AnySP: Anytime Anywhere Anyway Signal Processing

Mark Woh[1], Sangwon Seo[1], Scott Mahlke[1], Trevor Mudge[1], Chaitali Chakrabarti[2] and Krisztian Flautner[3]

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

[2]Department of Electrical Engineering
Arizona State University, Tempe, AZ
chaitali@asu.edu

[3]ARM, Ltd.
Cambridge, United Kingdom
krisztian.flautner@arm.com

ABSTRACT

In the past decade, the proliferation of mobile devices has increased at a spectacular rate. There are now more than 3.3 billion active cell phones in the world—a device that we now all depend on in our daily lives. The current generation of devices employs a combination of general-purpose processors, digital signal processors, and hardwired accelerators to provide giga-operations-per-second performance on milliWatt power budgets. Such heterogeneous organizations are inefficient to build and maintain, as well as waste silicon area and power. Looking forward to the next generation of mobile computing, computation requirements will increase by one to three orders of magnitude due to higher data rates, increased complexity algorithms, and greater computation diversity but the power requirements will be just as stringent. Scaling of existing approaches will not suffice instead the inherent computational efficiency, programmability, and adaptability of the hardware must change. To overcome these challenges, this paper proposes an example architecture, referred to as *AnySP*, for the next generation mobile signal processing. AnySP uses a co-design approach where the next generation wireless signal processing and high-definition video algorithms are analyzed to create a domain specific programmable architecture. At the heart of AnySP is a configurable single-instruction multiple-data datapath that is capable of processing wide vectors or multiple narrow vectors simultaneously. In addition, deeper computation subgraphs can be pipelined across the single-instruction multiple-data lanes. These three operating modes provide high throughput across varying application types. Results show that AnySP is capable of sustaining 4G wireless processing and high-definition video throughput rates, and will approach the 1000 Mops/mW efficiency barrier when scaled to 45nm.

ISCA'09, June 20–24, 2009, Austin, Texas, USA.

Categories and Subject Descriptors

C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors); C.3 [**Special-Purpose and Application-Based Systems**]: [Signal processing systems]

General Terms

Algorithms, Design, Performance

1. INTRODUCTION

In the coming years, the deployment of untethered computers will continue to increase rapidly. The prime example today is the cell phone, but in the near future we expect to see the emergence of new classes of such devices. These devices will improve on the mobile phone by moving advanced functionality such as high-bandwidth internet access, human-centric interfaces with voice recognition, high-definition video processing, and interactive conferencing onto the device. At the same time, we also expect the emergence of relatively simple, disposable devices that support the mobile computing infrastructure. The requirements of these low-end devices will grow in a manner similar to those for high-end devices, only delayed in time.

Untethered devices perform signal processing as one of their primary computational activities due to their heavy usage of wireless communication and their 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 [28]. This translates to an increase in the computational requirements of 10-1000x over previous third generation wireless technologies (3G) with a power envelope that can only increase by 2-5x [33]. Other forms of signal processing, such as high-definition video, are also 10-100x more compute intensive than current mobile video.

Figure 1 presents the demands of the 3G and 4G protocols in terms of the peak processing throughput and power budget. Conventional processors cannot meet the power-throughput requirements of these protocols. 3G protocols, such as W-CDMA, require approximately 100 Mops/mW. Desktop 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. High performance systems, like the IBM Cell [24], can provide excellent through-

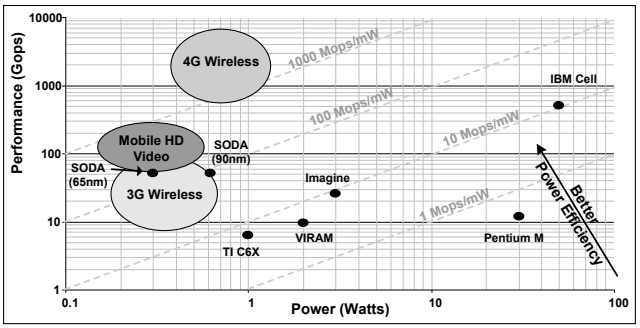

Figure 1: Performance verses power requirements for various mobile computing applications.

put, but its power consumption makes it infeasible for mobile devices [24]. Research solutions, such as VIRAM [14] and Imagine [1], can achieve the performance requirements for 3G, but exceed the power budgets of mobile terminals. SODA improved upon these solutions and was able to meet both the power and throughput requirements for 3G wireless [17]. Companies such as Phillips [32], Infineon [26], ARM [34], and Sandbridge [8] have also proposed domain-specific systems that meet the requirements for 3G wireless.

For 4G wireless protocols, the computation efficiency must be increased to greater than 1000 Mops/mW. Three central technologies are used in 4G: orthogonal frequency division multiplexing (OFDM), low density parity check (LDPC) code, and multiple-input multiple-output (MIMO) techniques. OFDM is a modulation scheme that transmits a signal over many narrow band sub-carriers. Fast Fourier transforms (FFTs) are key as they are used to place signals from the baseband to the sub-carriers. LDPC codes provide superior error correction capabilities, and less computation complexity compared to Turbo codes used in 3G. However, parallelizing the LDPC decoding algorithm is more challenging because of the large amount of data shuffling. Lastly, MIMO is based on the use of multiple antennae for both transmission and reception of signals, and requires complex signal detection algorithms. The need for higher bandwidth and increases in computation complexity are the primary reasons for the two order-of-magnitude increase in processing requirements when going from 3G to 4G.

Mobile computing platforms are not limited to performing only wireless signal processing. High-definition video is also an important application that these platforms will need to support. Figure 1 shows that the performance requirements of video exceed that of 3G wireless, but are not as high as 4G wireless. Thus the power per Gops requirement is further constrained. Moreover, the data access complexity in video is much higher than wireless. Wireless signal processing algorithms typically operate on single dimension vectors, whereas video algorithms operate on two or three dimensional blocks of data. Thus, video applications push designs to have more flexible higher bandwidth memory systems. High definition video is just one example of a growing class of applications with diverse computing and memory requirements that will have to be supported by the next generation of mobile devices.

The design of the next generation of mobile platforms must address three critical issues: *efficiency, programmability*, and *adaptivity*. The existing computational *efficiency* of 3G solutions is inadequate and must be increased by at least an order of magnitude for 4G. As a result, straightforward scaling of 3G solutions by increasing the number of cores or the amount of data-level parallelism is not enough. *Programmability* provides the opportunity for a single platform to support multiple applications and even multiple standards within each application domain. Further, programmability provides: 1) faster time to market as hardware and software development can proceed in parallel; 2) the ability to fix bugs and add features after manufacturing; and, 3) 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 widespread amounts of vector parallelism in wireless signal processing algorithms, but lose most of their efficiency when vector parallelism is unavailable or constrained.

To address these challenges, this paper proposes *AnySP*—an example of an advanced signal processor architecture that targets the next generation mobile computing. Our objective is to create a fully programmable architecture that supports 4G wireless communication and high-definition video coding at efficiency levels of 1000 Mops/mW. Programmability is recognized as a first class design constraint, thus no hardwired ASICs are employed. To address the issues of computational efficiency and adaptivity, we introduce a configurable single-instruction multiple-data (SIMD) datapath as the core execution engine of the system. The datapath supports three execution scenarios: wide vector computation (64 lanes), multiple independent narrow vector computation threads (8 threads × 8 lanes), and 2-deep subgraphs on moderate wide vector computation (32 lanes × depth 2 computations). This inherent flexibility allows the datapath to be customized to the application, but still retain high execution efficiency. AnySP also attacks the traditional inefficiencies of SIMD computation: register file power, data shuffling, and reduction operators.

The key contributions of this paper are as follows:

- An in-depth analysis of the computational characteristics of 4G wireless communication and high-definition video algorithms.
- The design, implementation, and evaluation of AnySP—an example programmable processor that targets next generation mobile computing.
- The design and organization of a configurable SIMD datapath for sustaining high throughput computing with both wide and narrow vector widths.

2. MOBILE SIGNAL PROCESSING

2.1 Overview of Benchmarks

2.1.1 4G Wireless Protocol

We model our 4G wireless system after the NTT DoCoMo test 4G wireless system [31], because there is no standard yet for 4G. The major components of the physical layer consists of three blocks: a modulator/demodulator, a MIMO encoder/decoder, and a channel encoder/decoder. These blocks constitute the majority of the 4G computations [33].

The role of the modulator is to map data sequences into symbols with certain amplitudes and phases, onto multiple

Algorithm	SIMD Workload (%)	Scalar Workload (%)	Overhead Workload (%)	SIMD Width (Elements)	Amount of TLP
FFT/IFFT	75	5	20	1024	Low
STBC	81	5	14	4	High
LDPC	49	18	33	96	Low
Deblocking Filter	72	13	15	8	Medium
Intra-Prediction	85	5	10	16	Medium
Inverse Transform	80	5	15	8	High
Motion Compensation	75	5	10	8	High

Table 1: Data level parallelism analysis for MSP algorithms. Overhead workload is the amount of instructions needed to aid the SIMD operations like data shuffle and SIMD load/store.

orthogonal frequencies. This is done using inverse FFT. The demodulator performs the operations in reverse order to reconstruct the original data sequences. The MIMO encoder multiplexes many data signals over multiple antennae. The MIMO decoder receives all the signals from the antennae and either decodes all the streams for increased data rates or combines all the signals in order to increase the signal strength. The algorithm used to increase data rate is the vertical Bell Laboratories layered space-time (V-BLAST), and the algorithm used to increase the signal quality is the space time block coding (STBC). Finally, the channel encoder and decoder perform forward error correction (FEC) that enables receivers to correct errors in the data sequence without retransmission. There are many FEC algorithms that are used in wireless systems. LDPC is used because it supports the highest data rates. Our target for 4G wireless is the maximum data rate for high mobility, which is 100Mbps. This 4G configuration utilizes the FFT, STBC, and LDPC kernels.

2.1.2 H.264 Video Standard

Video compression standards, such as MPEG-4 and H.264, are being actively considered for mobile communication systems because of the increasing demand for multimedia content on handheld devices. In this paper, H.264 is selected as the multimedia benchmark because it achieves better compression compared to previous standards and also contains most of the basic functional blocks (prediction, transform, quantization, and entropy decoding) of previous standards. Of the three profiles in H.264, we focused on the *Baseline* profile due to its potential application in videotelephony, and videoconferencing.

The H.264 decoder receives a compressed bitstream from the network abstract layer (NAL). The first block is the entropy decoder which is used to decode the bitstream. After reordering the stream, the quantized coefficients are scaled and their inverse transform is taken to generate the residual block data. Using header information in the NAL, the decoder selects prediction values for motion compensation in one of two ways: from a previously decoded frame or from the filtered current frame (intra-prediction). According to the power profile of H.264, about 75% of the decoder power consumption is attributed to three algorithms: deblocking filter (34%), motion compensation (29%), and intra-prediction (10%) [16]. Therefore, these three algorithms are selected as the H.264 kernel algorithms in this study.

2.2 Algorithm Analysis

To build a unified architecture for mobile signal processing (MSP), we performed a detailed analysis of the key kernel algorithms. These are FFT, STBC, and LDPC for the wireless workload, and deblocking, intra-prediction, inverse transform, and motion compensation for the video processing workload. Our goal was to explore the characteristics of each algorithm in order to find their similarities and differences. This section will discuss the studies that helped define our example architecture for MSP.

2.2.1 Multiple SIMD Widths

Many previous studies have analyzed and determined the best SIMD width that should be implemented for general-purpose and application-specific workloads. The optimal SIMD width is different in each study because width depends on the algorithms that constitute the workload. Examples are the 2-8-wide Intel MMX extensions [23], the 16-wide NXP EVP [32], and the 32-wide SODA [17].

Table 1 presents our study analyzing the data level parallelism (DLP) of MSP algorithms. We calculate the available DLP within each of the algorithms and show the maximum natural vector width that is achievable. The instructions are broken down into 3 categories: SIMD, overhead, and scalar. The SIMD workload consists of all the raw SIMD computations that use traditional arithmetic and logical functional units. The overhead workload consists of all the instructions that assist SIMD computations, for example loads, stores and shuffle operations. The scalar workload consists of all the instructions that are not parallelizable and must be run on a scalar unit or on the address generation unit (AGU).

From Table 1, we see that many of the algorithms have different natural vector widths–4, 8, 16, 96, 1024. Also, the algorithms with smaller SIMD widths exhibit a high level of TLP, which means that we can process multiple threads that work on separate data on a wide SIMD machine. For instance, 8 instances of STBC that have SIMD width of 4 can be processed on a 32-wide machine. Unlike most SIMD architectures that are designed with a fixed SIMD width to process all the algorithms, this study suggests that the best solution would be to support multiple SIMD widths and to exploit the available thread-level parallelism (TLP) when the SIMD width is small. By supporting multiple SIMD widths, the SIMD lane utilization can be maximized as will be demonstrated in Section 3.1.1.

Though the scalar and overhead workloads are not the majority, they still contribute 20-30% of the total computation. In instances such as LDPC, comprised of mostly simple computations, data shuffling and memory operations dominate the majority of the workload. This suggests that we cannot simply improve the SIMD performance, but also must reduce the overhead workload. This can be accomplished

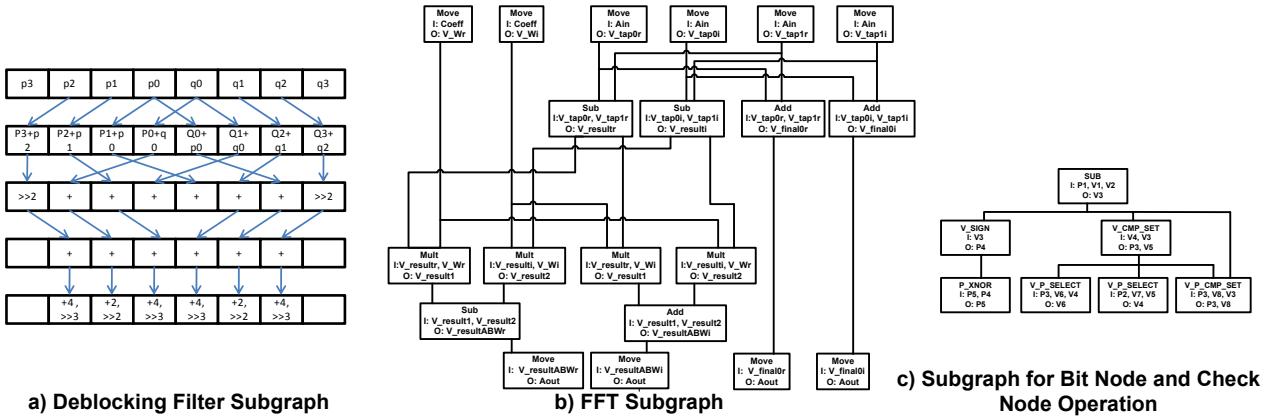

Figure 2: Select subgraphs for the inner loops of MSP algorithms.

by introducing better support for data reorganization or by increasing the scalar and AGU performance. Table 1 also shows how much TLP is available; high TLP means that many independent but identical threads can be spawned and low means only one or a few can be spawned mainly due to data dependencies or absence of outer loops.

2.2.2 *Register Value Lifetimes*

For many processors, the register file (RF) contributes a large percentage of the power consumption. Some architectures have RFs that consume 10-30% of the total processing element (PE) power [11] [22]. This is due to the reading and writing of values for each instruction. Many architectures, like very large instruction word (VLIW), compound this problem by having multi-ported RFs that increase the energy per access. Recent works in [9] and [7] show that performance and power can be reduced by bypassing the RF for short-lived values and storing the value within a register bypass pipeline. The bypass network is a non-critical path structure for holding values a fixed number of cycles before they are consumed. By bypassing the RF, access power and register pressure are reduced allowing for more effective register allocation. Another technique used to reduce register file power is register file partitioning. Previous research found that for some embedded applications, a small set of registers accounted for the majority of the register file accesses [11]. By splitting the register file into a small and large region, the high access registers can be stored in the small partition resulting in less energy consumed per access.

One focus of this paper is to study the potential for eliminating RF accesses in MSP applications. Figure 2 shows the subgraphs for several of the MSP inner loops. We see that there exists large amounts of data locality in these subgraphs, which is illustrated by the number of single lines that go from one source to one destination. The MSP applications fit the streaming dataflow model, where data is produced and then read once or twice. The values are usually consumed by the instructions directly succeeding the producer. The number of temporary registers needed are small but are accessed often suggesting a register bypass network or a partitioned RF can reduce power and increase performance.

We further analyze the MSP algorithms to determine the extent to which RF accesses can be reduced. We use the SODA architecture [17] with a small four entry RF partition

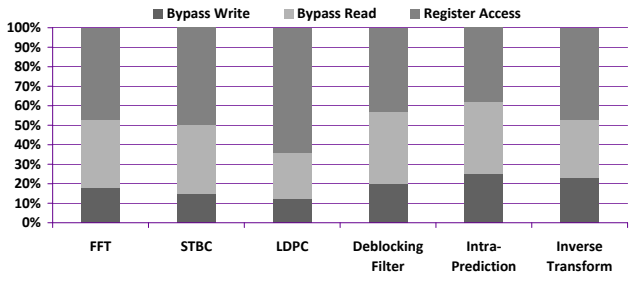

Figure 3: Register file access for the inner loops of MSP algorithms. Bypass write and read are the RF read and write accesses that do *not* need to use the main RF. Register accesses are the read and write accesses that *do* need to use the main RF.

modification creating a total of 20 registers: 16 main registers and 4 temporary registers. Figure 3 shows the breakdown of accesses that can and cannot bypass the RF. *Bypass write* and *bypass read* correspond to the RF read and write accesses that do not need to use the main RF and can be eliminated with either data forwarding or by storing the data in a small RF partition. *Register accesses* are the read and write accesses that cannot be bypassed and need to use the main RF. For most of the algorithms a large portion of register accesses can be bypassed. The exception to this is LDPC where many values are calculated and stored for a several cycles before one of them is chosen. Based on our analysis, we conclude that architectures that run MSP algorithms should support bypass mechanisms to help reduce the accesses to the main RF.

2.2.3 *Instruction Pair Frequency*

In typical digital signal processing (DSP) codes, there are pairs of instructions that are frequently executed back to back. A prime example of this is the multiply followed by accumulate instruction. To improve performance and decrease energy consumption, DSP processors such as the Analog Devices ADSP and the TI C series introduced the multiply-accumulate (MAC) operation. The work in [4] has exploited this further by building specialized hardware to exploit long chains of instructions.

We performed a study on the MSP algorithms to find the most common producer-consumer instruction pairs. In each

	Instruction Pair	Frequency	Total
1	multiply-add	26.71%	26.71%
2	add-add	13.74%	40.45%
3	shuffle-add	8.54%	48.99%
4	shift right-add	6.90%	55.89%
5	subtract-add	6.94%	62.83%
6	add-shift right	5.76%	68.59%
7	multiply-subtract	4.14%	72.73%
8	shift right-subtract	3.75%	76.48%
9	add-subtract	3.07%	79.55%
10	Others	20.45%	100.00%

a) Intra-prediction and Deblocking Filter Combined

	Instruction Pair	Frequency	Total
1	shuffle-move	32.07%	32.07%
2	abs-subtract	8.54%	40.61%
3	move-subtract	8.54%	49.15%
4	shuffle-subtract	3.54%	52.69%
5	add-shuffle	3.54%	56.23%
6	Others	43.77%	100.00%

b) LDPC

	Instruction Pair	Frequency	Total
1	shuffle-shuffle	16.67%	16.67%
2	add-multiply	16.67%	33.33%
3	multiply-subtract	16.67%	50.00%
4	multiply-add	16.67%	66.67%
5	subtract-mult	16.67%	83.33%
6	shuffle-add	16.67%	100.00%

c) FFT

Figure 4: Instruction pair frequencies for different MSP algorithms.

case the producer instruction's value, once consumed, is not referenced again. This allows the producer and consumer instructions to be fused. Figure 4 shows the breakdown of the most frequent instruction pairs that were found in MSP algorithms programmed for SODA [17]. Among all the algorithms, the MAC instruction was the most frequent, because a large number of these algorithms have a dot product as their inner loop. Other fused instruction pairs that were common were permute-add, add-add, and shift-add. The permute-add pair occurs in many of the algorithms because data must first be aligned to effectively utilize the SIMD lanes. The add-add and shift-add pairs also occur quite frequently in the video algorithms, because values are accumulated together and then normalized by a power of 2 division—a shift.

Based on this analysis, we conclude that the fused instruction pairs with high frequency should be supported. This increases performance and decreases power because register access between the producer and consumer instructions would no longer be needed.

2.2.4 Algorithm Data Reordering Patterns

Most commercial DSP and general purpose processor(GPP) multimedia extensions support some form of data reordering operations. By data reordering we refer to both permutation and replication of data. In graphics terminology this is referred to as swizzling. Swizzle operations are particularly important in SIMD machines where they are essential to keep the functional units utilized. For example, Intel's Larrabee [29], which supports 16-wide SIMD operations, has a hardware "gather and scatter" instruction that allows 16 distinct memory addresses to build one SIMD vector. This is one extreme case where the gather and scatter operations can take multiple cycles and consume significant power if a number of distinct cache lines have to be accessed. At the other end of the spectrum is the data "shuffle" network in SODA [17] where the programmer needs to align the data within 32-wide SIMD blocks often requiring multiple shuffle instructions. This consumes less power than Intel's Larrabee but can take multiple cycles to complete.

We performed a study to enumerate the number of swizzle operations required for the MSP algorithms. Figure 5 shows a few of the common examples. The key finding was that all of the algorithms had a predefined set of swizzle operations, typically below 10 that were known beforehand. Because all the swizzle operations were known beforehand, sophisticated gather and scatter hardware support like that of Larrabee is not needed. However, a shuffle network like that of SODA, is not complex enough to support the needed swizzle operations. An example is LDPC where the number of predetermined swizzle operations(in this case permutations) were large. In LDPC we need a mechanism to change

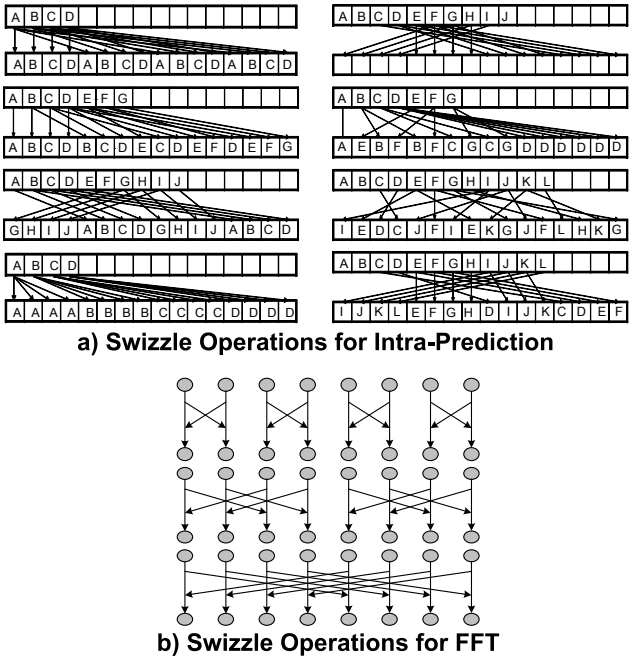

Figure 5: A set of swizzle operations that are required for a subset of MSP algorithms.

the swizzle operations based on a parity check matrix. In SODA, the data shuffle network may require many cycles to complete these task becoming a bottleneck in the performance of LDPC. A network which can be easily configured and also perform the operations in fewer cycles is desirable. Either a swizzle network or a more complex programmable fixed pattern network would be the best solution.

2.2.5 Design Study Summary

The result of these studies provided us with four key insights that will be exploited in the next section to design an efficient high-performance architecture for MSP applications:

- Most algorithms have small natural vector widths and large amounts of TLP
- A large percentage of register values are short-lived and many do not need to be written to the register file
- A small set of instruction pairs are used a large percentage of the time
- Each algorithm uses a small set of predetermined swizzle patterns

3. ANYSP ARCHITECTURE

In this section, we describe the AnySP architecture which exploits the MSP algorithm features described in the previous section.

3.1 AnySP PE Design

The PE architecture is shown in Figure 6. It consists of SIMD and scalar datapaths. The SIMD datapath in turn consists of 8-groups of 8-wide SIMD units, which can be configured to create SIMD widths of 16, 32 and 64. Each of

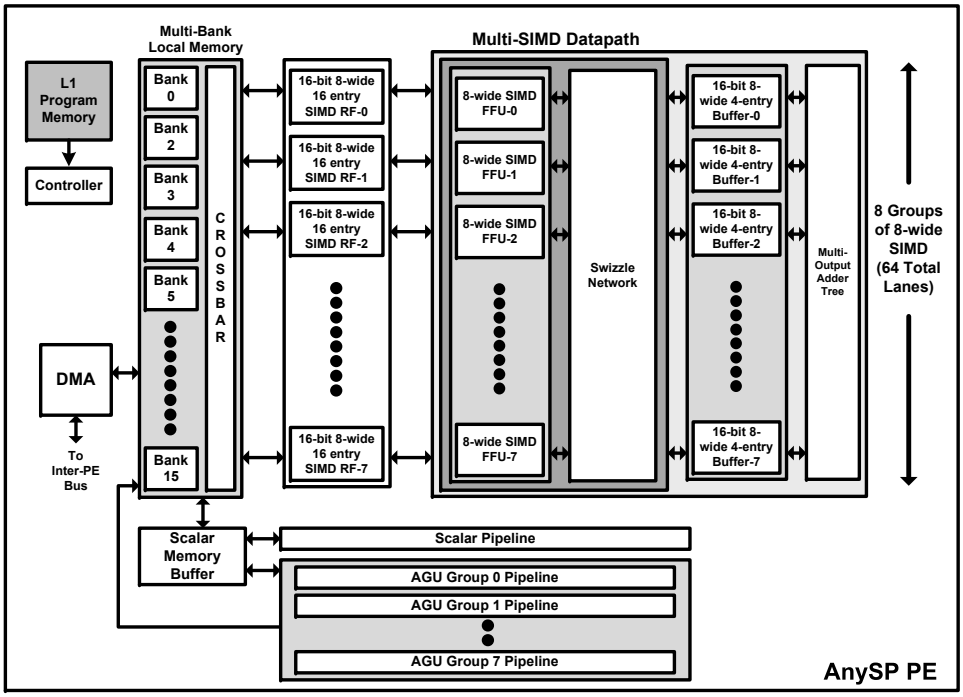

Figure 6: AnySP PE. It consists of SIMD and scalar datapaths. The SIMD datapath consists of 8-groups of 8-wide SIMD units, which can be configured to create SIMD widths of 16, 32 and 64. Each of the 8-wide SIMD units are composed of groups of Flexible Functional Units (FFUs). The local memory consists of 16 memory banks; each bank is an 8-wide SIMD containing 256 16-bit entries, totalling 32KB of storage.

the 8-wide SIMD units are composed of groups of Flexible Functional Units (FFUs). The FFUs contain the functional units of two lanes which are connected together through a simple crossbar. The SIMD datapath is fed by eight SIMD register files (RFs). Each RF is 8 wide and has 16 entries. The swizzle network aligns data for the FFUs. It can support a fixed number of swizzle patterns of 8, 16, 32, 64 and 128 wide elements. Finally, there is a multiple output adder tree that can sum of groups of 4, 8, or 16 elements, and store the results into a temporary buffer.

The local memory consists of 16 memory banks; each bank is an 8-wide SIMD containing 256 16-bit entries, totalling 32KB of storage. Each 8-wide SIMD group has a dedicated AGU unit. When not in use, the AGU unit can run sequential code to assist the dedicated scalar pipeline. The AGU and scalar unit share the same memory space as the SIMD datapath. To accomplish this, a scalar memory buffer that can store 8-wide SIMD locations is used. Because many of the algorithms access data sequentially, the buffer acts as a small cache which helps to avoid multiple accesses to the vector banks. Details about each of these architectural features is discussed in the rest of this section.

3.1.1 Configurable Multi-SIMD Width Support

We analyzed the performance of MSP algorithms on existing SIMD architectures like [17][32] and found that the full SIMD widths could not always be utilized. In many of the algorithms the natural vector width is smaller than the processors's SIMD width, leading to wasted lanes, and power or performance penalties. In order to improve SIMD utilization, code transform work, such as [18], tried to transform the algorithms to fit the SIMD width. This was achieved at the cost of increased total computation and complex code transformations. These techniques should be avoided because of their complexity and increase in power consumption from additional preprocessing requirements.

In Section 2.2.1, we showed that many of the algorithms had different natural vector widths. While the small SIMD width kernels had large amounts of TLP, the ones with higher DLP showed little TLP. An interesting insight was that independent threads were not the only contributors to TLP. Specifically, in H.264, the same task would be run many times for different macroblocks. Each task was independent of each other and ran the exact same code and followed almost the same control path. The difference between each thread was that the data accesses from memory were different. This meant that for each thread, separate memory addresses would have to be computed. In SODA, the only way we were able to handle these threads was to execute them one at a time on the 32-wide SIMD. This is inefficient because these algorithms had widths smaller than the SIMD width. In order to support these types of kernel algorithms, we chose to design a multi-SIMD width architecture. Each group of 8-wide SIMD units has its own AGU to access different data. The 8-wide groups can also be combined to create SIMD widths of 16, 32 or 64. Such a feature allows us to exploit the DLP and TLP together for large and small SIMD width algorithms. Small SIMD width algorithms like intra-prediction and motion compensation, can process multiple macroblocks at the same time while exploiting the 8-wide and 16-wide SIMD parallelism within the algorithms. Large SIMD width algorithms like FFT and LDPC, can use the full 64-wide SIMD width and run multiple iterations.

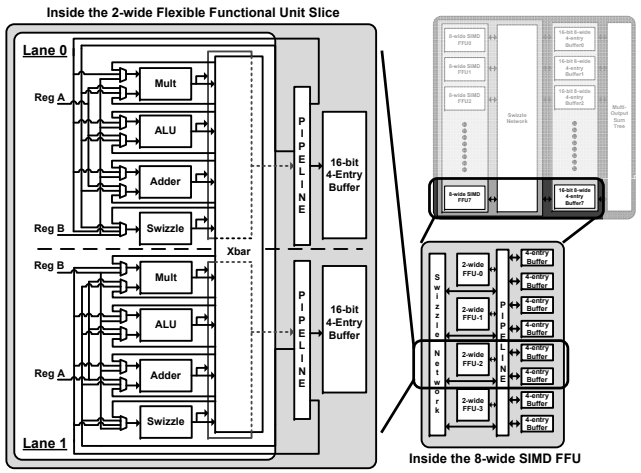

Figure 7: Flexible Functional Unit

3.1.2 Temporary Buffer and Bypass Network

As discussed in Section 2.2.2, the RF is one of the major sources of power consumption in embedded processors. We implemented two different techniques in order to lower RF power: temporary register buffers and a bypass network.

The temporary register buffers are implemented as a partitioned RF. The main RF still contains 16 registers. A second partition containing 4 registers is added to the design, making the total number of registers 20. This small partitioned RF shields the main RF from accesses by storing values which have very short lifetimes. This saves power because the smaller, lower power RF is accessed multiple times rather than the main, higher power, RF. Another benefit is that by reducing accesses to the main RF, register file pressure is lessened, thereby decreasing memory accesses.

The bypass network is a modification to the writeback stage and forwarding logic. Typically in processors, data that is forwarded to another instruction to eliminate data hazards is also written back to the RF. In the bypass network, the forwarding and writing to the RF is explicitly managed. This is similar to TTA [5], which defines all input and output ports of all modules as registers and the programmer explicitly manages where the data should go. However, in our design we only allow the bypass network to be visible to the programmer. This means that the instruction dictates whether the data should be forwarded and whether the data should be written back to the RF. This eliminates RF writes for values that are immediately consumed thus reducing RF power.

3.1.3 Flexible Functional Units

In typical SIMD architectures, power and performance is lost when the vector size is smaller than the SIMD width due as a result of underutilized hardware. The proposed flexible functional units (FFU) allow for reconfiguration and flexibility in the operation of different types of workloads. When SIMD utilization is low, the FFUs are able to combine the functional units between two lanes, which can speed up more sequential workloads that underutilize the SIMD width by exploiting pipeline parallelism. This effectively turns two lanes of the SIMD into a 2-deep execution pipeline. Two different instructions can be chained through the pipeline, and data can be passed between them without writing back to the RF. Ultimately, this allows two instructions per chained FFU to be in flight per cycle. Using instruction chaining we are able to extract different levels of parallelism like pipeline parallelism and DLP within the same FFU unit.

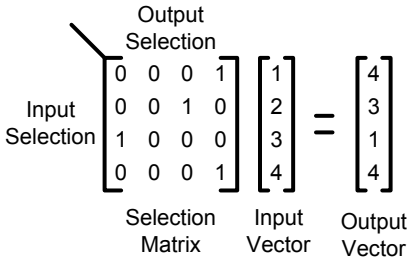

Figure 8: The swizzle operation in computer graphics. The rows of the selection matrix correspond to the input vector and the columns correspond to the output vector selected. Each row can only have a single 1, but columns can have multiple 1s which allow for selective multi-casting of the data.

As shown in Figure 7, each 8-wide SIMD group is built from four 2-wide FFUs. Each functional unit consists of a multiplier, ALU, adder and swizzle network sub-block. Such a structure benefits algorithms with SIMD widths that are smaller than 64. In chained execution mode, the functional units among two internal lanes can be connected together through a crossbar network. Overall, FFUs improve performance and reduces power by adding more flexibility and exploiting both TLP and DLP.

3.1.4 Swizzle Network

Our analysis in Section 2.2.4 showed that the number of distinct swizzle patterns that are needed for a specific algorithm is small, fixed, and known ahead of time. Previous research has discussed building application specific crossbars for SIMD processors [25]. Such designs hardwired only the swizzle operations needed to support the application, yielding a power efficient design. However, they lack flexibility as they are unable to support new swizzle operations for different applications post fabrication.

We propose using an SRAM-based swizzle network that adds flexibility while maintaining the performance of a customized crossbar. Figure 8 illustrates how the swizzle operation works. If the *(i,j)*th entry of the selection matrix is 1, then input i is routed to output j.

The SRAM-based swizzle network is similar to the layout of [10], where the layout is an X-Y style crossbar, where the input buses are layed out horizontally and the outputs are layed out vertically. Each point of intersection between the input and output buses contains a pass transistor which is controlled by a flip flop. This allows for more flexibility in that configurations can be changed post fabrication. However, this design has three drawbacks. First, each output is directly driven by the input. As the crossbar size increases, the capacitance of the output lines increase requiring input drivers to be larger, resulting in a higher power consumption. Second, each time a different permutation is needed, the flip flops have to be reprogrammed resulting in additional cycles. Third, a large number of control wires is required for reprogramming the flip flops increasing power consumption and complicating the layout.

To solve these short comings, we leverage an SRAM-based

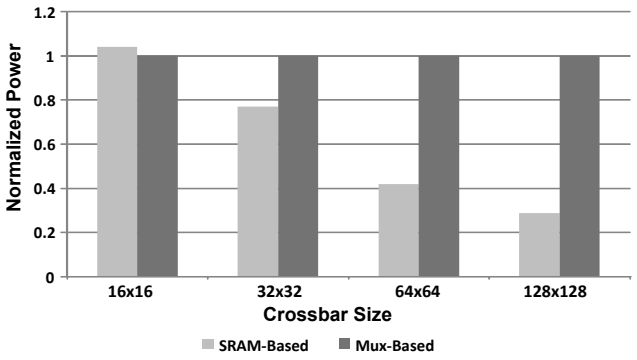

Figure 9: Comparison of power between the SRAM-based swizzle network and MUX-based crossbar. The values are normalized to the MUX-based crossbar for different crossbar widths with 16-bit data inputs. These results correspond to a switching activity of 0.50.

swizzle network in which multiple SRAM cells replace the single flip flop of [10]. First, by using circuit techniques, we decouple the input and output buses reducing the needed input driver strength and decreasing the overall power consumption while providing more scalability. Second, multiple sets of swizzle configurations can be stored into the SRAM cells, allowing zero cycle delays in changing the swizzle pattern. Third, by storing multiple configurations we remove a large number of control wires necessary.

Using the SRAM-based swizzle network, the area and power consumption of the network can be reduced while still operating within a single clock cycle. Figure 9 shows the power difference between the SRAM-based swizzle network and MUX-based crossbar for different crossbar sizes using 16-bit inputs. As we can see, for crossbar sizes larger than 32x32, the power of the SRAM-based swizzle network is dramatically lower than the MUX-based one and is able to run at almost twice the frequency.

This is one of the enabling technologies for very wide SIMD machines to be power efficient and fast. Though only a certain number of swizzle patterns can be loaded without reconfiguration, it is a viable solution since only a limited set of swizzle patterns need to be supported for each algorithm. These swizzle patterns can be stored into the SRAM-based swizzle network configuration memory at initialization. Because it supports arbitrary swizzle patterns with multi-casting capabilities, it functions better than previous permutation networks found in [17][34].

3.1.5 Multiple Output Adder Tree Support

SIMD architectures such as [17] have special SIMD summation hardware to perform "reduction to scalar" operations. To compute this, adder trees sum up the values of all the lanes and store the result into the scalar RF. These techniques worked for 3G algorithms but do not support video applications where sums of less than the SIMD width are needed. Other architectures capable of processing video applications like [2][27] have sum trees, but their SIMD widths were only 4-8 elements wide. For wide-SIMD machines like AnySP, we designed the adder tree to allow for partial summations of different SIMD widths, which then writeback to the temporary buffer unit. Writing the resultant sum into the temporary buffer unit is key, because many times the summed values have short lifetimes but are updated and used very frequently within that period.

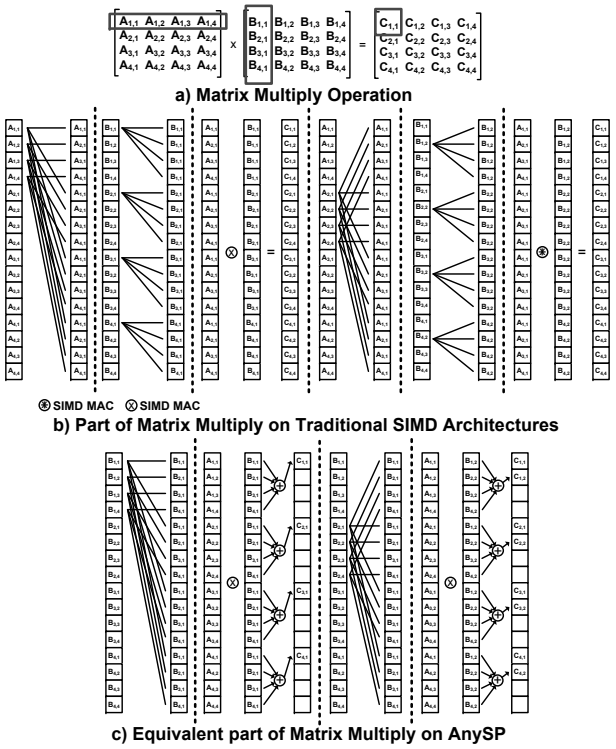

Figure 10: Demonstration of the output adder tree for matrix multiplication

In the 4G and video decoding algorithms, matrix calculations occur frequently. One of these calculations is matrix multiplication, shown in Figure 10a, and an example of doing a 4x4 matrix multiply on a typical SIMD architecture is shown in Figure 10b. This calculation requires many swizzle operations to align both matrices. Figure 10c shows how using an adder tree that adds only 4 numbers can reduce the total operations required. Here, we see that only one of the matrix values needs to be swizzled. Since the adder tree only produces 4 of the 16 values per iteration, the adder tree uses the temporary register buffer to store the partial SIMD values in order to prevent wasted reading and writing to the RF. After 4 iterations, 16 sums are produced forming a complete SIMD vector that is then written back to the RF. AnySP supports sums of 4, 8, 16, 32 or 64 elements.

4. RESULTS AND ANALYSIS

4.1 Methodology

The RTL Verilog model of the SODA processor [17] was synthesized in TSMC 180 nm technology, and the power and area results for 90 nm technology were estimated using a quadratic scaling factor based on Predictive Technology Model [12]. The main hardware components of the AnySP processor were implemented as RTL Verilog and synthesized in TSMC 90 nm using Synopsys physical compiler. The timing and power numbers were extracted from the synthesis and used by our in-house architecture emulator tool to calculate the timing and power values for each of the kernels.

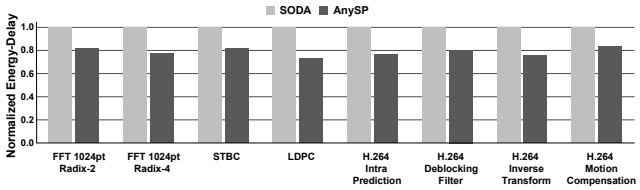

Figure 12: Normalized Energy-Delay product for each kernel algorithm

AnySP's PE area is 130% larger than SODA's estimated 90 nm PE area. Unlike SODA, where the target frequency was 400 MHz, AnySP was targeted at 300 MHz. The major kernels of the 100 Mbps high mobility 4G wireless protocol and high quality H.264 4CIF video are analyzed. The 4CIF video format is 704x576 pixels at 30 fps.

4.2 Algorithm-Level Results

In this section, we present the performance analysis of key algorithms in MSP, namely FFT, STBC and LDPC of 4G wireless protocols and intra-prediction, deblocking filter, inverse transform and motion compensation of an H.264 decoder.

The speedup of AnySP over SODA for each algorithm is shown in Figure 11. Each improvement is broken down by architectural enhancements: wider SIMD width (from 32 to 64), use of single-cycle SRAM-based swizzle network, fused operations, and temporary buffer with the bypass network. The energy-delay product for running the kernel algorithms on SODA and AnySP are also presented in Figure 12. On average, AnySP achieves 20% energy-delay improvement over SODA with more than 2x speedup. A detailed analysis is presented next.

4.2.1 4G Algorithm Analysis

Fast Fourier Transform. Figure 11 provides results of two different FFT configurations: 1024 point Radix-2, and 1024 point Radix-4. On average, AnySP achieves a 2.21x speedup over SODA. A wider SIMD width is the main contribution of the speedup because FFT is a highly parallel algorithm. The radix-4 FFT algorithm has better performance compared to radix-2. 34% of the speedup is attributed to the fused-operations, such as shuffle/add. The radix-4 FFT algorithm takes more advantage of the single-cycle SRAM-based swizzle network compared to the radix-2 algorithms, because radix-4 algorithms require complex shuffle operations that cannot be done in a single cycle by other networks.

STBC. As shown in Table 1, STBC contains large amounts of DLP and TLP. Figure 11 shows that the wider SIMD width along with the multi-SIMD support helped speed up STBC by almost 100%. The rest of the speedup was obtained with the FFU, where the fused-operations contributed another 20% speedup. This was because fused pairs such as add-add, subtract-add, add-subtract, and subtract-subtract were used frequently in the algorithm. A combination of fused-operations, register buffers, and data forwarding bypass network added to an almost 20% decrease in the energy-delay product, making the algorithm run more efficiently compared to the SODA architecture.

LDPC. Among the many decoding algorithms for LDPC, the min-sum algorithm was selected because of its simple operations and low memory requirements. As shown in Figure 11, AnySP's LDPC implementation has a speedup of 2.43x compared to SODA. This performance jump is because of the wider SIMD width and the versatile swizzle network which significantly reduced the number of register reads/writes and cyclic shuffle operations when dealing with large block codes. In addition, the support of temporary buffers gives the LDPC min-sum decoding function storage for the minimum and the second minimum values thereby accelerating the corresponding compare/update process [30].

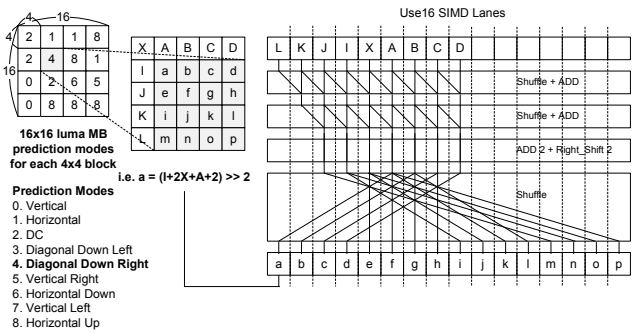

Figure 13: Mapping a luma 16x16 MB intra-prediction process on AnySP; Example of the Diagonal Down Right intra prediction for a 4x4 sub block (grey block) is presented with each cycle's operations listed.

4.2.2 H.264 Algorithm Analysis

Intra-Prediction. Figure 13 shows how the H.264 intra-prediction process is mapped onto AnySP. A 16x16 luma macroblock is broken into 16 4x4 sub-blocks, and each 4x4 sub block has a different intra-prediction mode: Vertical, Horizontal, DC, Diagonal Down Left, Diagonal Down Right, Vertical Right, Horizontal Down, Vertical Left, or Horizontal Up. As can be seen in Figure 13, 16 SIMD lanes are used to generate 16 prediction values (a,b, ..., p) with neighboring pixel values (capital letters). Independence between intra-prediction computations for 4x4 sub-blocks allows other SIMD units to execute on different 4x4 sub blocks simultaneously. A factor of 2 increase in the SIMD width almost doubles the processing performance. In addition, fused-operations (shuffle-add and add-shift) reduces unnecessary register read/write accesses, and the SRAM-based swizzle networks supports single cycle complex swizzle operations, both results in a speedup of 30%.

Deblocking Filter. The H.264 deblocking filter smoothens block edges of decoded macroblocks to reduce blocking distortion. Based on data-dependent filtering conditions, the function of this filter varies dynamically (three-tap, four-tap, or five-tap filter). Figure 11 shows AnySP achieves about 2.18x performance increase compared to SODA. The wider SIMD width allows the deblocking filter to take twice as many pixel values to process, which results in 2x speedup over SODA. Like H.264 intra-prediction, fused operations such as shuffle-add, shuffle-shift, and add-shift and the swizzle network helps boost performance up to 27% and 16%, respectively.

Motion Compensation. H.264 adopts tree structured motion compensation (MC). The size of a MC block can be one of 16x16, 16x8, 8x16, 8x8, 4x8, or 4x4, which uses integer-pixel, half-pixel, quarter-pixel, or eighth-pixel resolution. Because sub-sample positions do not exist in the

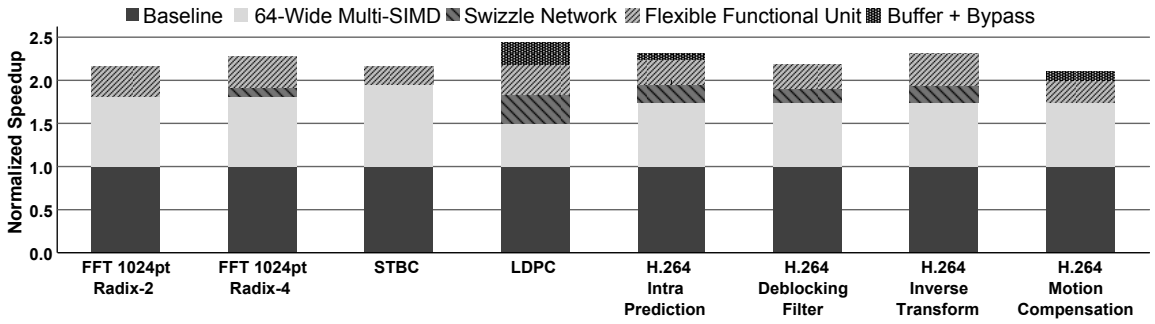

Figure 11: AnySP speedup over SODA for the key algorithms used in 4G and H.264 benchmarks. The speedup is broken down into the different architectural enhancements - wider SIMD width, single-cycle SRAM-based crossbar, fused-operations and buffer support

			Area		4G + H.264 Decoder	
	Components	Units	Area mm^2	Area %	Power mW	Power %
PE	SIMD Data Mem (32KB)	4	9.76	38.78%	102.88	7.24%
	SIMD Register File (16x1024bit)	4	3.17	12.59%	299.00	21.05%
	SIMD ALUs, Multipliers, and SSN	4	4.50	17.88%	448.51	31.58%
	SIMD Pipeline+Clock+Routing	4	1.18	4.69%	233.60	16.45%
	SIMD Buffer (128B)	4	0.82	3.25%	84.09	5.92%
	SIMD Adder Tree	4	0.18	<1%	10.43	<1%
	Intra-processor Interconnect	4	0.94	3.73%	93.44	6.58%
	Scalar/AGU Pipeline & Misc.	4	1.22	4.85%	134.32	9.46%
System	ARM (Cortex-M3)	1	0.6	2.38%	2.5	<1%
	Global Scratchpad Memory (128KB)	1	1.8	7.15%	10	<1%
	Inter-processor Bus with DMA	1	1.0	3.97%	1.5	<1%
Total	90nm (1V @300MHz)		**25.17**	100%	**1347.03**	100%
Est.	65nm (0.9V @ 300MHz)		**13.14**		**1091.09**	
	45nm (0.8V @ 300MHz)		**6.86**		**862.09**	

Figure 14: PE Area and Power Summary for AnySP running 100Mbps high mobility 4G wireless and H.264 4CIF video at 30fps

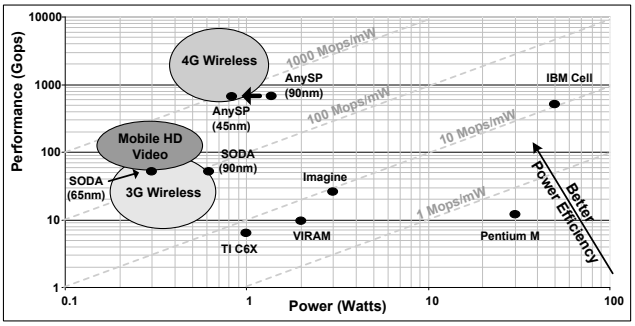

Figure 15: Performance verses power requirements for various mobile computing applications.

reference frames, the fractional pixel data is created by interpolation. For example, half-pixel values are generated by the 6-tap filtering operation, which fits into an 8-wide SIMD partition; therefore a wider SIMD engine supports processing of multiple filters at the same time. Also, buffer support helps store half-pixel values while sliding a filter window, which saves the number of SIMD-scalar data exchanges. Overall, a 2.1x speedup is achieved by these enhancements.

4.3 System-Level Results

Figure 14 shows the power and area breakdown of AnySP running both 100 Mbps high mobility 4G wireless and high quality H.264 4CIF video. AnySP was able to meet the throughput requirement of 100 Mbps 4G wireless while consuming 1.3 W at 90 nm. This is just below the 1000 Mops/mW target but close enough to meet that target in 45 nm process technology. We show this in Figure 15 where we replot AnySP on the performance verses power chart of Figure 1. It can also be seen that high quality H.264 4CIF video at 30 fps can be achieved with 60 mW at 90 nm, meeting the requirements for mobile HD video.

The power breakdown of AnySP shows that the SIMD functional units are the dominant power consumers. The RF and data memory are lower in comparison which shows that our design is efficient because the largest portion of the power is spent doing actual computations. In comparison, SODA's RF and data memory power were the dominant consumers which suggested that a lot of power was being wasted reading and writing data from the register file and memory. Our proactive approach to reducing register and memory accesses has helped in improving the efficiency of our design.

5. RELATED WORK

Many architectures have tried to exploit DLP and TLP in order to increase power efficiency. For instance, the Vector-Thread (VT) architecture [15] can execute in multiple modes:

SIMD, MIMD, and hybrids of the two by controlling which processors fetch instructions. AnySP always executes in SIMD mode. When vector lengths are less than the SIMD width, neighboring lanes are combined to execute complex subgraphs or simultaneously operate on multiple vectors using the FFU. Maintaining the single-instruction mode of operation translates into gains in power efficiency compared VT. Although VT is more flexible, the workload characteristics of 4G and H.264 show that this level of flexibility is not required. Judiciously restricting the flexibility helps increase the power efficiency of AnySP.

The use of temporary buffer networks is not a new concept, many former architectures have used it [6][11][21]. The novelty here is the usage model of the temporary buffer and bypass network. Like ELM [6], we use it to store the temporary values that usually do not need to be written back to the register file. The temporary buffer units are connected to the swizzle network in the 8-wide SIMD FFU. This allows the temporary values to be exchanged within the lanes of the SIMD group, which ELM cannot do. In order to communicate data across multiple ALU units in ELM, the data has to go through an ALU and then enter the distributed switch network. By allowing connection between ALU units, the performance and power is optimized in our SIMD design.

Current solutions that support 3G wireless protocols in SDR solutions can be broken into two categories: SIMD-based architectures and reconfigurable architectures. SIMD-based architectures typically consist of one or few high-performance SIMD DSP processors. The processors are usually connected together through a shared bus, and managed through a general purpose control processor. Some SIMD-based architectures also have a shared global memory connected to the bus. Processors that fall into this category are [17][34][8][13]. Reconfigurable architectures are usually made up of many simpler PEs. Depending on the particular design, these PEs range from fine-grain LUTs to coarser-grain ALU units and even ASICs. The PEs are usually connected together through a reconfigurable fabric. Processors that fall into this category are [20][19][3].

ARM's Ardbeg [34] is an example of a low power SIMD-based architecture that focused solely on 3G wireless communication. On top of the 3G requirements, AnySP was designed to deal with two more challenges: 1) higher bandwidths and more complex algorithms in 4G wireless; and 2) the support for video on the same substrate. The central challenge is simultaneously achieving both high performance and low power for mobile platforms. There is evidence to show that some companies are trying to reduce the number of distinct intellectual properties in their systems, because there is significant cost and power reduction. They are looking for convergent architectures that can not only execute the communication workload, but also handle some of the other applications like video and graphics. AnySP can be viewed as an example of such a convergent architecture that can support applications with similar characteristics within a limited power and cost budget.

6. CONCLUSION

Future uses for mobile devices will require more connectivity at higher data rates, support of high quality audio and video, as well as interactive applications. This increase in application diversity can be addressed by combining different processor types each tailored to a specific application. Such a solution is costly in terms of time, silicon area, and power. In this paper, we presented AnySP, a programmable and flexible SIMD architecture that is also low power. The major contributions of AnySP are the configurable SIMD datapath which supports wide and narrow vector lengths, flexible functional units which can fuse instructions together, temporary buffer and a bypass network which reduces register and memory accesses, SRAM-based swizzle network which reduces the power and latency of swizzle operations, and a multiple output adder tree with speeds up video applications. Results show AnySP's architectural design achieves 2-2.5x speedup over SODA for the set of MSP algorithms while also operating at an energy-delay profile 20-30% more efficient than SODA. AnySP meets the throughput requirement of 100 Mbps 4G wireless while consuming 1.3 W at 90 nm. This puts AnySP under the 1000 Mops/mW target but is close enough that in 45 nm process technology it will be able reach this milestone. Also, high quality H.264 4CIF video at 30 fps can be achieved with power consumption of 60 mW at 90 nm making AnySP a suitable candidate for mobile HD video processing.

7. ACKNOWLEDGEMENTS

We thank Ron Dreslinski for his help in organizing and editing this paper. We also thank the anonymous referees for their useful comments and suggestions. This research was supported by ARM Ltd. and the National Science Foundation under grants CNS-0615261, CSR-EHS-0615135, and CCF-0347411.

8. REFERENCES

[1] J. H. Ahn, W.J. Dally, B. Khailany, U.J. Kapasi, and A. Das. Evaluating the imagine stream architecture. In *Proc. of the 31st Intl. Symposium on Computer Architecture*, pages 14–24, Jun. 2004.

[2] ARM Ltd. *The ARM Architecture Version 6 (ARMv6)*, 2002. White Paper.

[3] R. Baines and D. Pulley. The picoArray and reconfigurable baseband processing for wireless basestations. In *Software Defined Radio*, February 2004.

[4] N. Clark, J. Blome, M. Chu, S. Mahlke, S. Biles, and K. Flautner. An architecture framework for transparent instruction set customization in embedded processors. In *ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture*, pages 272–283.

[5] H. Corporaal and H. (J.M.) Mulder. Move: A framework for high-performance processor design. In *Supercomputing '91: Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, pages 692–701, New York, NY, USA, 1991.

[6] W.J. Dally et al. Efficient embedded computing. *Computer*, 41(7):27–32, July 2008.

[7] K. Fan et al. Systematic register bypass customization for application-specific processors. *Proceedings. IEEE International Conference on Application-Specific Systems, Architectures, and Processors, 2003*, pages 64–74, June 2003.

[8] J. Glossner, E. Hokenek, and M. Moudgill. The sandbridge sandblaster communications processor. In

3rd Workshop on Application Specific Processors, pages 53–58, Sept. 2004.

[9] N. Goel, A. Kumar, and P. R. Panda. Power reduction in VLIW processor with compiler driven bypass network. In *VLSID '07: Proceedings of the 20th International Conference on VLSI Design held jointly with 6th International Conference*, pages 233–238, 2007.

[10] R. Golshan and B. Haroun. A novel reduced swing CMOS bus interface circuit for high speed low power VLSI systems. volume 4, pages 351–354 vol.4, May-2 Jun 1994.

[11] X. Guan and Y. Fei. Reducing power consumption of embedded processors through register file partitioning and compiler support. *International Conference on Application-Specific Systems, Architectures and Processors*, pages 269–274, July 2008.

[12] Nanoscale Integration and Modeling Group. *Predictive Technology Model*. http://www.eas.asu.edu/ ptm/.

[13] S. Knowles. *The SoC Future is Soft*. IEE Cambridge Branch Seminar 2005, Dec. 2005. http://www.iee-cambridge.org.uk/arc/seminar05/slides/SimonKnowles.pdf.

[14] 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.

[15] R. Krashinsky et al. The vector-thread architecture. In *Proceedings of the 31st Annual International Symposium on Computer Architecture, 2004.*, pages 52–63, June 2004.

[16] T. A. Lin, T. M. Liu, and C. Y. Lee. A low-power H.264/AVC decoder. *International Symposium on VLSI Design, Automation and Test, 2005.*, pages 283–286, April 2005.

[17] Y. Lin et al. SODA: A low-power architecture for software radio. In *Proc. of the 33rd Annual International Symposium on Computer Architecture*, pages 89–101, 2006.

[18] Y. Lin, S. Mahlke, T. Mudge, C. Chakrabarti, A. Reid, and K. Flautner. Design and implementation of Turbo decoders for software defined radio. *IEEE Workshop on Signal Processing Systems Design and Implementation, 2006. SIPS '06.*, pages 22–27, Oct. 2006.

[19] A. Lodi et al. Xisystem: A XiRisc-based SoC with reconfigurable IO module. *IEEE Journal of Solid-State Circuits*, 41(1):85–96, Jan. 2006.

[20] B.F. Mei et al. ADRES: An architecture with tightly coupled vliw processor and coarse-grained reconfigurable matrix. *13th International Conference on Field-Programmable Logic and Applications, 2003. FPL 2003*, pages 61–70, Sept. 2003.

[21] S. Park et al. Register file power reduction using bypass sensitive compiler. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(6):1155–1159, June 2008.

[22] S. Park, A. Shrivastava, N. Dutt, A. Nicolau, Y. Paek, and E. Earlie. Bypass aware instruction scheduling for register file power reduction. In *LCTES '06: Proceedings of the 2006 ACM SIGPLAN/SIGBED conference on Language, compilers, and tool support for embedded systems*, pages 173–181, New York, NY, USA, 2006.

[23] A. Peleg, S. Wilkie, and U. Weiser. Intel MMX for multimedia PCs. *Commun. ACM*, 40(1):24–38, 1997.

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

[25] P. Raghavan et al. A customized cross-bar for data-shuffling in domain-specific simd processors. In *ARCS*, volume 4415 of *Lecture Notes in Computer Science*, pages 57–68. Springer, 2007.

[26] U. Ramacher. Software-defined radio prospects for multistandard mobile phones. *Computer*, 40(10):62–69, Oct. 2007.

[27] S.K. Raman, V. Pentkovski, and J. Keshava. Implementing streaming simd extensions on the pentium III processor. *Micro, IEEE*, 20(4):47–57, Jul/Aug 2000.

[28] International Telecommunications Union M.1645 Recommendation. *Framework and overall objectives of the future development of IMT-2000 and systems beyond IMT-2000"*. http://www.ieee802.org/secmail/pdf00204.pdf.

[29] L. Seiler et al. Larrabee: A many-core x86 architecture for visual computing. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–15, New York, NY, USA.

[30] S. Seo, T. Mudge, Y. Zhu, and C. Chakrabarti. Design and analysis of LDPC decoders for software defined radio. *IEEE Workshop on Signal Processing Systems, 2007*, pages 210–215, Oct. 2007.

[31] H. Taoka, K. Higuchi, and M. Sawahashi. Field experiments on real-time 1-Gbps high-speed packet transmission in MIMO-OFDM broadband packet radio access. *IEEE 63rd Vehicular Technology Conference, 2006. VTC 2006-Spring.*, 4:1812–1816, May 2006.

[32] 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.

[33] M. Woh et al. The next generation challenge for software defined radio. In *Proc. 7th Intl. Conference on Systems, Architectures, Modelling, and Simulation*, pages 343–354, Jul. 2007.

[34] M. Woh et al. 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.