem in VLSI design. Clearly, mapping applications to subgraphs is a challenging compilation problem.

Previous work has shown that greedy solutions work poorly, particularly when the accelerator is large, like SIMD-Morph is [8]. For that reason, we leveraged a more thorough compilation approach very similar to previous work [8]. Essentially, the compiler performs an exhaustive search of the design space to enumerate and select the best possible set of subgraphs for acceleration. Several pruning heuristics keep the compilation time reasonable for the vast majority of cases, and timeouts prevent corner cases from taking an intractable amount of time. This more thorough compilation strategy ensures that the design space exploration is as accurate as possible.

3.4 Baseline Observations

In this work, benchmarks are generally classified into two categories. The "media" benchmarks are the same as those in Figure 4 and are from the Mediabench benchmark suite. The rest are from the MiBench [13], SPECINT2000 and NetBench [19] suites. These benchmarks were chosen as they were representative of applications that are normally run on mobile devices.

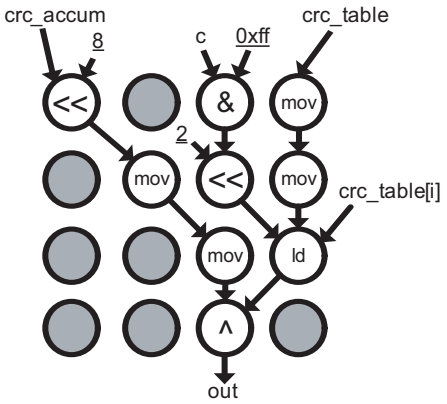

Figure 7: Graphical representation of a portion of the `crc` benchmark on SIMD-Morph

.

For the media benchmarks, the SIMD-izable inner loops are not considered as they are assumed to execute on a normally configured SIMD datapath. The other benchmarks demonstrate limited data-level parallelism and, therefore, the entirety of the benchmarks are considered for execution under SIMD-Morph.

Figure 8 shows the distribution of various sizes of subgraphs in each of the applications in increments of 4. The overwhelming majority of subgraphs, 83%, are between 1 and 4 operations. Approximately 16% of subgraphs have between 5 and 8 ops and just over 1% of subgraphs have more than 8 operations.

Factoring in the speedup contributed by each of these subgraphs presents only a slightly different story. In Figure 9, each segment shows the % of overall cycles saved came from subgraphs of various sizes. The speedup contribution of 1 to 4-operation subgraphs is 71.5% and that of 5 to 6-operation subgraphs is 26%. This indicates that while there while there are few subgraphs larger than 4 operations in size, their speedup contribution is significant.

Figures 10 and 11 illustrate the distribution of maximum depths of subgraphs and the speedups obtained by varying the subgraph

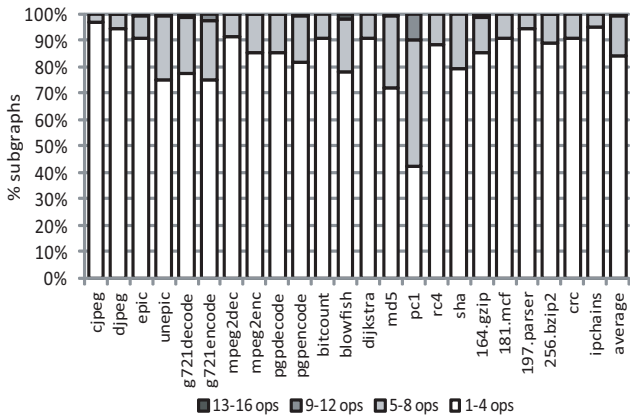

Figure 8: Distribution of subgraphs consisting of various numbers of operations

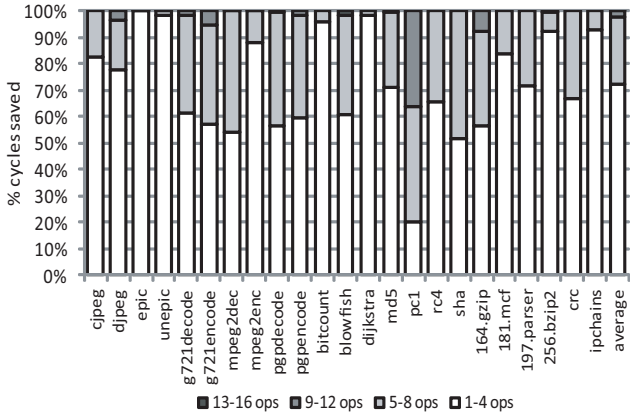

Figure 9: % Cycles saved by subgraphs consisting of various numbers of operations

depths. The subgraph depths for the benchmarks in consideration ranged from 1 operation to 5, although the execution frequency of the 5-deep subgraphs (there is one in each of `g721encode` and `g721decode`) was so small that it is not noticeable in these graphs.

On average, approximately 85% of the subgraphs have depths of 3 operations or fewer. The performance contribution paints a similar picture as well, with 80% of the performance improvement coming from subgraphs with depths of 3 operations or fewer.

Finally, Figure 12 shows the distribution of memory operations in the subgraphs. Over 90% of the subgraphs use only 2 of the 4 available memory units. However, over 60% of subgraphs have at least one memory operation, indicating that support for memory operations is still important, although it is unlikely that reducing the number of operations supported will have a significant impact on performance.

There is one salient point from all of these results: the baseline design is grossly over-provisioned. Further, the following may be considered:

- The maximum number of operations can be reduced, reducing the overall latency of SIMD-Morph

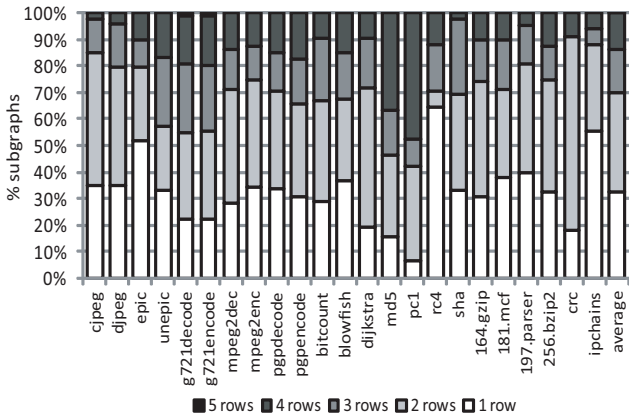

Figure 10: Distribution of subgraphs consisting of various depths of operations

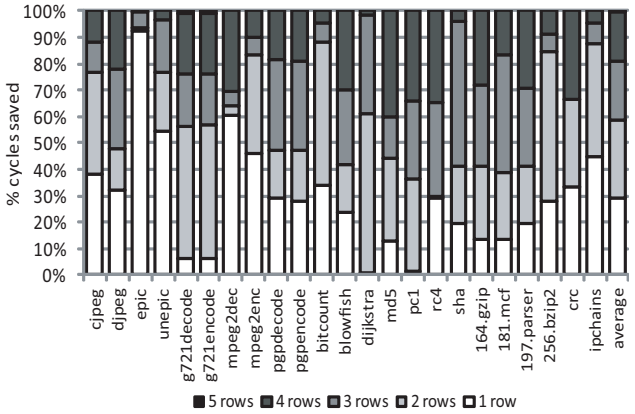

Figure 11: % Cycles saved by subgraphs consisting of various depths of operations

- Reducing the number of operations will also reduce the complexity of the interconnect network, further reducing latency and power
- Reducing the maximum depth of subgraphs by eliminating connections between some groups will prevent some subgraphs from executing but could potentially allow more, shallower subgraphs to execute. Increasing the number of subgraphs, however, could require increasing the number of live-in and live-out values
- Reducing the number of memory units will reduce the complexity of SIMD-Morph but will potentially severely limit the number of subgraphs that can be executed.

4. DESIGN-SPACE EXPLORATION AND RESULTS

The overall design-space was explored in order to best understand the bottlenecks of the design. The features that were varied include:

- The number of register inputs
- The number of register outputs

Configuration	#inputs	#outputs	#Mem	#Ops	Topology	Control bits required
baseline	4	2	4	16	Default	313
config2	**2**	2	4	16	Default	313
config3	**3**	2	4	16	Default	313
config4	**5**	2	4	16	Default	313
config5	**6**	2	4	16	Default	317
config6	4	2	**0**	16	Default	309
config7	4	2	**1**	16	Default	310
config8	4	2	**2**	16	Default	311
config9	4	2	4	16	**2 groups of 8**	313
config10	4	2	**2**	**8**	**1 group of 8**	157
config11	4	2	**1**	**4**	**1 group of 4**	79
config12	4	**1**	4	16	Default	311
config13	4	**3**	4	16	Default	315
config14	4	**4**	4	16	Default	317

Table 1: Configurations used in design-space exploration. Entries in bold indicate deviations from the baseline.

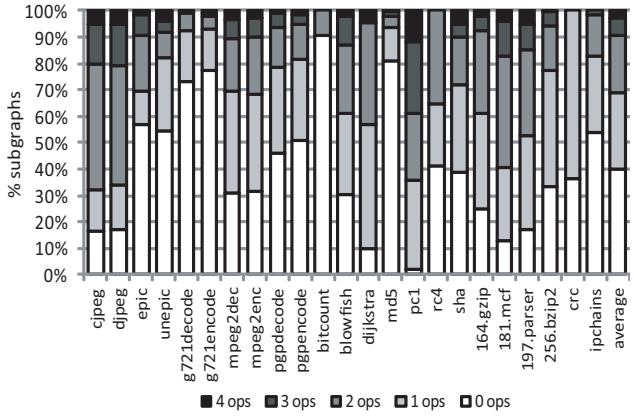

Figure 12: Distribution of subgraphs consisting of various numbers of memory operations

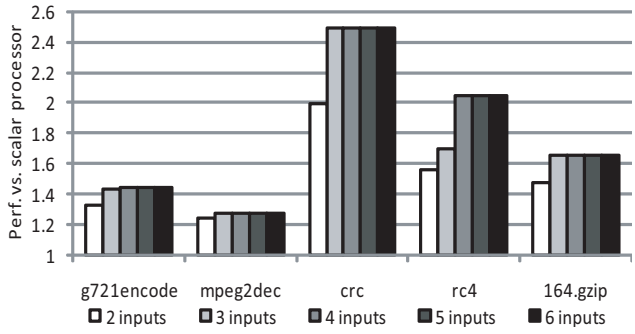

Figure 13: Performance impact of varying the number of register inputs

- The number of memory units in SIMD-Morph
- The interconnect topology

For the purposes of design-space exploration, a limited number of benchmarks were selected which were representative of the various benchmark suites in consideration: The Mediabench benchmarks `g721encode` and `mpeg2dec`, the Netbench benchmark `crc`, the encryption benchmark `rc4`, and the SPECINT2000 benchmark `164.gzip`. Each variation is compared to the baseline results shown in Section 3.

4.1 Varying Register Inputs

For these experiments, the number of register inputs was varied between 2, 3, 4 (the baseline), 5 and 6 as indicated by the configurations `config2`, `config3`, `config4` and `config5`. Every extra read port on the scalar register file contributes muxing cost and also increases the encoding space required in the configuration memory. However, increasing the number of inputs also increases the number of parallel, independent operations that can be executed on SIMD-Morph and provide improved utilization and speedup.

Figure 13 shows the results of this exploration, with the second column showing the data for the baseline 4-input configuration. While there is significant improvement for `rc4`, most of the benchmarks show a very limited increase in performance; the principle reason for this being that the memory ports allow data to be "live-in" from memory arrays, reducing the sensitivity to the amount of live-in register data. The difference observed between the 2-input and 3-input performance of all the benchmarks shown, however, justifies the added cost of adding at least one extra port to a conventional 2-read-port register file. However, because performance saturates when using a 4-read-port register file, this is the preferred configuration. In our experiments, the power consumed by the scalar register file increases approximately linearly with the number of read ports, as shown in Figure 14.

4.2 Varying Number of Memory Units

The memory units in SIMD-Morph access memory via a multi-ported load/store queue. This system helps prevent inter-lock and overlapping accesses in the event of memory aliasing when executing pointer-intensive code. Increasing the number of units therefore requires appropriate modifications to the memory system. Further, address generation support needs to be added to elements in the SIMD datapath in order to support base+displacement memory operations. For these experiments, the number of memory units in SIMD-Morph was varied between 0, 1, 2 and 4 (the baseline) units as indicated by the configurations `config6`, `config7`, and `config8`.

Figure 15 shows the results of this exploration, with the 4th column showing the data for the baseline 4-memory unit configura-

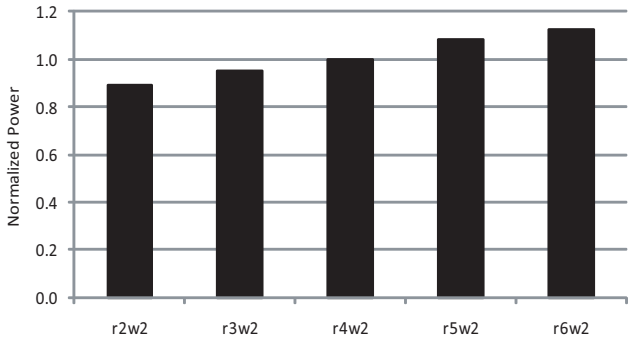

Figure 14: Register file power with varying number of register inputs, normalized to the 4-port baseline

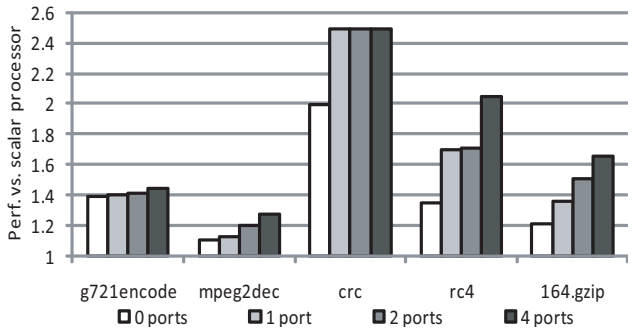

Figure 15: Impact of varying the number of memory units on speedup

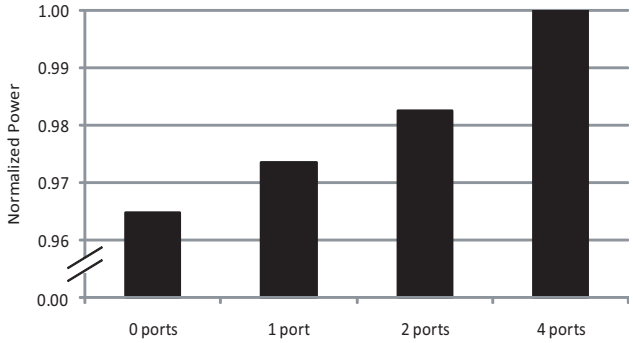

Figure 16: Normalized power impact of varying the number of memory units relative to the 4-port baseline. Note that the y-axis does not start at 0.

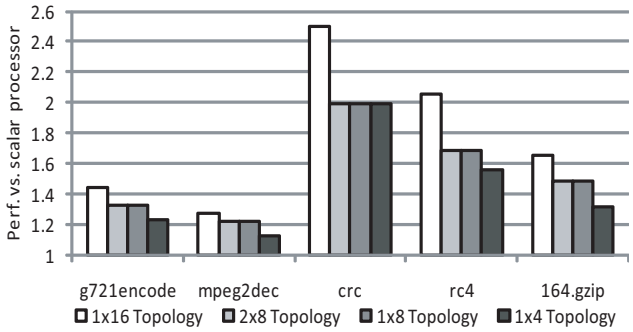

Figure 17: Impact of varying the interconnect topology on speedup

tion. There is obviously a noticeable correlation between the performance impact of changing the number of memory units and the % of subgraphs that have the corresponding number of memory operations, shown in Figure 12. For instance, `164.gzip` is most sensitive to the number of units and it also has the most number of subgraphs that have at least 1 memory operation in them. Similarly, `g721encode` is the least sensitive and fewer than 30% of its subgraphs have memory operations. The power impact of varying the number of memory ports is quite negligible as shown in Figure 16 so there is very little incentive to reduce the total number of memory operations in SIMD-Morph.

4.3 Varying Interconnect Topology

The data from Section 3 indicates that the performance impact of subgraphs with a large number of operations and with long chains of operation is, at best, minimal. In response to this, different SIMD-Morph topologies were explored, varying the interconnection network between the elements and also varying the total number of elements. These are indicated by configurations `config9`, `config10` and `config11`. In `config9`, each group of 8 elements does not communicate with the other, but one 4-element CEG within each group still feeds another. The configurations `config10` and `config11` explore the notion of not using all 16 lanes of the SIMD datapath but using 8 or 4, respectively, instead. The performance impact of these configurations is shown in Figure 17. The most important observation from this graph is that there is virtually no difference between the 2x8 and 1x8 configuration; i.e. once the maximum depth of subgraphs is halved, having extra units has no added benefit, so one may as well configure 8 elements in the SIMD-Morph style in order to save on overheads. The power savings from moving to a 1x16 configuration to a 2x8 configuration is also negligible as seen in Figure 18.

4.4 Varying Register Outputs

For these experiments, the number of register outputs was varied between 1, 2 (the baseline), 3 and 4. The modified configurations are indicated by configurations `config12`, `config13` and `config14`. Every extra write port on the scalar register file contributes muxing cost and also increases the encoding space required in the configuration memory. There is also extra overhead associated with forwarding values from the write ports to the read ports. However, much like with increasing the number of inputs, increasing the number of outputs also increases the number of parallel, independent operation chains that can be executed on SIMD-Morph, providing improved utilization and speedup, as shown in Figure 19.

Memory access support in SIMD-Morph allows for reduced use of register live-outs especially in situations where final values are written to memory arrays. For this reason, the benchmarks here are, to a point, insensitive to the number of outputs. A minimum of 1 or 2 is required to support incrementing loop counter or pointer values. A notable exception to this is, of course, `crc`, where few hot loops store values out to memory but values are live-out from one loop iteration to the next. Figure 20 shows the normalized power consumption of varying the number of write ports to the register file, indicating an approximately linear relationship between the number of ports and power consumed.

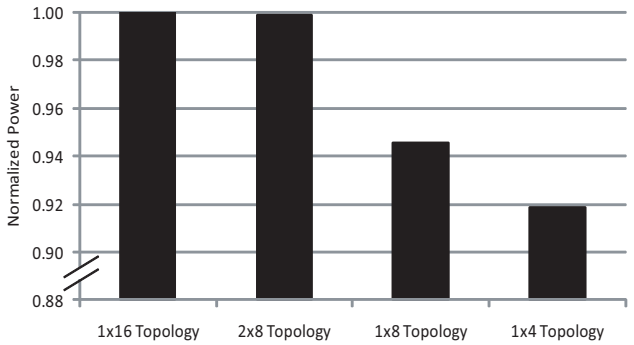

Figure 18: Normalized power impact of varying the topology relative to the 1x16 baseline. Note that the y-axis does not start at 0.

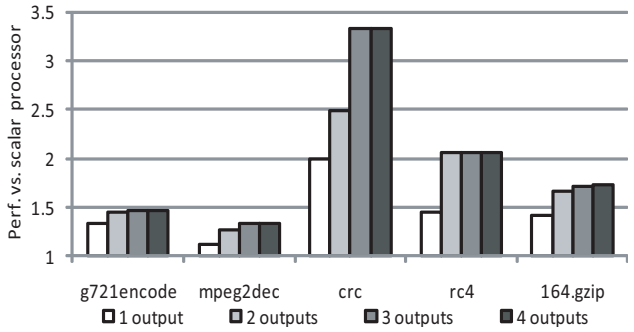

Figure 19: Impact of varying the number of register outputs on speedup

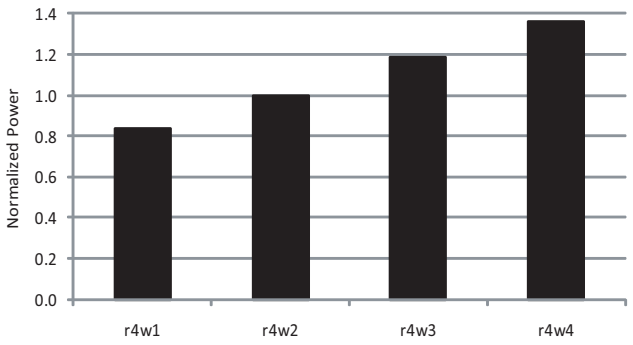

Figure 20: Register file power with varying number of register outputs, normalized to the 2-port baseline

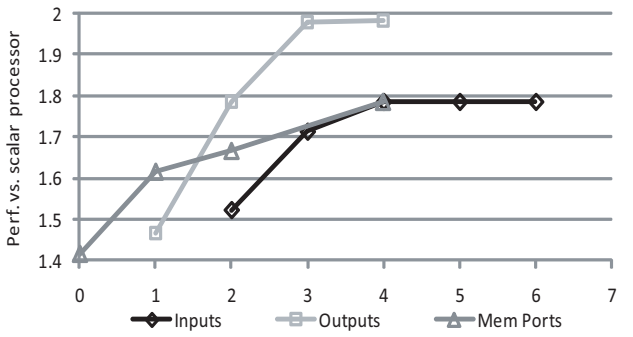

Figure 21: Average performance improvements with varying configurations

5. RESULTS

The simplest configuration that allows for the maximum performance from SIMD-Morph has 4 input values, 3 output values and 4 memory ports, using the baseline 1x16 interconnect topology. Performance saturation from using more complex configurations is illustrated in Figure 21, which shows no change in performance when using more than 4 input values and a negligible performance improvement when using more than 3 output values. This configuration is `config13` in Table 1. Figure 22 illustrates the performance:power efficiency of each of the configurations used in the design-space exploration . This figure, too, illustrates the optimal efficiency of `config13`.

The overall performance improvement when using the optimal-efficiency configuration of SIMD-Morph is shown in Figure 23. For the benchmarks from the Mediabench suite, only the non-SIMD-ized portions of the code are considered for acceleration on SIMD-Morph, while for the other benchmarks, the entire application is considered. For this reason, the averages of the two categories are displayed separately. The average speedup for the media applications is 1.4X, while the average speedup for the other applications is 2.6X.

SIMD-Morph was synthesized on a 90nm technology with a targeted clock frequency of 200 MHz using Synopsys Design Compiler and Synopsys Physical Compiler. Power consumption was measured using Synopsys Primetime-PX. The chosen baseline assumes a processor with 16-wide SIMD datapath and a 16-entry scalar register file with three read ports and two write ports, the power consumption of which is 63.48mW. The baseline SIMD-Morph configuration, illustrated in Figure 6 is connected to a scalar register file with four read ports and two write ports, the power consumption of which is 71.01mW – a power overhead of 11.9% for the modified components. The efficiency-optimal configuration, `config13`, with three write ports instead of two in the scalar register file has a power consumption of 71.53mW – a power overhead of 12.7% for the modified components. The power consumption of the scalar datapath and the SIMD register files is not accounted for here as they are not modified in this work; accounting for the power consumption of these components will reduce this power overhead. The average performance/power efficiency improvement of using the optimal SIMD-Morph configuration is 1.75X.

6. RELATED WORK

Utilizing instruction set extensions to improve the computational efficiency of applications is a well studied field. Examples of industry standard domain specific instruction set extensions such as Intel's SSE or AMD's 3DNow! multimedia instructions are commonplace in modern systems. Techniques for generating domain specific extensions are typically ad-hoc, where an architect examines a family of target applications and determines what extensions can be expected to provide increased performance.

In contrast to domain specific extensions, many techniques for generating application specific instruction set extensions have been proposed [2, 4, 9, 12, 14, 26]. Each of these algorithms provide either exact formulations or heuristics to effectively identify those portions of an application's dataflow graph that can efficiently be implemented in hardware. These techniques are not directly ap-

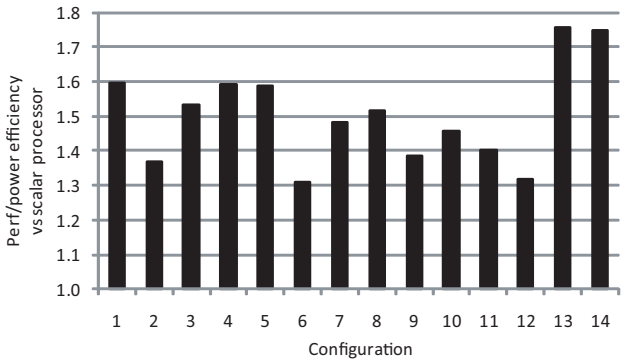

Figure 22: Average performance/power efficiency improvement of SIMD-Morph with the different datapath configurations

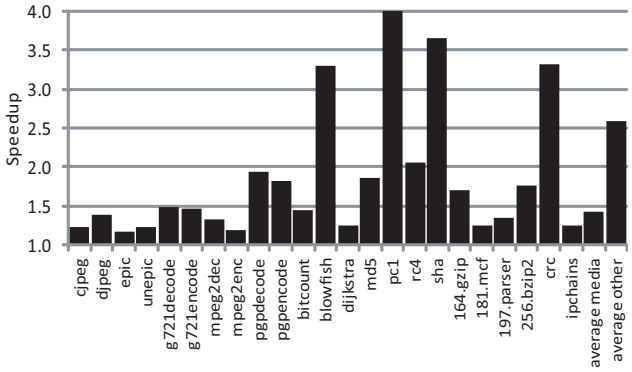

Figure 23: Speedup from using SIMD-Morph using the optimal configuration. "Average media" refers to the average speedup obtained from accelerating the outer loops of Mediabench applications while "average other" refers to the average speedup obtained from accelerating the entirety of the other applications. The speedup seen in `pc1` is 9.5X but is capped at 4X in this graph.

plicable to this work, because they do not take into account the underlying structure of the execution hardware.

Much attention has been given to the structure of a CCA (configurable compute accelerator) design for accelerating dataflow subgraphs. The research in [31] proposed using a fine granularity CCA based on slightly specialized FPGA-like elements. Restricting the interconnect of the FPGA-like elements reduces the delay of a CCA without radically affecting the number of subgraphs that can be mapped onto the accelerator. While the flexibility to map many subgraphs onto configurable hardware is appealing, there are significant drawbacks of a large number of control bits and the substantial delay of FPGA-like elements. A key observation is that the flexibility of an FPGA is generally more than is necessary for dataflow graph acceleration.

Once a CCA execution engine is developed, techniques are needed to map dataflow subgraphs onto the execute engines. Many hardware based frameworks exist for this process. Most of these arose from the observation that in systems with a trace cache, the latency of the fill unit has a negligible performance impact until it becomes very large (on the order of 10,000 cycles [10]). That is, once instructions retire from the pipeline and a trace is constructed, there is ample time before that trace will be needed again. Two recently proposed schemes [5, 24] used this latency to perform the mapping of dataflow subgraphs onto specialized execution hardware. While the trace cache provides an excellent place for mapping subgraphs into the instruction stream, it is far too large and inefficient for embedded domains.

A simplified dynamic subgraph mapping system was described in [15, 25]. These papers used the design proposed in [22] as the baseline of their system, which greatly simplifies the mapping problem. Because our goal was to allow for more flexibility than their CCA design allowed for, our presented identification algorithm is much more complex.

The key difference between this paper and prior work is that instead of proposing a CCA *architecture*, we propose an *architecture framework* into which many CCA architectures fit. This framework provides a clean interface between a processor pipeline and a CCA, enabling easy customization of a CCA for the expected system workload. We demonstrate how the framework can process a dataflow subgraph to generate CCA instruction on the fly, without the costs associated with a trace cache. Beyond the architecture framework, we describe the compilation process, by which subgraphs are identified in applications and communicated to the architecture framework.

7. CONCLUSION

Modern wireless devices are often equipped with wide SIMD processors in order to exploit data-level parallelism that is very often present in the media applications that these devices need to execute. However, not all portions of these applications is amenable to SIMD-izing. Further, several other, non-media applications are often run on these devices as well and during their executing only the scalar portion of the processor is used while the SIMD datapath is left idle. SIMD-Morph is a design which allows the SIMD datapath to be used in these situations as well. This design modifies a standard 32-bit, 16-wide SIMD datapath by adding connectivity between the different lanes in order to exploit instruction-level parallelism in sections of code that would normally be executed on the scalar pipeline of a SIMD processor. The performance benefit of SIMD-Morph is evaluated in two ways: speedup obtained from executing outer loops of applications when the inner loops are easily SIMD-ized and also the speedup obtained from executing purely sequential code on a substrate that is normally used to execute code with data-level parallelism. The performance impact for these two scenarios is a very impressive 1.4X and 2.6X, respectively.

Acknowledgements

We thank the anonymous referees who provided excellent feedback. This research was supported by National Science Foundation grant CNS-0964478, ARM Ltd, and Google.

8. REFERENCES

[1] J. H. Ahn et al. Evaluating the Imagine stream architecture. In *Proc. of the 31st Annual International Symposium on Computer Architecture*, pages 14–25, June 2004.

[2] K. Atasu, L. Pozzi, and P. Ienne. Automatic application-specific instruction-set extensions under microarchitectural constraints. In *Proc. of the 40th Design Automation Conference*, pages 256–261, June 2003.

[3] H.-M. Bluethgen, C. Grassmann, W. Raab, and U. Ramacher. A programmable platform for software-defined radio. *International Symposium on System-on-Chip*, pages 15–, Nov. 2003.

[4] P. Brisk et al. Instruction generation and regularity extraction for reconfigurable processors. In *Proc. of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 262–269, 2002.

[5] N. Clark et al. Application-specific processing on a general-purpose core via transparent instruction set customization. In *Proc. of the 37th Annual International Symposium on Microarchitecture*, pages 30–40, Dec. 2004.

[6] N. Clark et al. An architecture framework for transparent instruction set customization in embedded processors. In *Proc. of the 32nd Annual International Symposium on Computer Architecture*, pages 272–283, June 2005.

[7] N. Clark et al. Liquid SIMD: Abstracting SIMD hardware using lightweight dynamic mapping. In *Proc. of the 13th International Symposium on High-Performance Computer Architecture*, pages 216–227, 2007.

[8] N. Clark, A. Hormati, S. Mahlke, and S. Yehia. Scalable subgraph mapping for acyclic computation accelerators. In *Proc. of the 2006 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 147–157, Oct. 2006.

[9] N. Clark, H. Zhong, and S. Mahlke. Processor acceleration through automated instruction set customization. In *Proc. of the 36th Annual International Symposium on Microarchitecture*, pages 129–140, Dec. 2003.

[10] D. Friendly, S. Patel, and Y. Patt. Putting the fill unit to work: Dynamic optimizations for trace cache microprocessors. In *Proc. of the 25th Annual International Symposium on Computer Architecture*, pages 173–181, June 1998.

[11] J. Glossner, E. Hokenek, and M. Moudgill. The Sandbridge Sandblaster Communications Processor. In *3rd Workshop on Application Specific Processors*, pages 53–58, Sept. 2004.

[12] D. Goodwin and D. Petkov. Automatic generation of application specific processors. In *Proc. of the 2003 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 137–147, 2003.

[13] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *Proc. of the 4th IEEE Workshop on Workload Characterization*, pages 10–22, Dec. 2001.

[14] I. Huang. *Co-Synthesis of Instruction Sets and Microarchitectures*. PhD thesis, University of Southern California, 1994.

[15] Q. Jacobson and J. E. Smith. Instruction pre-processing in trace processors. In *Proc. of the 5th International Symposium on High-Performance Computer Architecture*, pages 125–133, 1999.

[16] C. Kozyrakis and C. Patterson. Vector vs. superscalar and VLIW architectures for embedded multimedia benchmarks. In *Proc. of the 35th Intl. Symposium on Microarchitecture*, pages 283–293, Nov. 2002.

[17] C. Lee, M. Potkonjak, and W. Mangione-Smith. MediaBench: A tool for evaluating and synthesizing multimedia and communications systems. In *Proc. of the 30th Annual International Symposium on Microarchitecture*, pages 330–335, 1997.

[18] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, and C. Chakrabarti. SODA: A low-power architecture for software radio. In *In Proc. of the 33rd Annual International Symposium on Computer Architecture*, pages 89–101, 2006.

[19] G. Memik, W. H. Mangione-Smith, and W. Hu. NetBench: A benchmarking suite for network processors. In *Proc. of the 2001 International Conference on Computer Aided Design*, pages 39–42, 2001.

[20] P. Paulin. Real-life challenges on mapping high-end video to mp-soc, 2009. 9th International Forum on Embedded MPSoC and Multicore.

[21] D. Pham et al. The design and implementation of a first generation CELL processor. In *IEEE Intl. Solid State Circuits Symposium*, Feb. 2005.

[22] J. Phillips and S. Vassiliadis. High-performance 3-1 interlock collapsing ALU's. *IEEE Transactions on Computers*, 43(3):257–268, 1994.

[23] I. T. U. M. Recommendation. *Framework and overall objectives of the future development of IMT-2000 and systems beyond IMT-2000"*. http://www.ieee802.org/secmail/pdf00204.pdf.

[24] P. Sassone and D. S. Wills. Dynamic strands: Collapsing speculative dependence chains for reducing pipeline communication. In *Proc. of the 37th Annual International Symposium on Microarchitecture*, pages 7–17, Dec. 2004.

[25] Y. Sazeides, S. Vassiliadis, and J. E. Smith. The performance potential of data dependence speculation & collapsing. In *Proc. of the 29th Annual International Symposium on Microarchitecture*, pages 238–247. IEEE Computer Society, 1996.

[26] F. Sun et al. Synthesis of custom processors based on extensible platforms. In *Proc. of the 2002 International Conference on Computer Aided Design*, pages 641–648, Nov. 2002.

[27] K. van Berkel, F. Heinle, P. P. E. Meuwissen, K. Moerman, and M. Weiss. Vector processing as an enabler for software-defined radio in handheld devices. *EURASIP J. Appl. Signal Process.*, 2005(1):2613–2625, 2005.

[28] M. Woh et al. The next generation challenge for software defined radio. In *Proc. of the* 7^{th} *International Symposium on Systems, Architectures, Modeling, and Simulation*, pages 343–354, July 2007.

[29] M. Woh, Y. Lin, S. Seo, S. Mahlke, T. Mudge, C. Chakrabarti, R. Bruce, D. Kershaw, A. Reid, M. Wilder, and K. Flautner. From soda to scotch: The evolution of a wireless baseband processor. *Proceedings. 41th Annual IEEE/ACM International Symposium on Microarchitecture, 2008. MICRO-41.*, pages 152–163, Nov. 2008.

[30] M. Woh, S. Seo, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner. AnySP: Anytime Anywhere Anyway Signal Processing. In *Proc. of the 36th Annual International Symposium on Computer Architecture*, pages 128–139, June 2009.

[31] S. Yehia and O. Temam. From sequences of dependent instructions to functions: An approach for improving performance without ILP or speculation. In *Proc. of the 31st Annual International Symposium on Computer Architecture*, pages 238–249, June 2004.