AnySP: Anytime Anywhere Anyway Signal Processing

LOOKING FORWARD, THE COMPUTATION REQUIREMENTS OF MOBILE DEVICES WILL INCREASE BY ONE TO TWO ORDERS OF MAGNITUDE, BUT THEIR POWER REQUIREMENTS WILL REMAIN STRINGENT TO ENSURE REASONABLE BATTERY LIFETIMES. SCALING EXISTING APPROACHES WON'T SUFFICE; INSTEAD, THE HARDWARE'S INHERENT COMPUTATIONAL EFFICIENCY, PROGRAMMABILITY, AND ADAPTABILITY MUST CHANGE. ANYSP, A FULLY PROGRAMMABLE ARCHITECTURE THAT TARGETS MULTIPLE APPLICATION DOMAINS, ADDRESSES THESE CHALLENGES FOR NEXT-GENERATION MOBILE SIGNAL PROCESSING.

In the past decade, mobile devices' proliferation has been spectacular. Worldwide, there are more than 3.3 billion active cell phones—devices that we all depend on daily. Current-generation devices use 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, and they waste silicon area and power.

The next generation of mobile computing will have higher data rates, increasingly complex algorithms, and greater computational diversity—but its power requirements will be just as stringent. Mobile devices perform signal processing as a primary computational activity owing to their heavy use of wireless communication and their rendering of audio and video signals. Fourth-generation (4G) wireless technology has been proposed with greatly increased bandwidth—100 Mbps for high mobility and 1 Gbps for low mobility. This translates to a 10 to 1,000 times increase in the computational requirements over previous third-generation (3G) wireless technologies with a power envelope that can only increase two to five times.[1] Other signal processing tasks, such as high-definition video, are also 10 to 100 times more compute-intensive than current mobile video and scaling existing approaches won't suffice.

The design of the next-generation mobile platforms must address three critical issues: efficiency, programmability, and adaptivity. Current 3G wireless technologies have inadequate computational efficiency; we must increase them by at least an order of magnitude for 4G. As a result, scaling 3G solutions by increasing the number of cores or the amount of data-level parallelism isn't enough. Programmability allows a single platform to support multiple applications and even multiple standards within each application domain. It also supports faster time-to-market and higher chip volumes, reducing manufacturing cost. Lastly, hardware adaptivity is necessary to maintain efficiency as the applications' core computational characteristics change. 3G solutions

Mark Woh
Sangwon Seo
Scott Mahlke
Trevor Mudge
University of Michigan, Ann Arbor

Chaitali Chakrabarti
Arizona State University

Krisztián Flautner
ARM, Ltd.

0272-1732/10/$26.00 © 2010 IEEE Published by the IEEE Computer Society

Table 1. Data-level parallelism analysis for mobile signal processing algorithms.

Algorithm	SIMD workload (%)	Scalar workload (%)	Overhead workload (%)*	SIMD width (elements)	Amount of thread-level parallelism
Fast Fourier transform/Inverse FFT (FFT/IFFT)	75	5	20	1,024	Low
Space–time block coding (STBC)	81	5	14	4	High
Low-density parity-check (LDPC)	49	18	33	96	Low
Deblocking filter	72	13	15	8	Medium
Intraprediction	85	5	10	16	Medium
Inverse transform	80	5	15	8	High
Motion compensation	75	5	10	8	High

* Overhead workload consists of the additional instructions needed to aid the single instruction, multiple data (SIMD) operations such as data shuffle and SIMD load/store.

rely heavily on the extensive amounts of vector parallelism in wireless signal processing algorithms, but they lose most of their efficiency when vector parallelism is unavailable or constrained, as in other application domains such as high-definition video.

To address these challenges, we present AnySP, an advanced signal processor architecture that targets next-generation mobile computing.[2] We aimed to create a fully programmable architecture that targets multiple application domains, specifically supporting 4G wireless communication and high-definition video decoding at efficiency levels of 1,000 million operations per second per milliwatt (MOPS/mW).

Mobile signal processing algorithms

To build a unified architecture for mobile signal processing (MSP), we performed a detailed analysis of the key kernel algorithms for two diverse workloads. These are fast Fourier transform (FFT), space–time block coding (STBC), and low-density parity-check (LDPC) for 4G wireless communication; and deblocking, intraprediction, inverse transform, and motion compensation for H.264 decoding. Analyzing their characteristics helped us define AnySP.

Multiple SIMD widths

Many previous studies have analyzed and determined the best single instruction, multiple data (SIMD) width for general-purpose and application-specific workloads. Table 1 presents the results of our study of MSP algorithms' data-level parallelism (DLP). We calculate the available DLP within each algorithm and show the maximum natural vector width that is achievable. We divide the instructions into three 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 aren't parallelizable and must be run on a scalar unit or the address generation unit (AGU).

From Table 1, we see that many algorithms have different natural vector widths—4, 8, 16, 96, and 1,024. Also, the algorithms with smaller SIMD widths exhibit a high level of thread-level parallelism (TLP), which means that we can process multiple threads that work on separate data on a wide SIMD machine. For example, we can process eight instances of STBC that have a SIMD width of 4 on a 32-wide machine. Unlike most SIMD architectures, which 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 TLP when the SIMD width is small.

Although the scalar and overhead workloads aren't the majority, they still contribute 20 to 30 percent of the total computation.

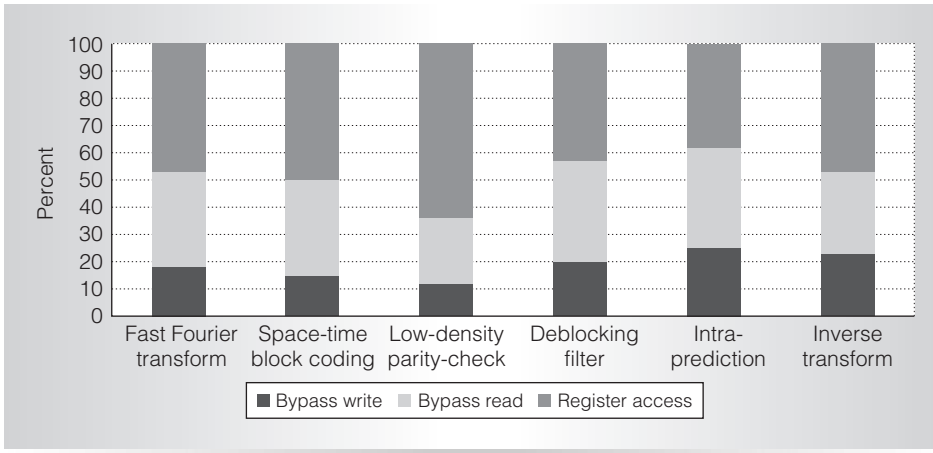

Figure 1. Register file access breakdown for the inner loops of mobile signal processing (MSP) algorithms. Bypass write and read are the register file read and write accesses that don't need to use the main register file. Register accesses are the read and write accesses that need to use the main register file.

For instance, in LDPC, data shuffling and memory operations dominate most of the workload. This suggests that we must reduce the overhead workload as well, by introducing better support for data reorganization or increasing the scalar and AGU performance.

Register value lifetimes

For many processors, the register file contributes as much as 10 to 30 percent of the total processing element power.[3] This is due to the reading and writing of values for each instruction. Many architectures, such as very large instruction word (VLIW), compound this problem by having multiported register files that increase the energy per access. In this article, we also study the potential for eliminating register file accesses in MSP applications. The MSP applications fit the streaming dataflow model, where produced data is consumed only once or twice. Usually, the instructions directly succeeding the producer consume the data. We build on the work by Goel et al.,[4] showing that bypassing the register file for short-lived values and storing the value within a register bypass pipeline can reduce power while sustaining performance. We also use register file partitioning, where the register file is split into regions and the high access registers are stored in the small partition, resulting in less energy being consumed per access.

To determine the extent to which we can reduce register file accesses, we analyze the MSP algorithms using the signal processing on demand architecture (SODA)[5] with a small, four-entry register file partition. The modified architecture has a total of 20 registers: 16 main registers and four temporary registers. Figure 1 shows the breakdown of accesses that can and can't bypass the register file. *Bypass write* and *bypass read* correspond to the register file read and write accesses that don't need to use the main register file and can be eliminated with either data forwarding or by storing the data in a small register file partition. *Register accesses* are the read and write accesses that can't be bypassed and need to use the main register file. For the MSP algorithms, with the exception of LDPC, many register accesses can be bypassed. Thus, we conclude that architectures running MSP algorithms should support bypass mechanisms to help reduce accesses to the main register file.

Instruction pair frequency

We performed a study on the MSP algorithms to find the most common producer-consumer instruction pairs. Among all the algorithms, the MAC instruction is the most frequent because many kernel algorithms have a dot product as their inner loop. Other common fused instruction

pairs are permute-add, add-add, and shift-add. The permute-add pair occurs in many algorithms because data must first be aligned to effectively use the SIMD lanes. The add-add and shift-add pairs also occur frequently in the video algorithms because values are accumulated and then normalized by a power-of-2 division—a shift.

Algorithm data-reordering patterns

Most commercial DSP and general-purpose processors support some form of data-reordering operations for multimedia applications. By data reordering, we mean both permutation and replication of data, also called *swizzle* operations in graphics. Intel's Larrabee has a hardware ''gather and scatter'' instruction that lets 16 distinct memory addresses build one SIMD vector and can take multiple cycles. SODA's data ''shuffle'' network can also take multiple cycles, although its power consumption is lower than Larrabee's.

Our study of the number of swizzle operations the MSP algorithms require showed that all the algorithms had a predefined set of swizzle operations, typically fewer than 10. Because we knew these beforehand, we didn't need sophisticated gather and scatter hardware support like in Larrabee. At the same time, a shuffle network like SODA's isn't complex enough to support the needed swizzle operations. We want an easy-to-configure network that can perform the operations in fewer cycles. Either a swizzle network or a more complex, programmable fixed-pattern network would be ideal.

AnySP processing element design

Our algorithm analysis studies provided us with five key insights for designing an efficient high-performance architecture for MSP applications:

- Opportunities for SIMD parallelism vary widely across the algorithms. Some have large inherent vectors, up to 1,024 elements in length. However, most algorithms have small to moderate vectors.
- Algorithms with smaller vector lengths frequently contain a high degree of identical threads, where each thread performs the same instructions, but on discontinuous data.
- 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 is used a large percentage of the time.
- Each algorithm uses a small set of predetermined swizzle patterns.

We applied these insights into our design of the AnySP processing element.

Figure 2 shows the processing element architecture, which consists of SIMD and scalar data paths. The SIMD data path consists of eight groups of 8-wide SIMD units, which can be configured to also create SIMD widths of 16, 32, and 64. Each of the 8-wide SIMD units is composed of groups of flexible functional units (FFU). The FFUs contain the functional units of two lanes that are connected through a simple crossbar. Eight SIMD register files feed the SIMD data path. Each register file contains 16 entries where each entry is 8 elements wide. 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, a multiple output adder tree can sum groups of 4, 8, 16, 32, or 64 elements and store the results into a temporary buffer.

The local memory consists of 16 memory banks; each bank consists of 8-element-wide SIMD data containing 256 16-bit entries, totaling 32 Kbytes of storage. Each group of 8-wide SIMD units has a dedicated AGU. When not in use, the AGU can run sequential code to assist the dedicated scalar pipeline. The AGU and scalar unit share the same memory space as the SIMD data path. To accomplish this, we use a scalar memory buffer that can store 8-wide SIMD locations. Because many algorithms access data sequentially, the buffer acts as a small cache that helps avoid multiple accesses to the vector banks. (We'll discuss these architectural features in more detail later.)

Configurable multi-SIMD width support

A large application can have kernel algorithms with different SIMD widths. Mapping them all onto a fixed-width SIMD

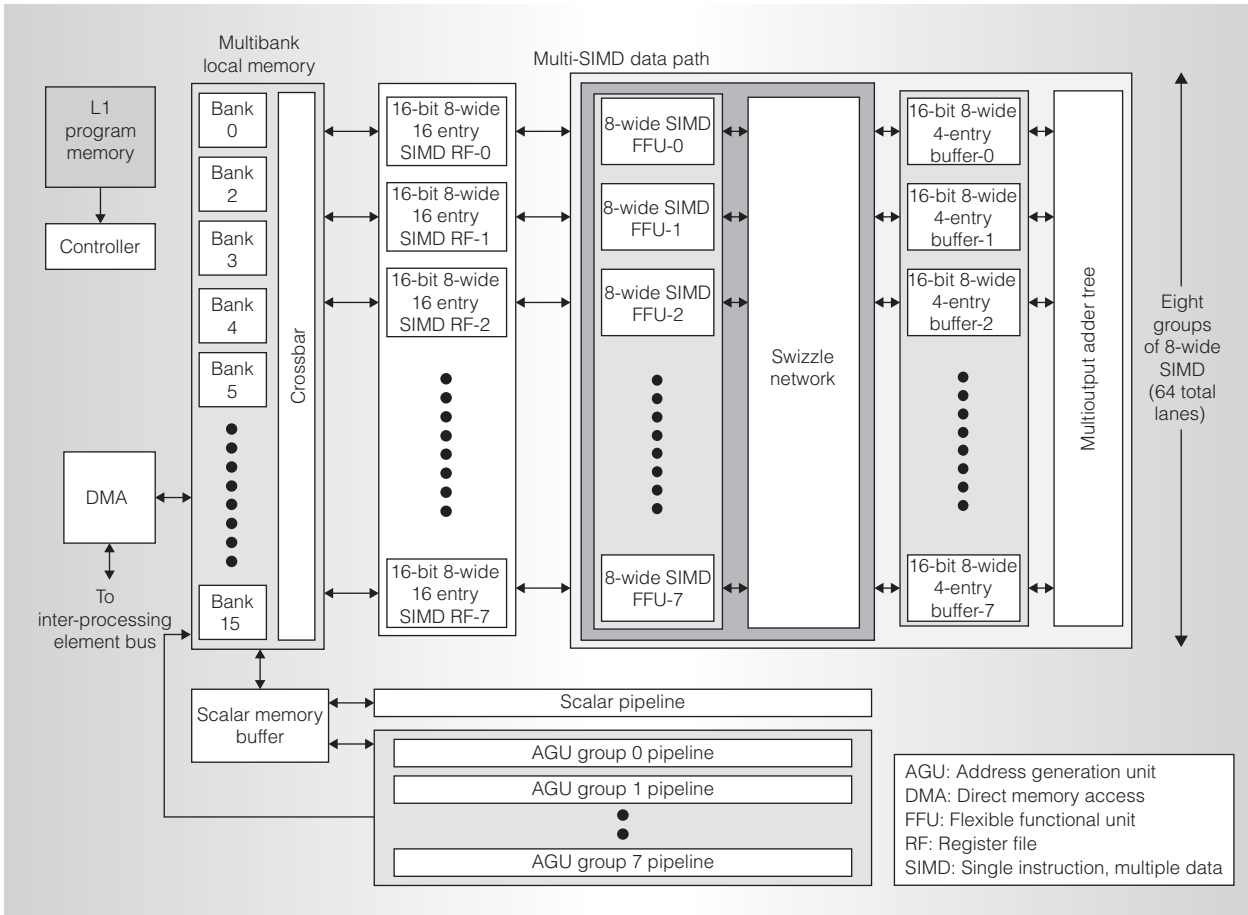

Figure 2. The AnySP processing element, which consists of SIMD and scalar data paths. The SIMD data path consists of eight groups of 8-wide SIMD units, which can be configured to create SIMD widths of 16, 32, and 64. Each of the 8-wide SIMD units is composed of groups of flexible functional units (FFUs). The local memory consists of 16 memory banks; each bank consists of 8-element-wide SIMD data containing 256 16-bit entries, totaling 32 Kbytes of storage.

architecture is typically inefficient. To improve SIMD utilization, we would need to use complex code transformations for the algorithm, which usually increases total computations and power consumption. In the applications we studied, independent threads weren't the only contributors to TLP. Specifically, in H.264, the same task is run many times for different macroblocks. Each task is independent of the others, runs the exact same code, and follows almost the same control path. The difference between each thread is that the data accesses from memory are different, which means separate memory addresses will need to be computed. To support these types of kernel algorithms, we designed 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 lets us exploit the DLP and TLP together for large and small SIMD width algorithms. Small SIMD width algorithms, such as intraprediction 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 such as FFT and LDPC can use the full 64-wide SIMD width configuration and run multiple iterations.

Temporary buffer and bypass network

We implemented temporary register buffers and a bypass network to lower register

file power. We implemented the temporary register buffers as a partitioned register file. The main register file still contains 16 registers. We added a second partition containing four registers to the design, making the total number of registers 20. This small partitioned register file shields the main register file from accesses by storing values that have very short lifetimes. In addition, reducing accesses to the main register file lessens the register file pressure, thereby decreasing memory accesses.

The bypass network modifies the write-back stage and forwarding logic. Typically in processors, data that's forwarded to another instruction to eliminate data hazards is also written back to the register file. In the bypass network, the programmer explicitly manages the forwarding and writing to the register file. This means that the instruction dictates whether the data should be forwarded and whether it should be written back to the register file. This eliminates register file writes for values that are immediately consumed, reducing register file power.

Flexible functional units

In typical SIMD architectures, we lose power and performance when the vector size is smaller than the SIMD width as a result of underutilized hardware. The proposed flexible functional units allow for reconfiguration and flexibility in operating different workload types. When SIMD use is low, the FFUs can combine the functional units between two lanes, which speeds 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 register file.

As Figure 3 shows, each 8-wide SIMD group is built from four 2-wide FFUs. Each functional unit consists of a multiplier, arithmetic logic unit (ALU), adder, and swizzle network subblock. Such a structure benefits algorithms with SIMD widths that are smaller than 64. In chained execution mode, a crossbar network can connect the functional units in two internal lanes. Overall, FFUs improve performance and reduce power by adding more flexibility and exploiting both TLP and DLP

Swizzle network

The number of distinct swizzle patterns needed for a specific algorithm is small, fixed, and known ahead of time. Previous research has addressed building application-specific crossbars for SIMD processors, which lack flexibility because they can't support new swizzle operations for applications that emerge post-fabrication.[6]

We propose using an SRAM-based swizzle network that adds flexibility while maintaining a customized crossbar's performance. The layout of the proposed network is similar to one by Golshan and Haroun in that it is an X-Y style crossbar in which the input buses are laid out horizontally and the outputs are laid out vertically.[7] Each intersection point between the input and output buses contains a pass transistor that's controlled by a flip flop. This allows for more flexibility because configurations can be changed post-fabrication. Instead of using a single flip flop cell, the proposed network uses SRAM cells, which can store multiple sets of swizzle configurations, allowing zero cycle delays in changing the swizzle pattern. Also, by storing multiple configurations, we reduce the controller complexity. Finally, 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. As a result, we can reduce the network's area and power consumption while still operating within a single clock cycle. For crossbar sizes larger than 32×32, the power is dramatically lower than the mux-based one and can run at almost twice the frequency. For instance, a 128×128 SRAM-based swizzle network consumes less than 30 percent of the power consumed by an equivalent mux-based crossbar.

The proposed SRAM-based swizzle network allows wide SIMD machines to be power efficient and fast. Although only a certain number of swizzle patterns can be loaded without reconfiguration, it's a viable solution because only a limited set of swizzle patterns are required for each algorithm. These swizzle patterns can be stored in the SRAM-based swizzle network configuration

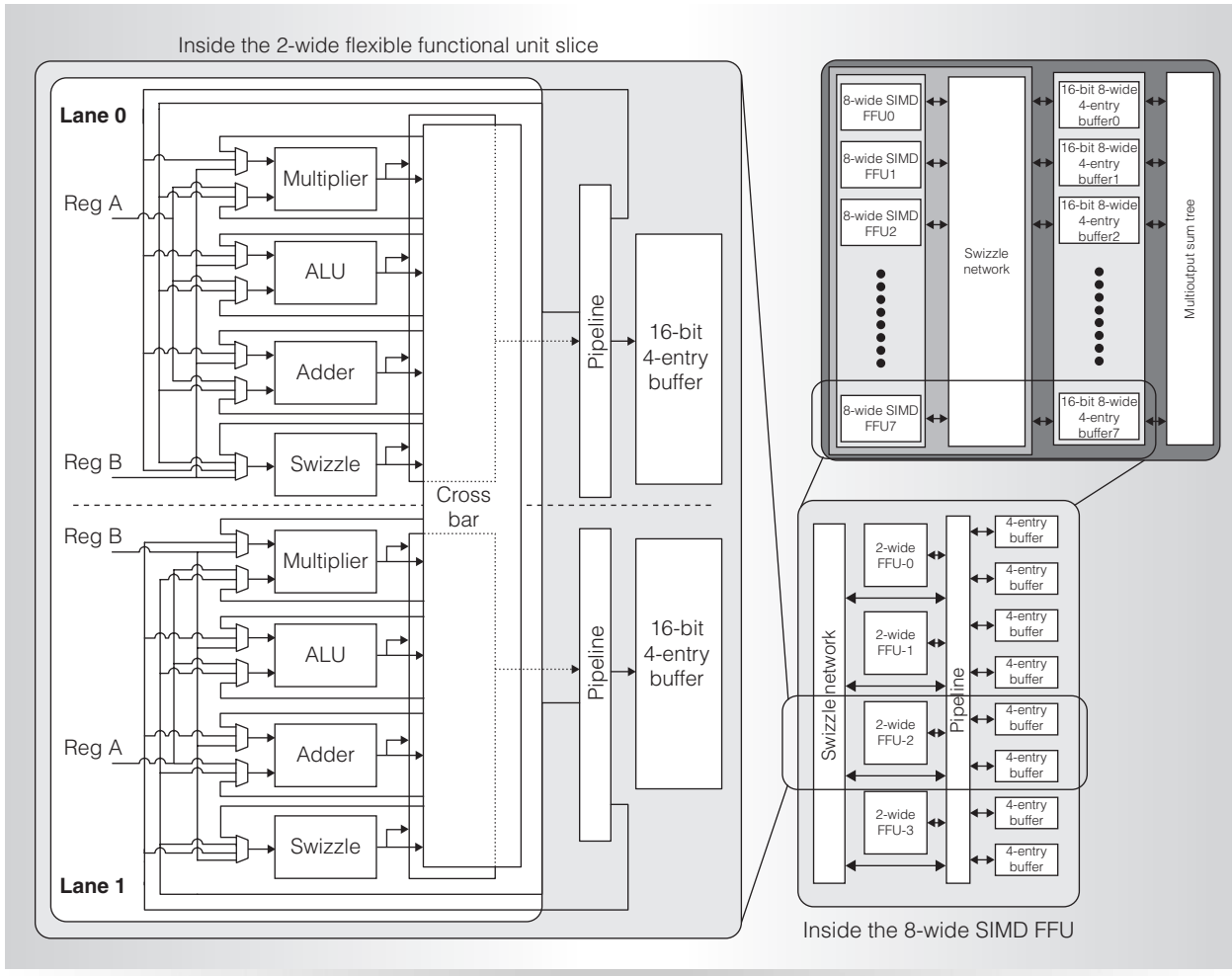

Figure 3. Flexible functional unit (FFU). Each functional unit consists of a multiplier, arithmetic logic unit (ALU), adder, and swizzle network subblock. ''Reg.'' is the register file.

memory at initialization. Because this swizzle network supports arbitrary swizzle patterns with multicasting capabilities, it provides more functionality than permutation networks found in typical SIMD architectures.

Multiple output adder tree support

SIMD architectures such as Lin et al.'s SODA have special SIMD summation hardware to perform ''reduction to scalar'' operations.[5] To compute this, adder trees sum up the values of all the lanes and store the result in the scalar register file. Although these techniques worked for 3G algorithms, in many of the video decoding algorithms, sums of less than the SIMD width are needed. For the wide-SIMD AnySP architecture, we designed the adder tree to allow for partial summations of 4, 8, 16, 32, or 64 elements, which then are written back to the temporary buffer unit. The summed values have short lifetimes, so writing into the temporary buffer unit helps reduce the number of read and write accesses to the register file.

Results

Figure 4 shows the speedup of AnySP over SODA for each algorithm. Each improvement is broken down by architectural enhancement: multiwidth SIMD, use of single-cycle SRAM-based swizzle network, FFU, and temporary buffer with the bypass network. We implemented the AnySP processor's main hardware components as RTL Verilog and synthesized them in TSMC 90 nm using the Synopsys physical compiler.

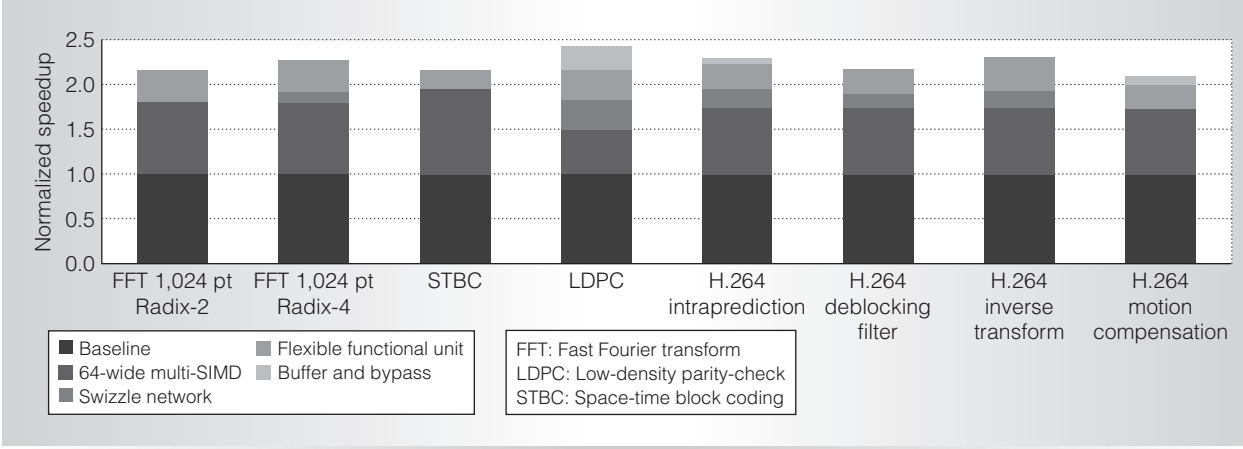

Figure 4. AnySP speedup over the SODA for the key algorithms used in 4G and H.264 benchmarks. The bars show the speedup for the different architectural enhancements—wider SIMD width, single-cycle synchronous RAM-based crossbar, fused-operations support, and buffer support.

Table 2. Processing element area and power summary for AnySP running 100-Mbps high-mobility 4G wireless and H.264 4CIF video at 30 fps.

			Area		4G + H.264 decoder	
	Components	**Units**	**Area (mm^2)**	**Area (%)**	**Power mW**	**Power (%)**
Processing element	SIMD data memory (32 Kbytes)	4	9.76	38.78	102.88	7.24
	SIMD register file (16 × 1,024 bits)	4	3.17	12.59	299.00	21.05
	SIMD arithmetic logic units, multipliers, and SIMD swizzle network	4	4.50	17.88	448.51	31.58
	SIMD pipeline + clock + routing	4	1.18	4.69	233.60	16.45
	SIMD buffer (128 bytes)	4	0.82	3.25	84.09	5.92
	SIMD adder tree	4	0.18	<1.0	10.43	<1.0
	Intraprocessor interconnect	4	0.94	3.73	93.44	6.58
	Scalar/address generation unit pipeline and misc.	4	1.22	4.85	134.32	9.46
System	ARM (Cortex-M3)	1	0.6	2.38	2.5	<1.0
	Global scratchpad memory (128 Kbytes)	1	1.8	7.15	10	<1.0
	Interprocessor bus with direct memory access	1	1.0	3.97	1.51	<1.0
Total	90 nm (1 V at 300 MHz)		25.17	100	1,347.03	100
Estimate	65 nm (0.9 V at 300 MHz)		13.14		1,091.09	
	45 nm (0.8 V at 300 MHz)		6.86		862.09	

Our in-house architecture emulator tool extracted the timing and power numbers from the synthesis and used them to calculate the timing and power values for each kernel.

Table 2 shows AnySP's power and area when running both 100-Mbps, high-mobility 4G wireless and high-quality H.264 4CIF video. AnySP met the throughput requirement of 100-Mbps 4G wireless while consuming 1.3 W at 90 nm. This is just below the 1,000 MOPS/mW target, but close enough to meet that target in

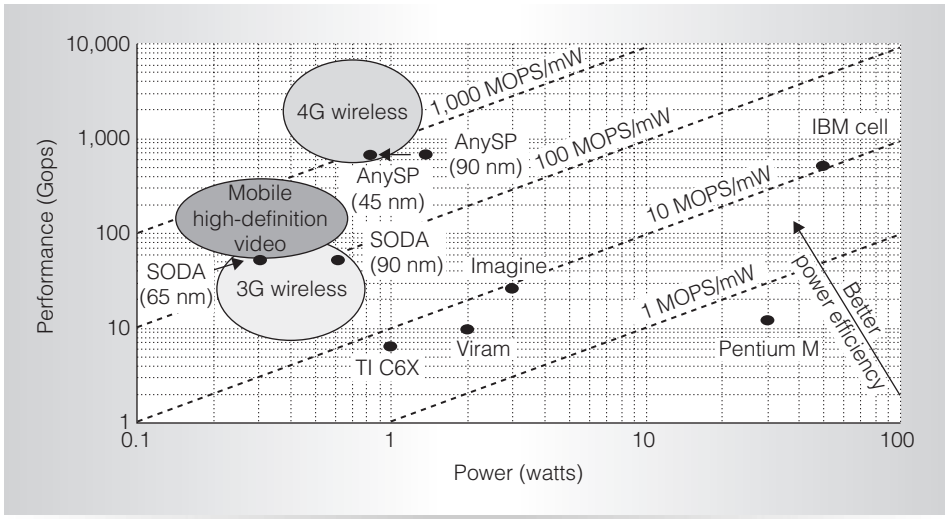

Figure 5. Performance versus power landscape for various mobile computing systems.

45-nm process technology. It can also achieve high-quality H.264 4CIF video at 30 fps with 60 mW at 90 nm, meeting the requirements for mobile HD video. AnySP's power breakdown shows that the SIMD functional units are the dominant power consumers. The register file and data memory are lower, proving our design's efficiency because the largest portion of the power is spent doing actual computations. In comparison, SODA's register file and data memory power are the dominant consumers, suggesting that much 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 improve our design's efficiency.

Comparative analysis

We can classify current solutions supporting 3G wireless protocols in SDR as SIMD-based architectures or reconfigurable architectures. SIMD-based architectures usually consist of multiple processors connected through a shared bus and managed through a general-purpose control processor. Processors falling into this category are SODA, ARM's Ardbeg, Sandbridge's Sandblaster, and Icera's DXP. Reconfigurable architectures usually consist of many simpler processing elements, ranging from fine-grain lookup tables to coarser-grain ALU units and even application-specific integrated circuits. The processing elements usually connect through a reconfigurable fabric. Processors that fall into this category are IMEC's architecture for dynamically reconfigurable embedded system (ADRES) and picoChip's picoArray.

Many architectures have tried to exploit DLP and TLP to increase power efficiency. For instance, the vector-thread architectures 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 combine to execute complex subgraphs or simultaneously operate on multiple vectors using the FFU. Maintaining the single-instruction operation mode translates into gains in power efficiency compared to vector-thread architecture.

We designed AnySP to demonstrate how we can achieve power efficiency while implementing complex 4G and video decoding algorithms in the same substrate. Companies are now trying to reduce the number of distinct intellectual property blocks in the system and deriving convergent architectures that can execute diverse workloads spanning communication, video, and graphics. AnySP can be viewed as an example of such a convergent architecture because it supports applications with similar characteristics within a limited power and cost budget.

Figure 5 summarizes AnySP's performance versus power consumption with respect to current processors aimed at a

similar application domain. It also shows AnySP's performance with respect to the requirements of 3G and 4G wireless protocols and mobile HD video.

Future uses for mobile devices will require more connectivity at higher data rates and support for high-quality audio and video, as well as interactive applications. These advanced features will require supercomputer-like processing requirements at mobile device power constraints. We believe that adopting the AnySP design approach, which uses algorithm characteristics to define the architectural features, will be essential.

Our future work tries to combine more signal processing functionality with higher energy efficiency. We propose using Near Threshold Computing techniques[8] to provide multi-mode energy efficient operating points. MICRO

Acknowledgments

This research was supported by ARM Ltd. and the National Science Foundation under grants CNS-0615261, CSR-EHS-0615135, and CCF-0347411.

References

1. M. Woh et al., ''The Next Generation Challenge for Software Defined Radio,'' *Proc. 7th Int'l Workshop Embedded Computer Systems: Architectures, Modeling, and Simulation,* LNCS 4599, Springer, 2007, pp. 343-354.
2. M. Woh et al, ''AnySP: Anytime Anywhere Anyway Signal Processing,'' *Proc. 36th Ann. Int'l Symp. Computer Architecture,* ACM Press, 2009, pp. 128-139.
3. X. Guan and Y. Fei, ''Reducing Power Consumption of Embedded Processors through Register File Partitioning and Compiler Support,'' *Proc. 2008 Int'l Conf. Application-Specific Systems, Architectures and Processors,* IEEE CS Press, 2008, pp. 269-274.
4. N. Goel, A. Kumar, and P.R. Panda, ''Power Reduction in VLIW Processor with Compiler Driven Bypass Network,'' *Proc. 20th Int'l Conf. VLSI Design Held Jointly with 6th Int'l Conf. Embedded Systems* (VLSID 07), IEEE CS Press, 2007, pp. 233-238.
5. Y. Lin et al., ''SODA: A Low-Power Architecture for Software Radio,'' *Proc. 33rd Ann. Int'l Symp. Computer Architecture,* IEEE CS Press, 2006, pp. 89-101.
6. P. Raghavan et al., ''A Customized Cross-bar for Data-Shuffling in Domain-Specific SIMD Processors,'' *Architecture of Computing Systems* (ARCS 07), LNCS 4415, Springer, 2007, pp. 57-68.
7. R. Golshan and B. Haroun, ''A Novel Reduced Swing CMOS Bus Interface Circuit for High Speed Lowpower VLSI Systems,'' *Proc. 1994 IEEE Int'l Symp. Circuits and Systems* (ISCAS 94), vol. 4, IEEE CS Press, 1994, pp. 351-354.
8. B. Zhai et al., ''Energy Efficient Near-Threshold Chip Multi-processing,'' *Proc. 2007 Int'l Symp. Low Power Electronics and Design,* ACM Press, 2007, pp. 32-37.

Mark Woh is a PhD candidate in electrical engineering at the University of Michigan, Ann Arbor. His research interests include low-power microarchitecture, wireless communications, and signal processing language and compiler design. Woh has a BS in electrical engineering and computer engineering from the University of Michigan, Ann Arbor.

Sangwon Seo is a PhD candidate in electrical engineering and computer science at the University of Michigan, Ann Arbor. His research interests include low-power microarchitecture and wireless signal processing. Seo has a BS in electrical engineering from Seoul National University.

Scott Mahlke is an associate professor in the Electrical Engineering and Computer Science Department at the University of Michigan, Ann Arbor, where he leads the Compilers Creating Custom Processors group. His research interests include compilers for multicore processors, application-specific processors for mobile computing, and reliable system design. Mahlke has a PhD in electrical engineering from the University of Illinois, Urbana-Champaign.

Trevor Mudge is the first Bredt Family Professor of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor. His research interests

include computer architecture, CAD, and compilers. Mudge has a PhD in computer science from the University of Illinois, Urbana-Champaign. He is a member of the ACM, Institution of Engineering and Technology, and British Computer Society, and is a Fellow of the IEEE.

Chaitali Chakrabarti is a professor of electrical engineering at Arizona State University. Her research interests are in the areas of VLSI architectures and algorithms for signal processing and communications, and low-power system design. Chakrabarti has a PhD in electrical engineering from the University of Maryland, College Park.

Krisztián Flautner is vice president of research and development at ARM, Ltd. His research interests include high-performance, low-energy processing platforms that support advanced software environments. Flautner has a PhD in electrical engineering and computer science from the University of Michigan, Ann Arbor. He is a member of the ACM and IEEE.

Direct questions and comments to Mark Woh, ACAL LAB—EECS Dept., Univ. of Michigan, 4856 CSE Bldg., 2260 Hayward Ave., Ann Arbor, MI 48109; mwoh@umich.edu.

cn *Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.*