Design Space Exploration for Real-Time Embedded Stream Processors

This tool explores tradeoffs between organization and number of ALUs and clock frequency in a stream processor. The tool provides candidate low-power configurations and estimates of their real-time performance. The tool relates instruction-level, subword, and data parallelism to functional units' organization and utilization. The exploration methodology is applicable to all embedded-processor designs in signal and media processing.

Sridhar Rajagopal
WiQuest Communications

Joseph R. Cavallaro
Scott Rixner
Rice University

• • • • • • Progress in processor technology and increasing consumer demand have created interesting possibilities for using embedded processors in a variety of platforms. Embedded processors are now supporting real-time, high-performance digital signal-processing applications such as video, image processing, and wireless communications. The use of programmable processors in such applications poses new challenges for embedded-system designers. Although programmable embedded processors trade power efficiency for flexibility in custom solutions, power awareness is an important goal in embedded-processor designs.

Stream processors are digital signal processors (DSPs) targeted at high-performance embedded applications.[1] They contain clusters of functional units and provide a bandwidth hierarchy, supporting hundreds of arithmetic units. They exploit instruction-level parallelism (ILP) and subword parallelism (SP) within a cluster and data parallelism (DP) across clusters. There is no clear methodology for designing an embedded stream processor that meets performance requirements and provides power efficiency for a workload with a certain real-time design constraint. The designer can vary the number of clusters, the number of arithmetic units, and the clock frequency to meet real-time constraints, and each factor can significantly affect power consumption.

The large architecture design space[2] and the inability of compilers to automatically exploit data parallelism limit an exhaustive simulation for exploration, necessitating hand optimizations for performance. Moreover, embedded applications such as wireless systems are evolving rapidly.[3] Therefore, designers must evaluate various candidate algorithms for future systems and would like to get a quick estimate of the lowest-power embedded processor that meets real-time requirements for each candidate. In this article, we present a tool that explores the

Published by the IEEE Computer Society

choice of ALUs within each cluster, the number of clusters, and the real-time clock frequency that will minimize the stream processor's power consumption. The tool provides candidate low-power configurations, which the designer can then refine with detailed, cycle-accurate simulations to ensure that the design meets real-time specifications. (The "Related work" sidebar briefly describes other design exploration techniques for embedded processors.)

Stream processor characteristics and applications

Figure 1 shows the various parallelism levels exploited in embedded processors. Traditional programmable embedded processors such as DSPs exploit ILP and SP.[4] Embedded stream processors exploit DP, in addition to ILP and SP, enabling high-performance programmable architectures with hundreds of ALUs.

Imagine is an example of an embedded stream processor.[1] We used this architecture and its simulator to evaluate the design methodology presented here. The Imagine simulator is programmed in a high-level

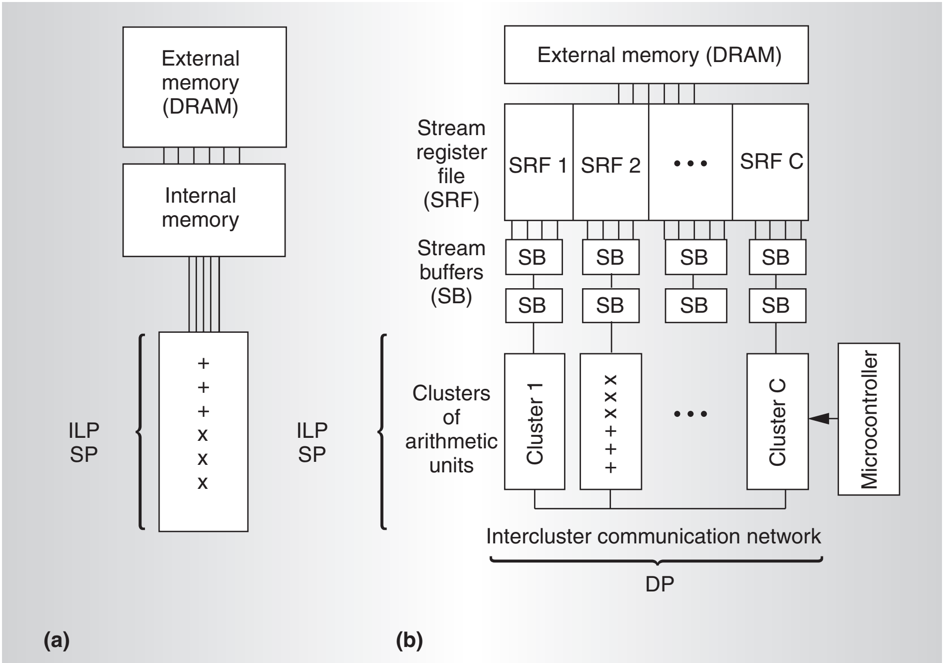

Figure 1. Parallelism levels in traditional embedded DSPs (a) and data-parallel embedded stream processors (b).

Related work

The combination of completely programmable solutions and the need for high performance presents new challenges for embedded-system designers, who have traditionally focused on exploring heterogeneous solutions to find the best flexibility, performance, and power tradeoffs. Researchers have studied design space exploration for performance and power in very-long-instruction-word (VLIW)-based embedded processors.[1,2] These design exploration techniques address single-cluster stream processors. However, exploring the number of clusters in the design adds an additional dimension to the search space. How to partition the arithmetic units into clusters and the number of arithmetic units to put in each cluster are not yet clear. Researchers have also studied design space exploration based on linear-programming methods for on-chip multiple-instruction, multiple-data (MIMD) multiprocessors.[3] This technique attempts to find the right number of processors (clusters) for performance and energy constraints, assuming a fixed configuration for cluster parameters. Using either of these exploration techniques for stream processors requires a more exhaustive and complex search for simultaneous optimization of the number of clusters and the number and type of arithmetic units in a cluster. The tradeoffs between exploiting ILP within a cluster and across clusters increases the exploration's complexity. In contrast, our new design space exploration tool exploits data parallelism across stream processor clusters to provide a simpler method of finding the right number of clusters and the number and types of arithmetic units in a cluster.

References

1. V.S. Lapinskii, M.F. Jacome, and G.A. de Veciana, "Application-Specific Clustered VLIW Datapaths: Early Exploration on a Parameterized Design Space," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 8, Aug. 2002, pp. 889-903.
2. J. Kin et al., "Power-Efficient Media Processors: Design Space Exploration," *Proc. 36th ACM/IEEE Design Automation Conf.* (DAC 99), IEEE Press, 1999, pp. 321-326.
3. I. Kadayif, M. Kandemir, and U. Sezer, "An Integer Linear Programming Based Approach for Parallelizing Applications in On-Chip Multiprocessors," *Proc. 39th ACM/IEEE Design Automation Conf.*, IEEE Press, 2002, pp. 703-708.

language and allows the programmer to modify features such as number and type of ALUs and their latency. The cycle-accurate simulator and retargetable compiler also give detailed insights into ALU utilization, memory stalls, and algorithm execution time. A power consumption and VLSI scaling model is also available to give a complete picture of the resulting architecture's power and performance.[5]

Stream operations all consume and/or produce streams that are stored in the centrally located stream register file (SRF). The two major stream instructions are memory transfers and kernel operations. A stream memory transfer either loads an entire stream into the SRF from external memory or stores an entire stream from the SRF to external memory. Multiple stream memory transfers can occur simultaneously, as hardware resources allow. Memory transfers to the SRF are decoupled from cluster operations. The SRF can prefetch data for future kernels while the current kernel is executing on the clusters. A kernel operation performs a computation on a set of input streams to produce a set of output streams.

Kernel operations are performed within a data-parallel array of arithmetic clusters. Each cluster performs the same sequence of operations on independent stream elements. Both the SRF and the stream buffers are banked to match the number of clusters. Hence, kernels that need to access data in other SRF banks must use the intercluster communication network for communicating data between clusters. Each cluster contains a set of ALUs controlled in a very-long-instruction-word (VLIW) fashion by the microcontroller. The microcontroller is an instruction sequencer that orders the kernel operations in the clusters. The intercluster communication unit supports applications that are not perfectly data parallel and that must communicate variables or data across clusters.

Embedded stream processors are promising solutions for meeting wireless communication systems' requirements of high performance and power efficiency. During the last few years, wireless systems' data rates have increased from kilobits per second (Kbps) for voice applications to megabits per second (Mbps) for multimedia applications. Moreover, sophisticated signal-processing algorithms are now used for reliably processing information received over the wireless channel. Because of the increased number of bits requiring processing in unit time, along with increased computational complexity, real-time processing now requires the support of hundreds of arithmetic units. Wireless systems also require greater flexibility to support different environments and wireless standards and to let designers quickly design, evaluate, and implement algorithms for these systems. DSPs typically use fixed-function coprocessors for complex tasks, such as Viterbi decoding. In contrast, a stream processor achieves scalability by using a uniform programmable structure, rather than dedicated coprocessors. Such scalability allows the efficient mapping of a wider range of algorithms to the stream processor.

To evaluate our design methodology, we chose a wireless base station as a design workload. The base station requires 24 billion computations per second,[3] employing sophisticated signal-processing algorithms such as multiuser estimation, multiuser detection, and Viterbi decoding, and provides a data rate of 128 Kbps per user for 32 users.

System model for design exploration

The execution time of signal-processing workloads is fairly predictable at compile time because of their static nature. A workload's execution time has two parts: computations (t_{compute}) and stalls (t_{stall}). Stalls can occur during memory accesses or during microcontroller sequencing of kernel operations in the clusters. Memory stalls are difficult to predict at compile time because the exact area of overlap between memory operations and computations is determined only at runtime. Microcontroller stalls depend on the data bandwidth required by the arithmetic units in the clusters and vary with the algorithms, the number of clusters, and the availability of data in internal memory. Some parts of memory and microcontroller stalls are constant because of internal-memory-size limitations or bank conflicts and don't change with the computations. As the addition of arithmetic units decreases computation time, some memory stalls become exposed and are thus variable with the part of the real-time frequency needed for computations, f_{compute}. The real-time frequency needed to account for constant memory stalls is denoted $f_{\text{stall-min}}$. The worst-case memory stall, $f_{\text{stall-max}}$, occurs when the processor exploits all the ILP, DP, and SP, thus changing the problem from compute bound to memory bound. Hence, memory stall time is bounded by $f_{\text{stall–min}}$ and $f_{\text{stall–max}}$, and the clock frequency needed to meet real-time requirements is

$$f = f_{\text{compute}} + f_{\text{stall}} \tag{1}$$

where

$$f_{\text{stall–min}} \leq f_{\text{stall}} \leq f_{\text{stall–max.}}$$

Signal-processing algorithms tend to have significant amounts of DP.[1,3,5] ILP exploitation in a cluster is limited because of the cluster's finite resources, such as finite register sizes, intercluster communication bottlenecks, and a finite number of input read and output write ports. Increasing some of the resources, such as register file sizes, is less expensive than adding an extra intercluster communication network, which can significantly affect the chip's wiring layout and power consumption.[5] However, any one of the finite resources could restrict ILP. Also, it is difficult to exploit ILP across basic blocks in applications with multiple loops. DP is available even after exploiting ILP, and designers can use the remaining DP to set the number of clusters in the processor.

Data parallelism

We define *DP* as the number of data elements that require exactly the same operations to be performed in an algorithm. DP is architecture independent. We also define a new term, cluster data parallelism ($CDP \leq DP$): the parallelism available in the data after exploiting SP and ILP. Thus, CDP is the maximum DP that can be exploited across clusters without significant decrease in ILP or SP.

Because of limited resources in a cluster, stream processors cannot exploit all DP as ILP through loop unrolling. The unused DP can be exploited across clusters as CDP. This observation lets us set the number of clusters according to the CDP and set the ALUs in the clusters according to ILP and SP, decoupling the joint exploration of clusters and ALUs in a cluster into independent problems. Thus, we drastically reduce the exploration space and programming effort for various cluster configurations. We can demonstrate the observation with an example of the Viterbi decoding algorithm used in wireless communication systems.

Figure 2 shows the performance of sequential Viterbi decoding with increasing clusters in a processor with 32 users, assuming a constant configuration of three adders and three multipliers in a cluster. The DP in Viterbi decoding is proportional to the constraint length, *K*, which is related to the strength of the error control code. A Viterbi decoder with $K = 9$ has $2^{K-1} = 256$ states and thus has a DP of 256 and can use 8-bit precision to pack four states in one cluster (SP = 4), reducing the CDP to $2^{K-3} = 64$. Hence, increasing the number of clusters beyond 64 does not provide performance benefits. However, as the clusters decrease from 64 to four for $K = 9$, there is an almost linear relationship between clusters and execution time, showing that we can approximate the ILP and SP we are exploiting as being independent of the CDP. The deviation of the performance curve with clusters from a slope of –1 represents the variation of ILP with CDP. We obtain similar curves for lower constraint lengths of 5 and 7, which have less DP and hence do not yield performance benefits over four and 16 clusters respectively.

Interestingly, the figure shows that stream processors and ASICs (coprocessors) exhibit the same characteristics while exploiting DP in the add-compare-select (ACS) computations. Typical Viterbi decoders can decode one bit every clock cycle.[6] The figure assumes that the ASIC can decode $DP/2^{K-3}$ bits every clock cycle. The effective DP for a decoder with constraint length K is 2^{K-1}. However, the stream processor implementation exploits SP and performs four ACS operations in one cluster. Hence, to make the plot scale the same for both implementations, we grouped four ACS units in the ASIC as a single ACS unit, giving rise to a net maximum DP on the graph of 2^{K-3}. The performance difference between the stream processor implementation and the ASIC implementation is two orders of magnitude, whereas the difference between the stream processor implementation and the single-processor C6416 TI DSP software implementation is one to two orders of magnitude. The single dot in the figure represents a software DSP implementation of Viterbi decoding with K of 9 without coprocessors. The lack of DP exploitation in DSPs accounts for their difference in performance from stream processors.

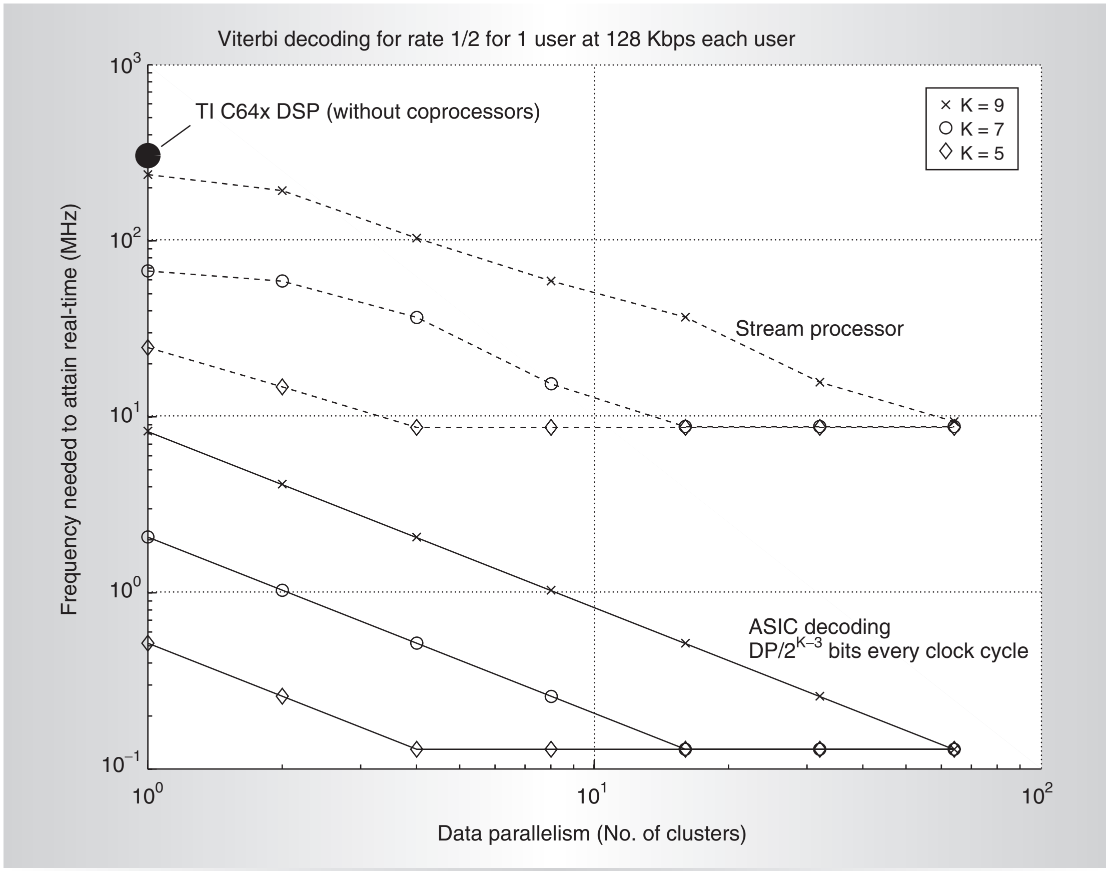

Figure 2. Performance of Viterbi decoding on DSPs, stream processors, and ASICs (coprocessors).

Design exploration framework

Assume the ALUs in the embedded stream processor are solely adders and multipliers. Let (*a*, *m*, *c*) be the number of adders per cluster, the number of multipliers per cluster, and the number of clusters. Let *f* be the processor's minimum clock frequency that meets the application's real-time requirements. Our design goal is to find (*a*, *m*, *c*, *f*) such that power *P*(*a*, *m*, *c*, *f*) is minimized as follows:

$$\min_{a,m,c,f} P = \min_{a,m,c,f} C(a,m,c) V^2 f(a,m,c) \quad (2)$$

where $C(a, m, c)$ is the loading capacitance, V is the supply voltage, and $f(a, m, c)$ is the clock frequency needed to meet real-time requirements. To estimate the capacitance and voltages for (*a*, *m*, *c*), we use a derivation for stream processor energy from two other publications, which is based on capacitance values extracted from the Imagine Stream Processor fabrication.[5,7]

Sensitivity analysis

The use of dynamic voltage scaling (DVS) has become very popular for power savings in embedded-processor designs. To analyze the design's sensitivity to clock frequency and voltage, we assume the following:

$$\min_{a,m,c} P = \min_{a,m,c} C(a,m,c) f^p(a,m,c) \quad (3)$$

where $(2 \le p \le 3)$, and p is the variation of power with respect to frequency. Typically, voltage scaling is assumed to be linear with frequency, providing cubic power savings. However, the actual relationship between power and frequency depends on several factors, including technology, the need for the clock to be an integral multiple of the I/O and memory bus speed, and the range over which clock frequency is assumed to vary. Physical realizations of processors employing DVS, such as Intel's XScale and Transmeta's Crusoe, show slopes between quadratic and cubic dependence of power on frequency, depending on the frequency range over which the DVS is applied.[8] The design exploration tool allows the designer to set p according to the technology and explore the design's sensitivity to p. Our design space exploration tool optimizes Equation 3 to find the number and organization of the arithmetic units that minimize power consumption.

Similarly, ALUs in an actual physical realization can have different power consumption values from those used in a power model. If we assume two types of ALUs, such as adders and multipliers, in the design, we must model one type's power consumption relative to the other's to make the values used independent of technologies and actual implementations. Adder and multiplier power consumptions are linear and quadratic, respectively, with bit width.[9] Thus, 32-bit adder designs and 32×32 multiplier designs can have relative power ratios varying between 2 and 4 percent after their bit widths are normalized to 32.[10] Hence, using equally aggressive 32-bit adders and 32×32 multipliers, we will assume that adder power variations are between 0.01 and 0.1 of multiplier power. As we will show, this variation is not critical because the additional register files and the intracluster communication network added with the ALUs dominate power consumption, not the ALUs alone. The adder power, P_{adder}, can be given by

$$P_{\text{adder}} = \alpha \times P_{\text{multiplier}} \tag{4}$$

where α is the adder-to-multiplier power ratio and $(0.01 \le \alpha \le 0.1)$.

The stream processor's organization provides a bandwidth hierarchy, which allows prefetching of data and mitigates memory stalls in the processor.[11] Memory stalls account for 5 to 16 percent of total execution time in media-processing workloads[11] and 20 percent of execution time in wireless communication workloads.[7] Stalls in stream processors are due to waits for memory transfers (both external memory and microcontroller stalls), inefficiencies in software pipelining, and time spent dispatching microcontroller code from the host processor to the microcontroller.[11] To model memory stalls and observe the design's sensitivity to stalls, we model the worst-case stall $f_{\text{stall-max}}$ as 25 percent of the workload at the minimum clock frequency needed for real time, $f = f_{\min}$, where the entire available ILP, DP, and SP are exploited. We model variations in memory and microcontroller stalls using parameter β between 0 and 1 to explore the design tools' sensitivity to stalls. Hence, from equation 1 we get

$$f_{\text{stall}} = (1 - \beta) f_{\text{stall-max}} \tag{5}$$

where $(0 \le \beta \le 1)$.

$$f_{\text{stall}} = 0.25(1 - \beta) f_{\min} \tag{6}$$

where $\beta = 1$ represents the no-stall case and $\beta = 0$ represents worst-case memory stall $f_{\text{stall-max}}$. We compute the minimum real-time frequency, $f_{\min}$, during design exploration.

Design space exploration

We start design exploration with an overprovisioned hypothetical architecture that has infinite clusters and infinite ALUs in each cluster. We then revise this architecture by decreasing the clusters and ALUs to find smaller configurations until the real-time frequency begins to increase. This revised architecture configuration still exploits all the possible ILP, SP, and DP available in the algorithms. We denote this configuration as $a_{\max}$, $m_{\max}$, max(cdp). From this point on, we explore the tradeoffs between frequency and capacitance defined in Equation 3 to find the configuration that attains real time at the lowest power.

Setting the number of clusters

The workload, W, consists of L algorithm kernels executed sequentially on the data-parallel embedded processor; this workload is denoted $k_1, k_2, \ldots, k_L$. The kernels' respective execution times are $t_1(a, m, c), t_2(a, m, c), \ldots, t_L(a, m, c)$. We define the cluster data parallelism in each kernel as $cdp_1, cdp_2, \ldots, cdp_L$. To find the number of clusters needed, we compile all kernels at their maximum CDP levels, assuming a sufficiently large number of adders and multipliers per cluster. We run kernel i with $a_{\max}, m_{\max}, cdp_i$, where $a_{\max}$ and $m_{\max}$ are a sufficiently large number of adders and multipliers per cluster to exploit the available ILP in all kernels. The compile time execution for kernel i is $t_i(a_{\max}, m_{\max}, cdp_i)$.

Hence, real-time frequency $f(a, m, c)$ is

$$f(a, m, c)(\text{MHz}) = \textit{Real-time target}\ (\text{Mbps}) \times \textit{Execution time per bit}\ (a, m, c). \quad (7)$$

The minimum real time frequency is given by

$$f_{\min} = \textit{Real-time target} \times \textit{Execution time per bit}\ [a_{\max}, m_{\max}, \max(cdp)]. \quad (8)$$

The real-time frequency, including the memory stalls, is then given by

$$f(a_{\max}, m_{\max}, c) = f_{\text{stall}} + \textit{Real-time target} \times \sum_{i=1}^{L} \left\lceil \frac{cdp_i}{c} \right\rceil t_i(a_{\max}, m_{\max}, cdp_i). \quad (9)$$

Equation 9 reduces frequency by half using cluster doubling; we based this equation on the observation of linear frequency benefits from clusters within the CDP range. If the number of clusters chosen is greater than the available CDP, there is no reduction in execution time. The f_{stall} term accounts for execution time stalls not predicted at compile time; we compute f_{stall} using equations 8 and 1. The number of clusters that minimizes power consumption is

$$\min_{c,f} P\left(a_{\max}, m_{\max}, c, f\right) = \min_{c,f} C\left(a_{\max}, m_{\max}, c\right) f^{p}\left(a_{\max}, m_{\max}, c\right). \quad (10)$$

Thus, by computing $f(a_{\max}, m_{\max}, cdp)$ at compile time and plotting this function for the desired range of clusters and for varying p, we compute the number of clusters that will minimize power consumption. The choice of clusters is independent of adder-to-multiplier power ratio α because all clusters have the same α and the same number of adders and multipliers.

Setting the number of ALUs per cluster

Once we have set number of clusters, c, we must decide the number of ALUs in each cluster. We now vary the number of adders and multipliers from (1, 1) to $(a_{\max}, m_{\max})$ to find the tradeoff point that minimizes the processor's power consumption. The design tool can handle varying ALUs without any changes in the application code. The design tool also provides information based on the schedule about ALU efficiencies. Hence, we can now perform an exhaustive compile time search within this constrained space to find the number of adders and multipliers that meets real time with minimum power consumption. We obtain power minimization using

$$\min_{a,m,f} P\left(a, m, c, f\right) = \min_{a,m,f} C\left(a, m, c\right) f^{p}\left(a, m, c\right). \quad (11)$$

It can be shown that this power minimization is related to maximization of the ALU utilization.[7] The choice of ALUs in a cluster depends on α, β, and p.

Results

The architecture design of embedded stream processors for meeting an application's real-time requirements greatly depends on the parallelism available in the application. Hence, a stream processor architecture exploration requires a parallelism study of the application. To evaluate our design exploration methodology, we applied it to the design of a 3G wireless base station embedded processor that meets real-time requirements. We considered a 32-user base station providing 128 Kbps per user (coded), employing multiuser channel estimation, multiuser detection, and Viterbi decoding.[3] This is the worst-case workload that the processor design must support.

Ideally, the exploration tool with the help of the compiler should determine the CDP range. The compiler should automatically exploit all the available ILP (using loop unrolling) and SP and set the remaining DP as CDP. Because the compiler lacks the ability to automate this process, we set the CDP manually after exploring different amounts of loop unrolling and observing the changes in ILP. A two-cluster stream processor is not practical because it does not recover the overhead of parallelizing the algorithms. Therefore, we varied the configuration from four clusters to 512 clusters, the maximum CDP available. Similarly, we varied the adders and multipliers up to 5 and 3, respectively, because we have seen ILP saturating above these num-

Table 1. Real-time frequency needed for a wireless base station providing 128 Kbps per user for 32 users.

Algorithm	Kernel	CDP	Cycles	MHz needed
Estimation	Correlation update	32	177	1
	Matrix mul	32	10,822	43
	Iteration	32	261	1
	Transpose	512	95	< 1
	Matrix mul L	32	5,449	22
	Matrix mul C	32	5,577	22
Detection	Matched filter	32	17,822	71
	Interference cancellation	32	20,685	83
Decoding	Packing	256	57	< 1
	Repacking	64	120	< 1
	Initialization	64	4,192	17
	Add-compare-select	64	63,488	254
	Decoding output	64	5,632	23
Minimum real-time frequency f*min* = *f*(5,3,512)				538 MHz
Mathematically required ALU operations				24 GOPS

bers with no performance benefits. The design exploration process confirmed these ranges.

Table 1 breaks down the workload computations for attaining the lowest real-time frequency at compile time using Equation 8. CDP varies in the algorithms between 32 and 512, justifying the range for CDP exploration. Also, more than 99 percent of real-time frequency is needed because of kernels that require 32 and 64 clusters, so there is little advantage in exploring a higher number of clusters. We estimate the minimum real-time frequency $f_{\min}$ needed for the design workload is 538 MHz.

Figure 3 shows the workload's real-time frequency with increasing clusters and varying β. Because the minimum CDP is 32, execution time decreases linearly to 32 and then no longer provides a linear decrease, as Equation 9 shows. Further increasing the clusters above 64 clusters has almost no effect on execution time because the algorithms using the higher CDP take less than 1 percent of the workload time, as Table 1 shows.

Figure 4 shows the variation of normalized power with increasing clusters as clock frequency decreases to achieve real-time execution of the workload. We obtain this result from Equation 10. The thick lines show the ideal, no-stall case of β = 1, and the thin lines show the variation as β decreases to 0. The figure shows that power consumption reduces drastically up to a factor of 100 as the number of clusters reaches 64 from four, because the clock frequency reduction outweighs the capacitance increase due to increased ALUs. After 64 clusters, the capacitance increase outweighs the small performance benefits and increases power consumption. The figure also shows that the design choice for clusters is actually independent of the value of *p* and β,

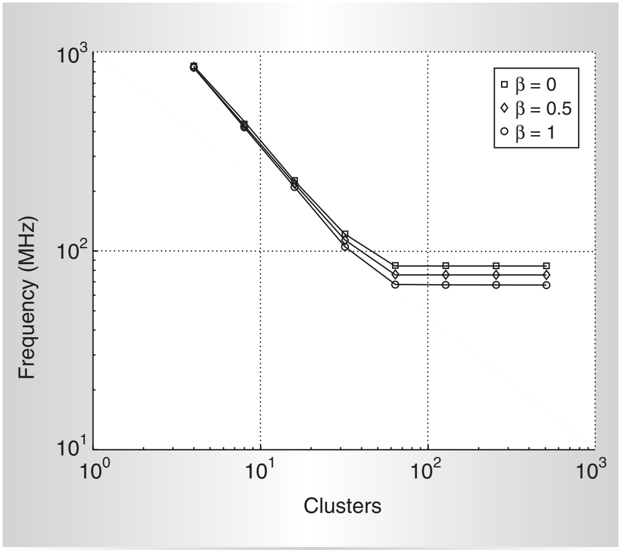

Figure 3. Variation of real-time frequency with increasing clusters.

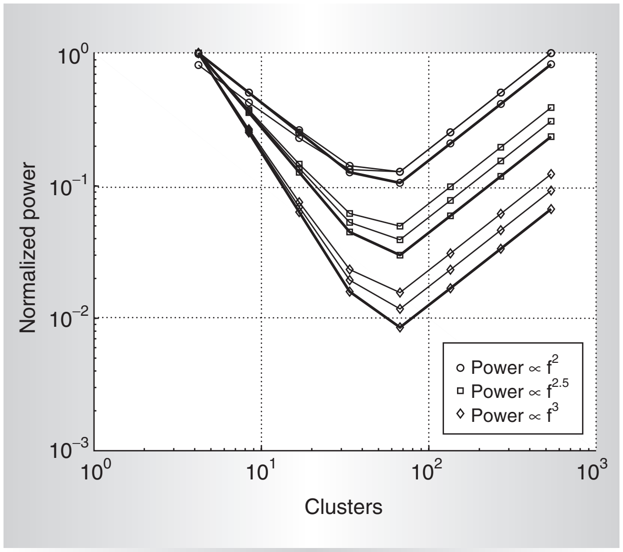

Figure 4. Minimum power point with increasing clusters and variations in p and β. Thick lines show ideal, no-stall case of β = 1. Thin lines show variation with β; (a_{max}, m_{max}) = (5, 3).

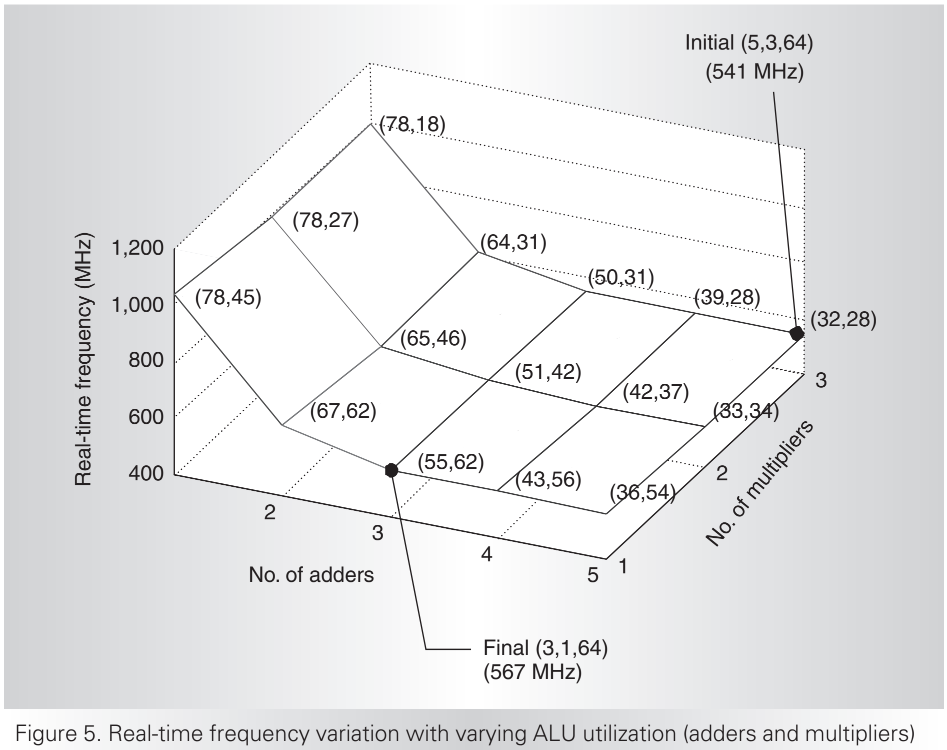

Figure 5. Real-time frequency variation with varying ALU utilization (adders and multipliers) for c = 64, α = 0.01, β = 1, and p = 3, refining the (a, m, c) = (5, 3, 64) solution.

with all variations showing the same design solution: 64 clusters ∀ (p, α, β).

After setting the number of clusters, we perform a similar exploration to choose the number of adders and multipliers within a cluster, minimizing power using Equation 11. Figure 5 shows the variation of real-time frequency with increasing adders and multipliers. We obtained the utilization of the adders and multipliers from a compile time analysis using the design tool. The pairs of numbers represent adders and multipliers utilization respectively. The figure shows that after the (3 adder, 1 multiplier) point, there is very little performance benefit from more adders or multipliers. This is the point where the processor exploits all the ILP. However, the (2 adder, 1 multiplier) point has a higher ALU utilization for the same amount of work. So we would expect one of these configurations to be a low-power solution as well. The actual low-power point depends on the variation of α, β, and p in the design. For the case of α = 0.01, β = 1, and p = 3, configuration (a, m, c) = (3, 1, 64) obtains the minimum power as computed from Equation 11.

Figure 6 shows the power minimization sensitivity of the ALUs in each cluster to p, β, and α. The columns represent variations in p. The first three rows show the design's sensitivity to variations in memory stalls β, and the last row shows its sensitivity to variations in α. We make the following observations: First, two candidate configurations of (a, m, c) emerge after sensitivity analysis: (2, 1, 64) and (3, 1, 64). Second, the design is most sensitive to variations in p. We can see that p = 2 always selects the (2, 1, 64) configuration, and p = 3 always selects the (3, 1, 64) configuration. We expect this, because variations in p affect the power minimization exponentially. In Figure 5, the (3, 1, 64) configuration choice (shown as the final choice) with adder-multiplier utilization of (55, 62) and the adjacent configuration choice of (2, 1, 64) having adder-multiplier utilization (67, 62) have among the highest ALU utilizations. This shows the correlation between ALU utilization maximization in Figure 5 with the power minimization shown in Figure 6. Third, the design is also

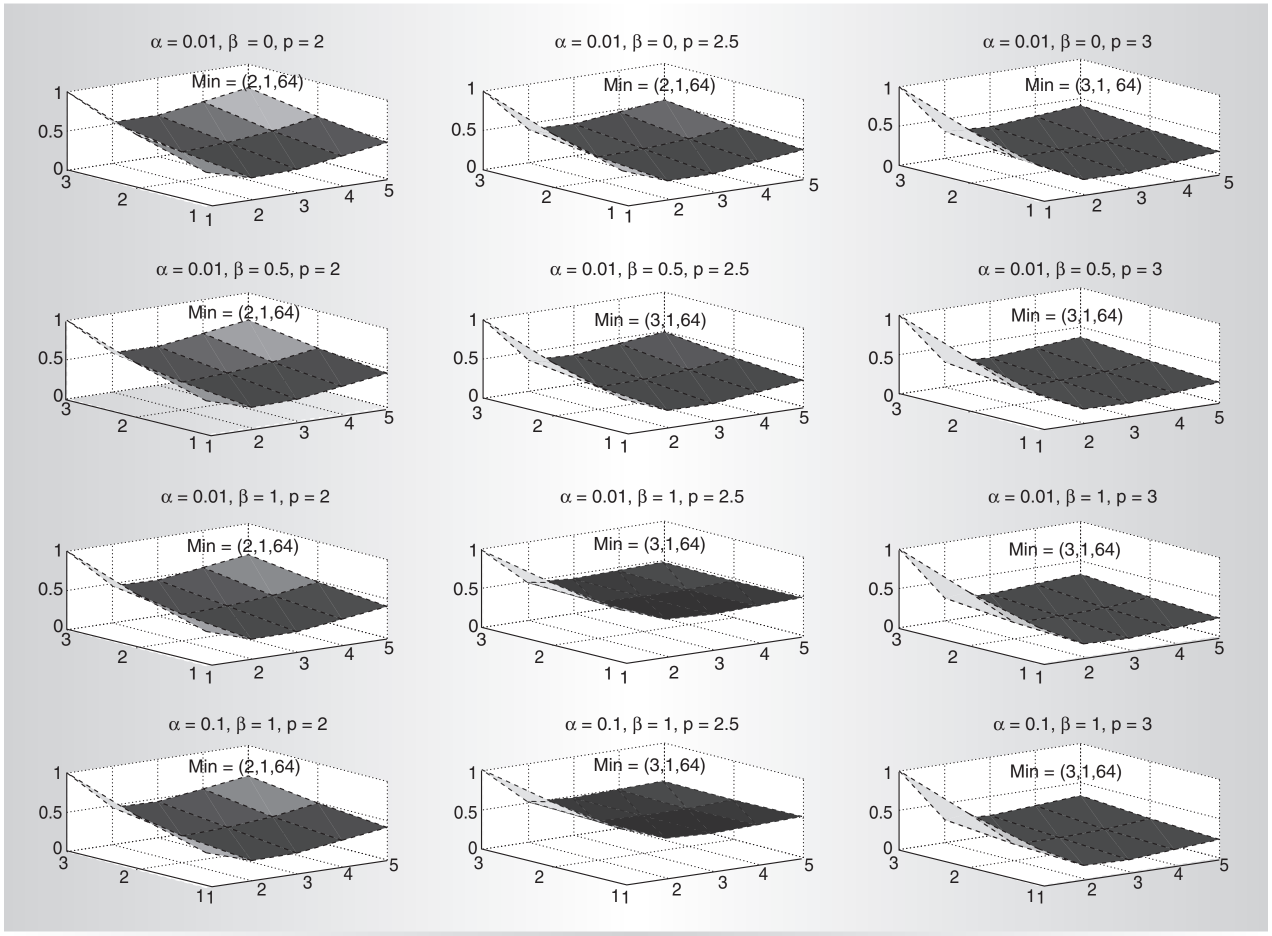

Figure 6. Sensitivity of power minimization to *p*, α, and β for 64 clusters. (*x*-axis: 1 to 5 adders; *y*-axis: 1 to 3 multipliers; *z*-axis: 0 to 1 normalized power).

sensitive to memory stalls β. For β = 0 and *p* = 2.5, the design choice is (2, 1, 64), and it changes to (3, 1, 64) as β increases. Finally, the last row in Figure 6 shows that the design is relatively insensitive to α variations. The reason for this is that the register files and associated intracluster communication network added as ALUs increase dominate power consumption, taking 70 to 79 percent of cluster power for the configurations studied. Cluster power, on the other hand, takes between 57 and 61 percent of total chip power.

Recall that Figure 4 shows that the 64-cluster architecture has a lower power consumption than the 32-cluster architecture. However, a 64-cluster configuration will never attain 100 percent cluster efficiency because clusters 33 to 64 will remain unutilized when the CDP falls below 64. A 64-cluster architecture obtains only a 54 percent cluster utilization for the workload but has a lower power consumption than a 32-cluster architecture with 100 percent cluster utilization, merely because of its ability to lower the clock frequency, which balances out the capacitance increase.

Verifications with detailed simulations

The design exploration tool gave two candidates as output, with variations in *p*, α, and β:

- Design 1: (*a*, *m*, *c*): (∞, ∞, ∞) → (5, 3, 512) → (5, 3, 64) → (2, 1, 64)
- Design 2: (*a*, *m*, *c*): (∞, ∞, ∞) → (5, 3, 512) → (5, 3, 64) → (3, 1, 64)

The design tool starts from a hypothetical infinite machine represented as (∞, ∞, ∞) and successively refines the architecture to provide low-power candidate architectures. Table 2

Table 2. Verification of design tool output (T) with a detailed cycle-accurate simulation (S).

Configuration (*a, m, c*)	**T/S**	**β**	**Computation time (cycles)**	**Micro-controller stalls (cycles)**	**Exposed stalls (cycles)**	**Total time (cycles)**	**Real-time frequency (MHz)**	**Capac. (*a, m, c*)**	**Relative power consumption**		
									***p* = 2**	***p* = 2.5**	***p* = 3**
Design 1	T	0	163,560	33,792		197,532	786	1	1	1	1
(2, 1, 64)	T	0.5	163,560	16,896		180,456	718				
	T	1	163,560	0		163,560	651				
	S		166,174	22,293	34,290	222,757	887				
Design 2	T	0	142,410	33,792		176,382	702	1.21	1.11	1.09	1.05
(3, 1, 64)	T	0.5	142,410	16,896		159,306	634				
	T	1	142,410	0		142,410	567				
	S		147,721	19,660	45,470	212,851	848				
Human	T	0	214,241	33,792		248,213	988	1.18	1.61	1.74	1.85
(3, 3, 32)	T	0.5	214,241	16,896		231,137	920				
	T	1	214,241	0		214,241	853				
	S		223,432	25,940	25,261	258,693	1,030				

shows the results of design verification of the tool output (T) with a cycle-accurate simulation (S), using the Imagine simulator, which produces execution time details including computation time, memory stalls, and microcontroller stalls.

The design tool models the workload's computation part very realistically. We attribute the relatively small errors to the assumption that ILP is independent of CDP and to the prologue and epilogue effects of ignored code loops. It is interesting to see the percentage increase in memory stalls with lower clock frequencies in the cycle-accurate simulations. In a traditional microprocessor system with a constant configuration, the stalls due to cache misses decrease with lower processor clock frequencies because the caches have more time to receive data from external memory. However, in our design exploration, the number of clusters in the stream processor increases to reduce clock frequency while keeping memory bandwidth the same. This implies that the memory now must provide more data to the SRFs at the same bandwidth, thereby increasing the number of stalls and making the architecture memory bound. The actual number of cycles lost to memory stalls was larger than estimated by β. However, even after increasing β to a larger range, we still obtained the same two candidates for evaluation. Both candidates are very close in power consumption, with the (3, 1, 64) configuration only 5 to 11 percent different from the (2, 1, 64) configuration.

We also compared our tool's candidate configurations with a configuration carefully chosen by a human designer.[3] The analysis of the workload kernels was based on the DP and the operation mix. The designer chose a (3, 3, 32) configuration because the algorithms show an equal number of additions and multiplications and a minimum DP of 32. Our design tool provided us with lower-power configurations than the human configuration and improves the design's power efficiency by a factor of 1.61 to 1.85 for the chosen workload.

Although we have focused on wireless base stations, our design exploration technique can obtain low-power solutions for all embedded stream processor designs in media and signal processing. Other embedded-processor design exploration schemes that exploit ILP, SP, and DP can also benefit from the concepts presented here.

The current design exploration tool does not model switching activity and static power dissipation in the simulations. Chip parts such as the stream register file and the microcontroller can be assumed to have fairly constant switching activity. Because the tool chooses ALUs that have a high functional unit utilization, switching activity in the ALUs can be approximated as constant as well. Static power dissipation is directly proportional to the number of transistors and hence to capacitance.[12] Static power dissipation is also a function of transistor leakage effects and varies with technology. The model can be extended by adding a static power consumption factor to the optimization cost function.[12] The clock frequency design point is decided by the application's real-time requirements, not by physical limitations of adders, multipliers, and clusters. If the design tool's clock frequency output exceeds stream processor limits, we must modify the design exploration to search for other feasible configurations, although they may not yield the minimum power solution. This modification is unlikely to yield lower clock frequency configurations because power minimization strives to provide a low clock frequency.

Other parameters can affect embedded stream processor performance and need exploration, such as the number of registers and the ALU pipelining depth.[2] These parameters affect a cluster's ILP and thus indirectly affect CDP. Although an exploration of these parameters will affect the actual design choice, the design exploration methodology does not change as we decouple ILP and DP. We focus on the exploration of (a, m, c, f) for power minimization and their sensitivity to the three parameters (α, β, p). Once we have determined (a, m, c, f), we can set other parameters on the basis of this configuration. With improvements in compilers for embedded stream processors, we can improve the design exploration tool heuristic by incorporating techniques such as integer-linear programming for jointly exploring (a, m, c, f), as well as other processor parameters such as register file sizes and ALU pipeline depths. Once a worst-case design is complete, we can use a multiplexer network between the internal memory and the clusters to adapt the clusters with runtime workload variations for further improvements in power efficiency.[7] MICRO

Acknowledgments

Sridhar Rajagopal and Joseph Cavallaro were supported in part by Nokia, Texas Instruments, and National Science Foundation grants EIA-0224458 and EIA-0321266.

References

1. U.J. Kapasi et al., "Programmable Stream Processors," *Computer*, vol. 36, no. 8, Aug. 2003, pp. 54-62.
2. D. Marculescu and A. Iyer, "Application-Driven Processor Design Exploration for Power-Performance Trade-Off Analysis," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design* (ICCAD 01), IEEE Press, 2001, pp. 306-313.
3. S. Rajagopal, S. Rixner, and J.R. Cavallaro, "A Programmable Baseband Processor Design for Software Defined Radios," *Proc. 44th IEEE Int'l Midwest Symp. Circuits and Systems* (MWSCAS 02), vol. 3, IEEE Press, 2002, pp. 413-416.
4. R. Leupers, "Instruction Scheduling for Clustered VLIW DSPs," *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques* (PACT 00), 2000, IEEE Press, pp. 291-300.
5. B. Khailany et al., "Exploring the VLSI Scalability of Stream Processors," *Proc. 9th Int'l Conf. High-Performance Computer Architecture* (HPCA 03), IEEE Press, 2003, pp. 153-164.
6. T. Gemmeke, M. Gansen, and T.G. Noll, "Implementation of Scalable and Area-Efficient High-Throughput Viterbi Decoders," *IEEE J. Solid-State Circuits*, vol. 37, no. 7, July 2002, pp. 941-948.
7. S. Rajagopal, *Data-Parallel Digital Signal Processors: Algorithm Mapping, Architecture Scaling and Workload Adaptation*, doctoral dissertation, Dept. of Electrical and Computer Eng., Rice Univ., 2004.
8. L.T. Clark et al., "An Embedded 32-bit Microprocessor Core for Low-Power and High-Performance Application," *IEEE J. Solid State Circuits*, vol. 36, no. 11, Nov. 2001, pp. 1,599-1,608.
9. A. Bogliolo et al., "Parameterized RTL Power Models for Combinational Soft Macros," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design* (ICCAD 99), IEEE Press, 1999, pp. 284-288.
10. A. Beaumont-Smith et al., "GaAs Multiplier and Adder Designs for High-Speed DSP Applications," *Proc. 31st Asilomar Conf. Signals, Systems & Computers*, vol. 2, IEEE Press, 1997, pp. 1517-1521.
11. J.D. Owens et al., "Media Processing Applications on the Imagine Stream Processor," *Proc. IEEE Int'l Conf. Computer Design* (ICCD 02), IEEE Press, 2002, pp. 295-302.
12. J.A. Butts and G.S. Sohi, "A Static Power Model for Architects," *Proc. 33rd Ann. Int'l Symp. Microarchitecture* (Micro-33), IEEE CS Press, 2000, pp. 191-201.

Sridhar Rajagopal is a member of the technical staff at WiQuest Communications. His research interests include wireless communication systems, computer architecture, computer arithmetic, and parallel computing. Rajagopal has PhD and MS degrees in electrical and computer engineering from Rice University, where he participated in the work described in this article, and a BE in electronics engineering from Mumbai University, India. He is a member of the IEEE.

Joseph R. Cavallaro is a professor of electrical and computer engineering at Rice University and associate director of the university's Center for Multimedia Communication. His research interests include computer arithmetic, VLSI and FPGA design, and architectures and algorithms for wireless communication systems. Cavallaro has a BS from the University of Pennsylvania, an MS from Princeton University, and a PhD from Cornell University, all in electrical engineering. He is a member of the IEEE.

Scott Rixner is an assistant professor of computer science and electrical and computer engineering at Rice University. His research interests include media, network, and communications processing; memory system architecture; and the interaction between operating systems and computer architectures. Rixner has BS and MEng degrees in computer science and engineering and a PhD in electrical engineering, all from the Massachusetts Institute of Technology. He is a member of the ACM.

Direct questions and comments about this article to Sridhar Rajagopal, WiQuest Communications, PO Box 2326, Allen, TX 75013; sridhar.rajagopal@wiquest.com.