Improving Power Efficiency in Stream Processors Through Dynamic Cluster Reconfiguration

Sridhar Rajagopal
WiQuest Communications
Allen, TX 75002
sridhar.rajagopal@wiquest.com

Scott Rixner and Joseph R. Cavallaro
Rice University
Houston, TX 77005
rixner,cavallar@rice.edu

ABSTRACT

Stream processors support hundreds of functional units in a programmable architecture by clustering functional units and utilizing a bandwidth hierarchy. Clusters are the dominant source of power consumption in stream processors. When the data parallelism falls below the number of clusters, unutilized clusters can be turned off to save power. This paper improves power efficiency in stream processors by dynamically reconfiguring the number of clusters in a stream processor to match the time varying data parallelism of an application. We explore 3 mechanisms for dynamic reconfiguration: using memory, conditional streams and a multiplexer network. A 32-user wireless basestation is a prime example of a workload that benefits from such reconfiguration. When the number of users supported by the basestation dynamically changes from 32 to 4, the reconfiguration from a 32-cluster stream processor to a 4-cluster stream processor yields 15–85% power savings over and above a stream processor that uses conventional power saving techniques such as dynamic voltage and frequency scaling. The dynamic reconfiguration support extends stream processors from traditional high performance applications to power-sensitive applications in which the data parallelism varies dynamically and falls below the number of clusters.

1. INTRODUCTION

Rapidly evolving standards and time-varying workloads motivate the need for programmable solutions for media, signal processing and wireless applications. These applications require high computation rates, real-time guarantees, and low power consumption. Stream processors enable such applications by utilizing a bandwidth hierarchy to keep tens to hundreds of clustered functional units busy in a power-efficient manner [1]. A stream processor used in applications such as wireless base-station must be provisioned to handle the worst-case load in order to provide the necessary real-time guarantees. However, such applications rarely operate at full capacity. When the base-station is supporting fewer users or lower data rates, a significant fraction of the stream processor resources are underutilized. These underutilized resources lead to significant wasted power under these lightly loaded conditions as clusters are the dominant source of power consumption. Therefore, techniques to turn off clusters dynamically in a stream processor increase the power-efficiency of stream processors for time-varying workloads.

When the load on the wireless base-station changes, so does the amount of data parallelism within the application. A stream processor interleaves data-parallel computations across clusters of arithmetic units. In order to turn off underutilized clusters, data streams must be reorganized to map only to active clusters. A stream processor can dynamically adapt to the varying levels of parallelism within the workload using three different approaches: memory transfers, conditional streams, and an adaptive multiplexer network. Memory transfers and conditional streams are existing mechanisms in stream processors that are being used in a novel way for reconfiguration. In addition, we introduce the multiplexer network as an additional efficient mechanism for reconfiguration.

Using these techniques, such a dynamically reconfigurable stream processor can adjust the frequency, voltage, and arithmetic resources, significantly reducing power dissipation under lightly loaded conditions. For example, a full capacity base-station (32 users with strong Viterbi decoding) using a stream processor would need to operate at 1.2 GHz and would dissipate approximately 12.4 W. However, with dynamic reconfiguration using an adaptive multiplexer network, the same base-station under lightly-loaded conditions (4 users with weaker Viterbi decoding) would only need to operate at 433 MHz and would dissipate only 300 mW. This is a 41.3$\times$ reduction in power consumption that could not be provided by a conventional stream processor.

The next section gives an introduction to the algorithms and processing requirements of the physical layer in base-stations. This is followed by an overview of stream processors in section 3. Section 4 shows the different mechanisms for reconfiguration in stream processors, their tradeoffs and the software support needed in stream processors to provide reconfiguration. Section 5 provides comparisons between the different reconfiguration methods in terms of performance and section 6 gives insights into the power savings made feasible due to reconfiguration.

2. BASE-STATION PROCESSING

Wireless cellular systems are evolving from low data rate voice-based (2G) systems to high data rate multimedia-based (3G) systems. Figure 1 shows the increase in the mathematically required operation count of the physical layer from millions of operations per second (MOPs) to billions of operations per second (GOPs) as a 32-user base-station goes from a second generation (2G) voice only system supporting 16 Kbps/user (coded data rate), to a third generation (3G) multimedia-oriented system supporting 128 Kbps/user. Note that any actual base-station processor will likely execute ad-

MSP-6 Workshop at Micro-37, December 2004.

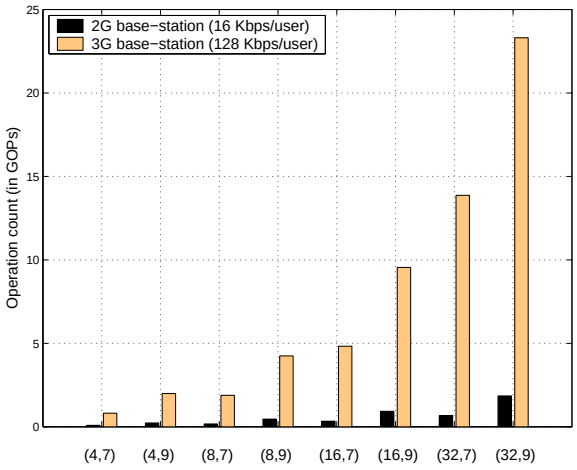

Figure 1: Operation count for 2G and 3G base-stations. The configurations such as (4,7) shown in the X-axis represent the workload variation with the number of users and constraint length (Users,Constraint lengths)

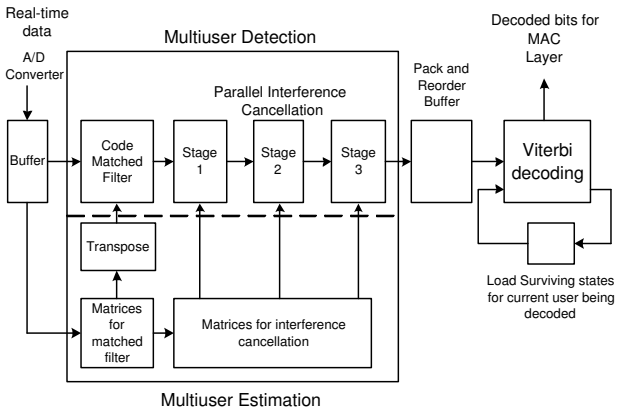

Figure 2: Detailed view of physical layer processing at the base-station

ditional operations beyond those strictly required. Regardless, the two to three orders-of-magnitude increase in complexity is due to two main reasons. First, the increase in data rate directly increases the real-time processing requirements at the wireless receiver. Second, there is an increase in the computational complexity of the physical layer algorithms in the receiver. Advanced algorithms with sophisticated signal processing are used in 3G systems to provide lower bit error rates. This results in better reception and makes more efficient use of the expensive wireless spectrum which enables higher capacity systems (higher bits/sec/Hz).

Figure 2 shows the receiver architecture of a 32-user base-station that supports 128 Kbps per user (coded data rate). In order to achieve high spectral efficiency, advanced signal processing algorithms such as multiuser channel estimation, multiuser detection, and Viterbi decoding are being proposed for base-stations [2]. Multiuser channel estimation is the process of jointly determining the parameters of the wireless channel for all users simultaneously. This estimate is then used to jointly detect the data for all users in the multiuser detection phase. Finally, Viterbi decoding is used to detect and correct errors, using the error control coding performed by the transmitter.

Physical layer wireless algorithms demonstrate significant data parallelism that benefit from SIMD processing. The computations for channel estimation involve matrix-matrix multiplications of the order of users (U) and the spreading gain (N), while detection computations involve matrix-vector multiplications of the order of the number of active users (U) in the base-station. Viterbi decoding typically involves a trellis [3] with two phases of computation: an add-compare-select phase and a traceback phase. However, the traceback phase is inherently sequential and involves serialized pointer chasing. With large constraint lengths, the computational complexity of Viterbi decoding increases exponentially and becomes the critical bottleneck, especially as multiple users need to be decoded simultaneously in real-time. The complexity of decoding is proportional to the number of users (U), the constraint length (K) and the coding rate (R). However, a data-parallel register-exchange based scheme can be used during the normally sequential traceback phase [3]. The register-exchange method of Viterbi traceback is especially well-suited to parallel processing. As base-stations begin to provide higher data rates to individual users, flexibility becomes important in addition to computational capacity and power efficiency. New emerging wireless standards require greater flexibility than past standards, such as supporting different data rates, varying number of users, various decoding constraint lengths and rates, adaptive modulation and spreading schemes. Programmability in the base-station provides this flexibility and makes the task of upgrading base-stations with emerging standards easier. As wireless standards evolve, programmable base-station designs will also allow limited versions of future standards to be implemented immediately without design time loss for new architectures.

Furthermore, power-efficiency is important, as it helps decrease operational and maintenance costs. Increased power efficiency also allows smaller and lower cost battery backup systems, enabling continued, uninterrupted service during critical times of power failures such as the recent August 2003 blackout [4]. Computational capacity and power efficiency are becoming critical to base-station design as they become denser (pico-cells) and more prevalent in high-traffic areas.

3. STREAM PROCESSORS

Traditional superscalar and VLIW DSP architectures cannot scale to support hundreds of arithmetic units needed to provide billions of operations per second computational requirements for wireless base-station processing because the register file size and port interconnections rapidly begin to dominate the area, delay, and power of such architectures [5]. On the other hand, use of multiple DSP processors create load-balancing, power and cost issues due to chip-to-chip communication in addition to making the programming of the multi-processor complex. As the number of processors increase, the efficiency of such a multi-processor system decreases drastically. Even a voice-oriented 2G base-station design typically requires more than 18 DSPs for physical layer processing [6]. Extending such an architecture to a 3G base-station may require greater than a DSP per user [7]. Hence, co-processor designs are currently being proposed to solve this problem. For example, the recently announced 3G TCI1× UMTS chipset from Texas Instruments for cellular base-stations contains application-specific hardware for operations such as chip de-spreading and Viterbi and Turbo decoding [8]. Less than 10% of the actual physical layer processing is being done on fully programmable DSPs, which limits the software reuse of the architecture.

Special-purpose processors for wireless communications perform

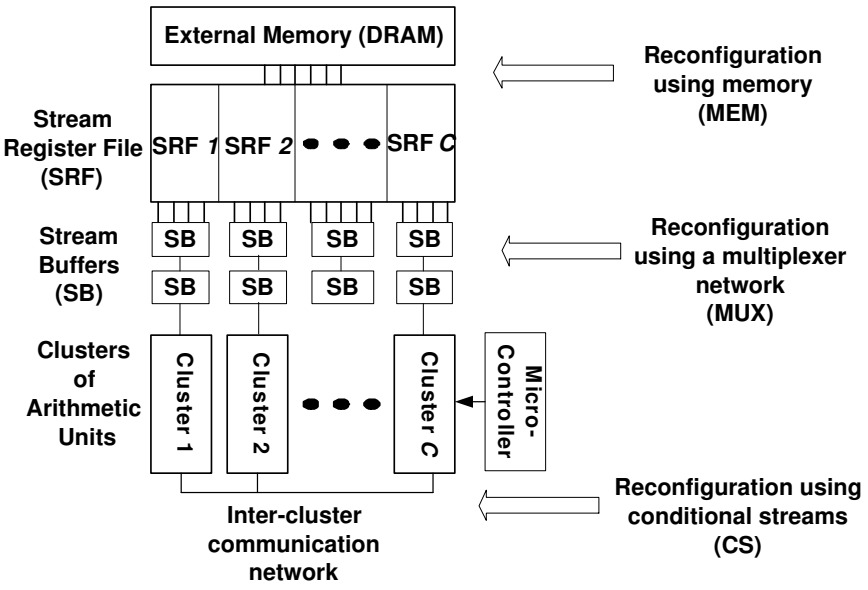

Figure 3: Opportunities for reconfiguration in stream processors

well because of the abundant parallelism and regular communication patterns within physical layer processing. These processors efficiently exploit these characteristics to keep thousands of arithmetic units busy with minimal expensive global communication and storage resources. The bulk of the parallelism in wireless physical layer processing can be exploited as data parallelism, as identical operations are performed repeatedly on incoming data elements.

A stream processor is designed to efficiently exploit this kind of data parallelism. Figure 3 shows the architecture of a stream processor with C arithmetic clusters. A stream processor is programmed at two levels: the stream level and the kernel level. Stream operations all consume and/or produce streams which 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. 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 VLIW cluster is a set of arithmetic units that performs the same sequence of operations as the other clusters on independent stream elements. The stream buffers (SBs) allow the single port into the SRF array (limited for area/power/delay reasons) to be time-multiplexed among all the interfaces to the SRF, making it appear that there are many logical ports into the array. The stream buffers (SBs) also act as prefetch buffers and prefetch the data for kernel operations. Both the SRF and stream buffers are banked to match the number of clusters. Such a stream processor can exploit data parallelism in three ways: using parallel-subword operations, using loop unrolling and software pipelining, and using SIMD cluster operations. For a given kernel, parallel-subword operations should be used first to exploit fine-grained data parallelism. This allows the bandwidth and arithmetic resources to be used efficiently, rather than leaving fractions of each data word empty. After data is packed into subwords, software techniques should be used to expose additional parallelism in order to keep the arithmetic units within each cluster occupied. A combination of loop unrolling and software pipelining are especially effective at converting data parallelism into instruction-level parallelism for such VLIW clusters. Resource limitations constrain the amount of parallelism that can be exposed and exploited in this manner. Finally, the remaining parallelism can be exploited by interleaving stream elements across arithmetic clusters. The arithmetic clusters efficiently exploit this remaining parallelism using a single microcontroller for the SIMD clusters and a set of stream buffers to deliver high bandwidth to and from the SRF. Additional details about stream processor architecture and the stream programming model can be found in [1].

If a particular computation kernel does not have enough data parallelism left over after packing data into subwords and using software techniques to expose parallelism, then the arithmetic clusters will be underutilized. Since the SRF interleaves stream elements across all of the clusters, short streams will not feed enough elements to each cluster in order to allow sufficient loop unrolling and software pipelining to keep the arithmetic units within each cluster busy. In this case, the structures within a stream processor that *normally* exploit large amounts of parallelism in a power-efficient manner will consume power while performing very little computation.

4. DYNAMIC CLUSTER RECONFIGURATION IN STREAM PROCESSORS

Figure 3 shows the architecture of a stream processor with C arithmetic clusters. Applications, such as wireless base-stations, have varying levels of data-parallelism within a particular system configuration. If the smallest data-parallelism configuration in the entire application is used as the choice for the number of clusters, the processor design fails to meet real-time requirements. On the other hand, a processor designed to handle the maximum data-parallelism configuration becomes under-utilized when running workloads with lower data parallelism than the number of clusters. The problem with adapting the number of clusters with the data-parallelism arises from the fact that the data is striped across all clusters and needs to be re-routed to the appropriate clusters before it can be

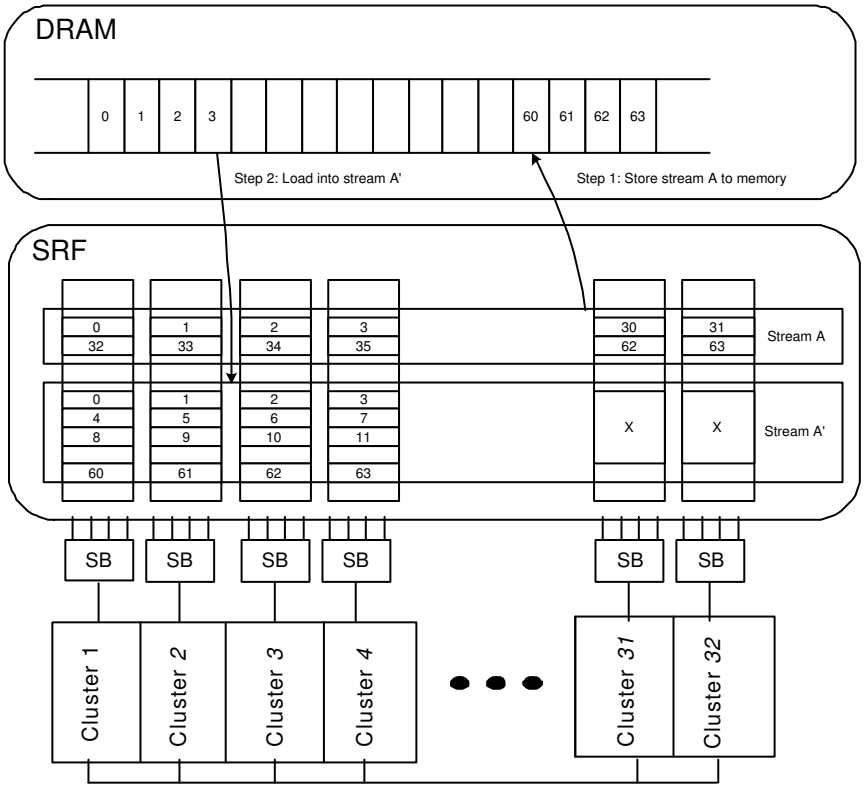

Figure 4: Reorganizing Streams in Memory

consumed. In [9], we have explored choices for the combination of clusters and ALUs within a cluster for power minimization during design time. However, further power savings are possible using reconfiguration and turning off clusters dynamically during run-time of an application. We propose 3 methods to dynamically adapt the resources of a stream processor to match the workload and route the data to the appropriate clusters, improving the power efficiency of stream processors. As can be seen from Figure 3, three different mechanisms could be used to reroute the stream data to the appropriate clusters: by using the memory system, by using conditional streams, and by using a multiplexer network. Memory transfers and conditional streams are existing mechanisms in stream processors that are being used in a novel way for reconfiguration. In addition, we introduce the multiplexer network as an additional efficient mechanism for reconfiguration. Tradeoffs exist between these different mechanisms and the choice of reconfiguration depends on the amount and type of variation of data parallelism in the application and the frequency of reconfiguration needed. Combinations of these different methods are also possible for reconfiguration.

In order to demonstrate the benefits of each of the reconfiguration schemes and the tradeoff between them, we consider a simple case of an application that has a 32-cluster stream processor and has a workload that requires changes in the data parallelism from 32 to 4 during execution such as the (4,9) configuration.

4.1 Reconfiguration in memory

Reconfiguration using memory transfers uses the interface between the SRF and external DRAM to reorganize data to match the number of active clusters. A data stream can be realigned to match the number of active clusters by first transferring the stream from the SRF to external memory, and then reloading the stream so that it only is placed in SRF banks that correspond to active clusters. Figure 4 shows how this would be accomplished on a 32 cluster stream processor. In this example, a 64 element data stream, labelled Stream A in the figure, is produced by a kernel running on all 32 clusters. Therefore, the stream is striped across all 32 banks of the SRF. It is important to note here that it is possible for a stream of length 64 to have a data parallelism of 4. An example of this would be a matrix of size 4×16, which would occupy 2 elements per SRF in a 32-cluster machine. If the machine is then reconfigured to only have four active clusters to match the data parallelism, the stream needs to be striped across only the first four banks of the SRF. By first performing a stream store instruction, the stream can be stored contiguously in memory, as shown in the figure. Then, a stream load instruction can be used to transfer the stream back into the SRF, only using the first four banks. The figure shows Stream A$'$ as the result of this load. As can be seen in the figure, the remaining banks of the SRF would not contain any valid data for Stream A$'$. (In actual hardware, the writes to banks 5-32 would be disabled) This mechanism for reconfiguration is beneficial for applications that need to reconfigure infrequently and/or require small amounts of data to be reorganized during reconfiguration. This is because the reconfiguration overhead can get easily amortized over the entire workload in such cases. Once the clusters are reconfigured, unused clusters as well as unused SRF banks can be turned off to conserve power. However, this mechanism for reconfiguration suffers from several limitations as well. First, the memory access stride needed to transfer the data from 32 clusters to 4 clusters is not uniform. The non-uniform access stride makes the data reorganization difficult and increases memory stalls. Next, the reconfiguration causes memory bank conflicts during the transfer as multiple reads (during reconfiguration to higher number

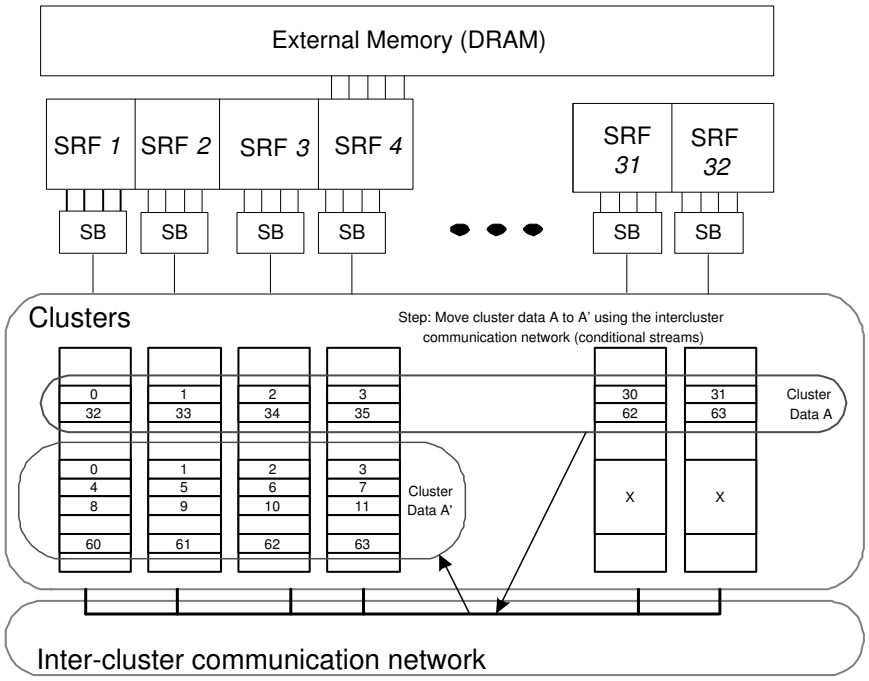

Figure 5: Reorganizing Streams with Conditional Streams

of clusters) or writes (during reconfiguration to lower number of clusters) are needed from the same bank. Also, one of the motivations for stream processor design is to keep the arithmetic units busy so that data transfer between memory and the processor core is minimized. Forcing all data to go through memory whenever the number of active clusters is changed directly violates this premise. The scheme also requires a larger SRF since the data which was spread across all 32 SRF banks now needs to fit in 4 SRF banks. Memory operations are also expensive in terms of power consumption.

4.2 Reconfiguration using conditional streams

The second option for reconfiguration uses the inter-cluster communication network between the clusters for re-routing data to appropriate clusters. Stream processors already contain a mechanism for reorganizing data streams using conditional streams [10]. Conditional streams allow data streams to be compressed or expanded in the clusters so that the direct mapping between SRF banks and clusters can be violated. When using conditional streams, stream input and output operations are predicated in each cluster. In a stream processor, conditional input streams are implemented by having each cluster read data from the stream buffer as normal, but instead of directly using that data, it is first buffered. Then, based upon the predicates, the buffered data is sent to the appropriate clusters using an intercluster switch. If the predicate is true (1) in a particular cluster then it will receive the next stream element, and if the predicate is false (0), that cluster will not receive any data. Conditional streams are used for three main purposes: switching, combining, and load balancing. Using conditional streams to deactivate parts of a cluster is a novel use of conditional streams.

While reconfiguring from 32 clusters to 4 clusters, as shown in Figure 5, the first 4 clusters are predicated to 1 while the remaining 28 clusters are predicated to 0. Thus, the data is re-routed from all clusters into the first 4 clusters using the inter-cluster communication network. The ALUs in the 28 clusters are idle and can be turned off to save power.

The benefits of reconfiguring using conditional streams are in applications that have non-structured data-parallelism or require non-contiguous SRF accesses. Thus, it has the greatest amount of flexibility among the different methods of reconfiguration. However, when using conditional streams, inactive clusters are not truly inactive. They must still buffer and communicate data. The buffers and the inter-cluster communication network are a significant source of power consumption in the clusters that cannot be saved when reconfiguring using conditional streams. Furthermore, if conditional streams are already being used to switch, combine, or load balance data, then the predicates must be modified to also account for active and inactive clusters, which complicates programming.

4.3 Reconfiguration using a MUX network

Reconfiguration using a multiplexer network uses the interconnection between the SRF and the clusters to re-route data to the appropriate clusters. We modify the interconnection between the SRF and clusters using a multiplexer network that allows reconfiguration to the number of active clusters. The addition of a multiplexer network mitigates the drawbacks of the previous two schemes, enabling more efficient reconfiguration than existing stream processors that will need to use reconfiguration using memory or conditional streams.

From Figure 3, we can observe that the input to the clusters arrive directly from the stream buffers, which are banked to match the number of clusters. Thus, if the data parallelism decreases below the number of clusters, the data for the stream will lie in the wrong bank and hence, cannot be accessed directly by the corresponding cluster. Hence, an interconnection network between the stream buffers and the clusters is an alternative solution to adapt the clusters to the data parallelism.

A fully connected network allowing data to go to any cluster would be extremely expensive in terms of area, power and latency. In fact, stream processors already have a inter-cluster communi-

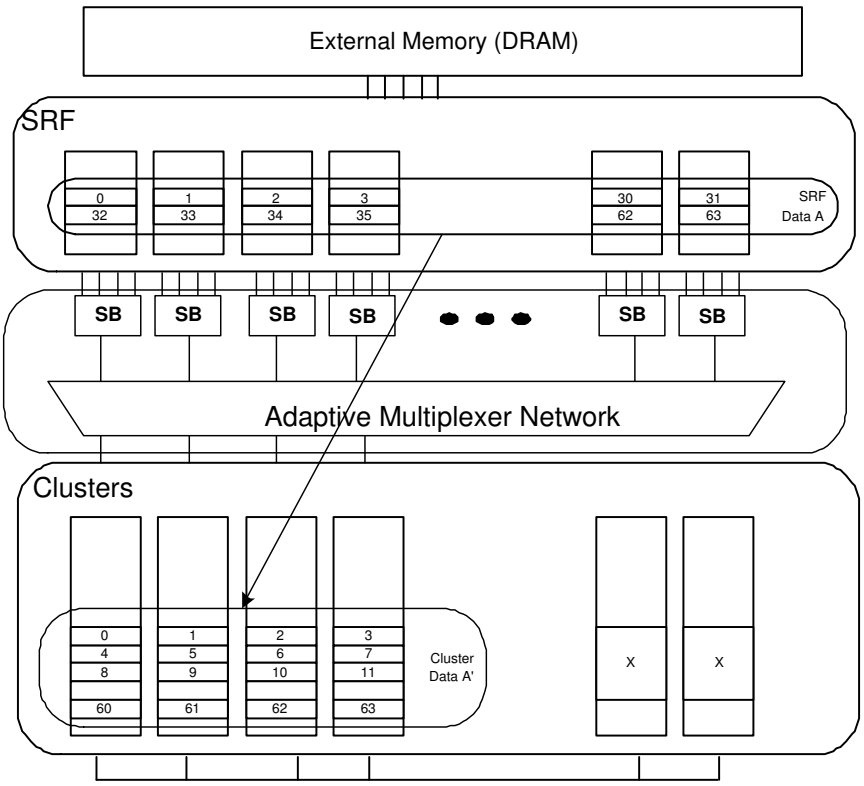

Figure 6: Reconfiguration using a multiplexer network

cation network that can be used to route data to any cluster. The intercluster communication network is not only a significant part of cluster power dissipation, but has large latency and area requirements as well. To investigate networks more suited for reconfiguration, we make use of the fact that it is not necessary to arbitrarily turn off any cluster since all clusters are identical. Hence, we only turn off only those clusters whose cluster identification number is greater than the data parallelism in the algorithms. We further simplify the interconnection network by making the clusters turn off only in powers of two, since most parallel algorithm workloads generally work on datasets in powers of two.

Figure 6 shows the reconfiguration from 32 clusters to 4 clusters using the multiplexer (MUX) network. The data from 32 SRF banks is multiplexed on 4 clusters, making the clusters think that all the data were coming from just 4 SRF banks in a 4-cluster machine. Figure 7 shows the multiplexer network in detail. The multiplexer network allows the ability to turn the clusters on and off, depending on the available data parallelism using an interconnection network within the stream buffers. This interconnection network allows the stream buffers to become adaptive to the workload. In the absence of any reconfiguration, the interconnection network acts as an extended stream buffer, providing the ability to prefetch more data.

Figure 7 shows how the multiplexer network would operate if reconfiguration is needed. The switches in the mux/demux network are set to control the flow of the data to the clusters. There are $\log_2(C)$ stages required in the stream buffer if complete reconfiguration to a single cluster is desired, where C is the number of clusters. Each stage of the multiplexer network holds the data until it is completely read into the clusters. This is done using counters for each stream buffer stage (not shown for clarity). The multiplexer/demultiplexer network shown in Figure 7 is per data stream. The network is configured as a multiplexer for each input stream and as a demultiplexer for each output stream during execution of a kernel.

Reconfiguration using the multiplexer network offers the following advantages. The reconfiguration network allows the stream processor to stripe and access data across all the SRF banks and provides high memory efficiency. The MEM and CS methods require explicit data copying to the appropriate memory banks/clusters before the data can be consumed by the clusters. The multiplexer network also provide higher SRF bandwidth to applications with insufficient data parallelism since all the SRF banks can be accessed and utilized even in cases with insufficient data parallelism. The multiplexer network adds a latency of $\log_2(C)$ to the data entering the clusters. However, the multiplexer network can prefetch data even during cluster stalls, allowing the processor to have the potential to hide latencies or even improve performance when clusters have significant stalls. The cluster reconfiguration is done dynamically by providing support in the instruction set of the host processor for setting the number of active clusters during the execution of a kernel. By default, all the clusters are turned off when kernels are inactive, thereby providing power savings during memory stalls and system initialization as well.

4.4 Software support for reconfiguration

We use a 32 cluster version of the Imagine stream processor [1] with 3 adders and 3 multipliers per cluster as the base stream processor for investigation. The stream processor is programmed in StreamC and KernelC, with a programming syntax similar to C/C++. The kernel code for computations that occur within the clusters is written in KernelC and the communication code for the streams between memory/SRF and the clusters is written in StreamC.

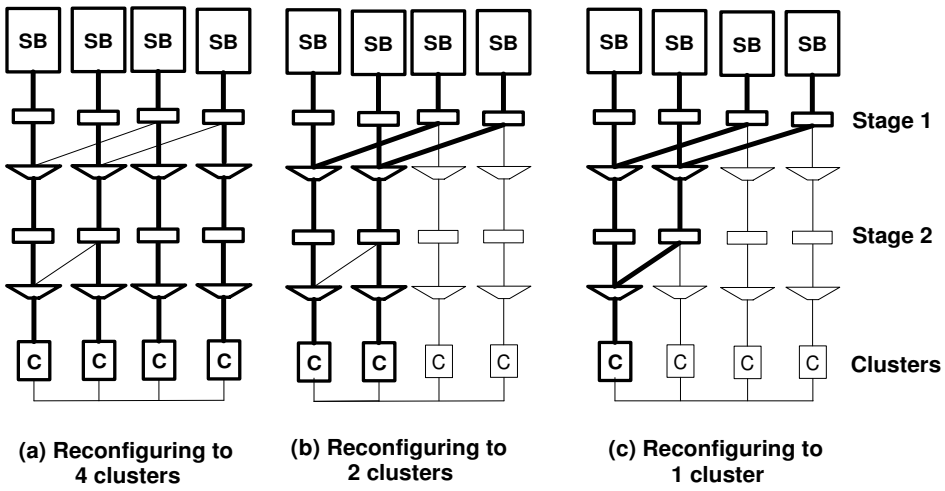

Figure 7: Details of the multiplexer network used in Figure 6 for a 4 cluster stream processor

To dynamically reconfigure using memory, the data is moved to the appropriate memory banks using StreamC. This data movement is similar to data movement in memory in DSPs using DMA transfers. Reconfiguration using conditional streams occurs inside the clusters using KernelC. The cluster identification number and the predicates in each cluster are used to select the clusters where the data should be moved to.

While support for reconfiguration using memory and conditional streams is built into the simulator, the simulator is enhanced with StreamC instructions to set the number of clusters in the processor, which in turn controls the switches in the multiplexer network. Thus, the flow of data between the SRF and the clusters is gated, depending on the KernelC instructions to set the number of clusters.

All applications are written with precompiled kernels that are written in a cluster-independent manner and support varying number of clusters as an input parameter. The points at which algorithms require dynamic reconfiguration are known statically and so, instructions to set the number of clusters are inserted at those points to enable reconfiguration. At run-time, these instructions reconfigure the clusters appropriately based on the information from the host processor about the data parallelism in the current execution of the program.

Table 1: Available data parallelism in wireless communication workloads (U = Users, K = constraint length, N = spreading gain (fixed at 32), R = coding rate (fixed at rate 1/2)). The numbers in columns 3-5 represent the amount of data parallelism. A 32-cluster processor will require reconfiguration for all cases where the data parallelism drops below 32. The fully loaded case does not require any reconfiguration.

Load	Workload (U,K)	Estimation f(U,N)	Detection f(U,N)	Decoding f(U,K,R)
Light	(4,7)	32	4	16
↓	(4,9)	32	4	64
↓	(8,7)	32	8	16
↓	(8,9)	32	8	64
↓	(16,7)	32	16	16
↓	(16,9)	32	16	64
↓	(32,7)	32	32	16
Full	(32,9)	32	32	64

5. RESULTS

Table 1 shows the available data parallelism [1] for the base-station with variation in the number of users (U) and the decoding constraint length (K). From the table, we can observe that the available data parallelism reduces as we go from the full capacity base-station case of 32 users, constraint length 9 (32,9) to lower capacity systems. Based on the data parallelism, if we choose a 32-cluster architecture as the worst case architecture for evaluation, we see that as we go down from case (32,9) to other cases, none of the other workloads meet the minimum data parallelism required to keep all the clusters busy. Hence, there needs to be reconfiguration support provided in stream processors to turn off unused clusters and allow clusters to dynamically match the available data parallelism in the workload.

5.1 Reconfiguration comparisons

[1]The data parallelism in this context is defined as the data parallelism available after packing and loop unrolling

There are three different methods of reconfiguration proposed in this paper, namely (a) Reconfiguration in memory (MEM), (b) Reconfiguration using conditional streams (CS), and (c) Reconfiguration using a multiplexer network (MUX) . As explained in Section 4, each of these reconfiguration mechanisms has trade-offs that are dependent on the amount of data parallelism in the application, the frequency of reconfiguration, the access patterns in the data and the amount of data that needs to be transferred to the active clusters. Combinations of these schemes are also possible.

Figure 8 quantifies the amount of execution time increase due to the processor hardware and software modifications for supporting reconfiguration for wireless base-station processing. In order to compare the increase in execution time, a 32-user system with constraint length 9 Viterbi decoding is considered. The (32,9) case does not require any reconfiguration as it always has data parallelism greater than 32 as can be seen from Table 1. Hence, Figure 8 allows us to see the overhead of providing the ability to reconfigure using memory transfers (MEM), the conditional streams (CS) and multiplexer network (MUX) over the base stream processor that

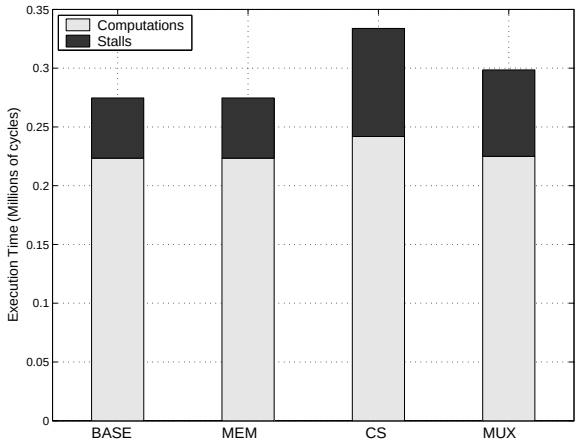

Figure 8: Comparing execution time overhead of the three types of reconfiguration methods for the 32-user, constraint-length 9 case with 32-clusters (no reconfiguration).

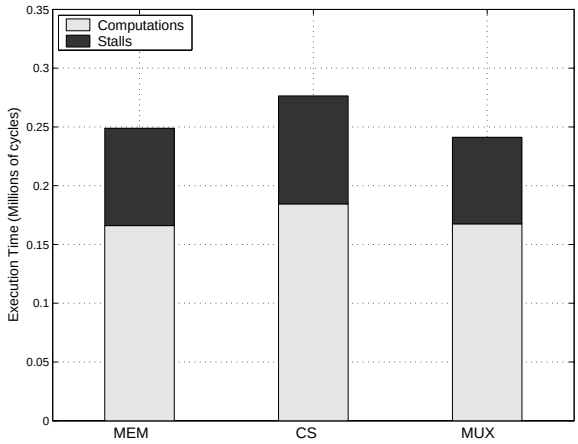

Figure 9: Impact of reconfiguration using memory transfers (MEM), multiplexer network (MUX) and conditional streams (CS) on execution time for the (32,7) case. BASE not shown as it does not support reconfiguration to lower clusters.

does not support reconfiguration (BASE). Since there is no reconfiguration needed in the system, MEM behaves the same as BASE as it does not require any change in the code. CS and MUX, on the other hand, have additional hardware and software changes that are not utilized when there is no reconfiguration. The data flows through the MUX network in spite of not having to reconfigure and this adds latency to the data reaching the clusters. Similarly, conditional streams move all data to the buffers inside the clusters and check the predicate before the data can be used, adding additional latency to the data consumption by the clusters. It can be seen that MUX has a 8.7% higher execution time than MEM and CS has a 21.53% higher execution time than MEM since there is no reconfiguration needed in the processor for the (32,9) configuration.

Figure 9 shows the impact of different reconfiguration schemes when the system is using the (32,7) case. This changes the number of active clusters to 16 clusters during Viterbi decoding from 32-clusters during multiuser detection and channel estimation. In this case, MEM performs worse than MUX, because the data for Viterbi decoding is now sent via external memory and reconfigured to 16 clusters, adding memory stalls in the system for reconfiguration. The MUX network has a 3.2% lower execution time than MEM and has a 14.63% lower execution time than CS.

Among the three different reconfiguration schemes considered, the multiplexer network solution provides the lowest power for reconfiguration for wireless base-station processing. This is because MUX has the lowest amount of data transfer and storage for completing the cluster operations while having the maximum bandwidth for data transfer between the different reconfiguration schemes. As will be shown in the next section, the multiplexer network by itself does not impact the power. MEM requires transfers to external memory while CS requires the buffers within all clusters and the inter-cluster communication network for all clusters to be kept active. The MUX network also allows the clusters to turn off completely during memory operations between kernels and during system initialization, allowing scope for further power savings. We compare the power savings using reconfiguration with a MUX network against a stream processor using voltage-frequency scaling in the following section.

6. POWER MODEL

The power model for stream processors is based on [11] with additions to account for the multiplexer network. The parameters used in power analysis in this paper are as shown in Table 2 and Table 3. All energy values shown are with respect to E_w. Based on the models in [11], with modifications to account for fixed point ALUs, the length of a wire track is 0.455 micron, a cluster is 1400 wire tracks wide and it takes 0.42 pJ to propagate a signal across a single wire track. The modifications are based on a low power version of stream processors for embedded systems (See (LP) in Chapter 6 of [12]). The power consumption of the mux network is wire length capacitance dominant to a first order approximation [11]. Assuming a linear layout of the clusters (worst case), the total wire length of the mux network for 1 bit is 341 units, where the unit is the distance between 2 clusters. For 8 streams with 32 bits of wire each, the mux network uses a total wire of length approximately $341 * 1400 * 8 * 32 * 0.455 = 55$ meters with a power consumption of 5 μJ [11]. The power switching device needs to be large enough in width to handle the average supply current during operation. Note that for turning off clusters using power gating, the width required would exceed maximum transistor widths. This is because, the clusters consume $\sim$80% of the chip power. Hence, in this case, multiple power gating devices need to be used to turn off a single cluster.

The addition of a gating device can result in reduced performance and decreased noise margins due to the drop across the gating device and the parasitic capacitances added. Also, there is a latency between the arrival of the signal to turn on the functional unit and the operation of the functional unit. Several clock cycles will be needed to allow the power supply to the clusters to reach its operating level due to the charging of the cluster capacitance. An on-off latency of 7 cycles is used and this number is based on clocking the multiplexer network. The pipelining depth of the multiplexer network is $\log_2(clusters)$. For a 32-cluster architecture, this evaluates to 5 cycles. In addition, 2 cycles are added during interfacing the network to the SRF and the clusters.

6.1 Voltage-Frequency Scaling

Figure 10 shows the clock frequency needed by the stream processor using MUX reconfiguration to meet real-time requirements of 128 Kbps/user. We can see that as the workload decreases, the percentage of cluster busy time and microcontroller stalls decrease.

Table 2: Power consumption (normalized to E_w units)

Description	Value
ALU operation energy	632578
Local Register File (LRF) access energy	79365
Energy of SB access/bit	155
SRAM access energy/bit	8.7
Scratchpad access energy	1603319
Norm. wire energy/track(E_w)	1

Table 3: Key simulation parameters

Parameter	Value
No. of clusters	32
Adders/cluster	3
Multipliers/cluster	3
Stream block size	32 words
Stream Buffer size	256 KB
DRAM frequency	$f_{CLK}/8$
Adder latency	2 cycles
Multiplier latency	4 cycles
Off-On latency	$\leq$ 7 cycles

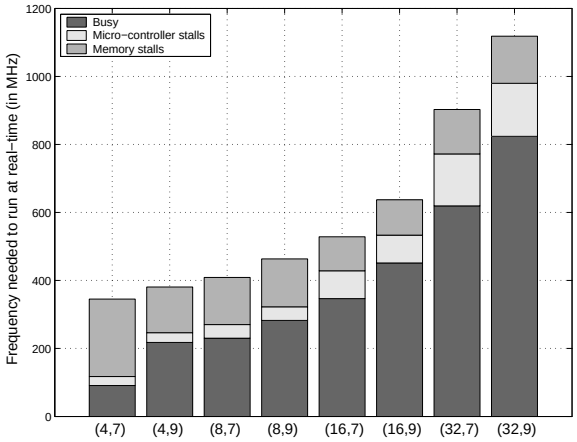

Figure 10: Clock frequency needed to meet real-time with varying workload : (32,9) to (4,7)

However, the memory stall time does not decrease with the workload. This is because as the workload decreases from (32,9) to the (4,7) case, some of the memory stalls that were hidden during kernel execution suddenly became visible due to the corresponding decrease in the kernel execution time. The reconfigurable stream processor needs to run at 1.12 GHz to meet real-time for the full capacity workload and can run at 345.1 MHz when the workload decreases to the (4,7) case. Stream processors are scalable with technology and clusters and can attain GHz clock speeds for 32-cluster configurations with 3 adders and multipliers per cluster [11]. Figure 11 shows the corresponding cluster utilization variation with the workload and the cluster index. The Y-axis shows the percentage utilization of the clusters while the X-axis shows the cluster identification number of each cluster which goes from 1 to 32 for a 32-cluster stream processor. We can see that in the fully loaded case (32,9), all clusters are equally utilized at 87%. The clusters are idle and are turned off 13% of the time due to memory stalls. However, as the workload decreases, the stream processor using MUX turns off unutilized clusters to lower their utilization factor and save power. For example, we can see in case (4,9) that only the first 4

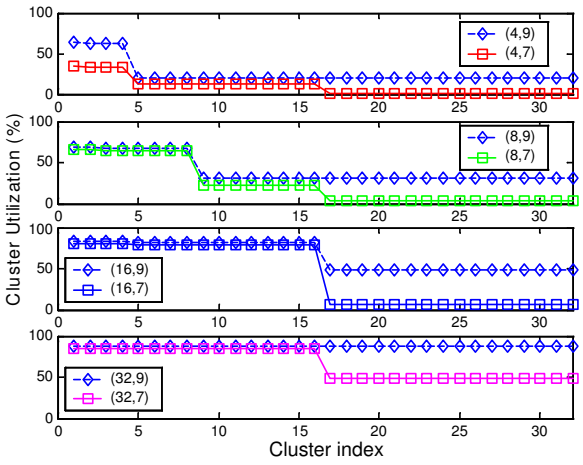

Figure 11: Cluster utilization variation with cluster index and with workload. X-axis represents the cluster id (1:32)

clusters are being used at 63% utilization, while the remaining 28 clusters are being used at 20% utilization. The multiplexer network provides the needed data alignment to collect the data from all the 32 SRF banks and stream buffers into only those clusters that are active. Lower cluster index numbers have a higher cluster utilization as the multiplexer network favors clusters with lower index numbers during reconfiguration. Thus, by turning off unused clusters during periods of memory stalls and insufficient data parallelism, stream processors with dynamic reconfiguration are able to provide power-efficient solutions for applications with varying levels of data parallelism.

Many embedded processors now provide power savings using voltage-frequency scaling techniques such as those used in Intel's XScale and Transmeta's Crusoe [13, 14] processor. In this paper, we assume frequencies and voltages used in Transmeta's 5800 Crusoe processor[14] with extensions to up to 1.2 GHz at 1.4 V.

It is not practical to run the processor clock at just the exact frequency to meet real-time. There are only a few possible frequency levels in programmable processors and these are usually standardized to simplify the interface with external DRAM. Hence, the reconfigurable stream processor needs to be over-clocked to work at these fixed frequencies. However, the clusters can be turned off during spare cycles now available as well. Power saving is achieved in the stream processor with MUX due to turning off clusters during over-clocking (the idle time due to mismatch between the frequency needed and the actual frequency used), during memory stalls and during kernels having clusters with insufficient data parallelism. This is shown in Table 4. In order to evaluate the benefits of the multiplexer network, the base case comparison is assumed to be a stream processor that already supports dynamic voltage and frequency scaling. For example, the (16,7) workload runs at 533 MHz and 0.95 V and consumes 2.98 W in a base configuration using voltage and frequency scaling (without turning off any clusters). With the multiplexer network added, it saves 0.02 W by turning off clusters in the spare time available during over-clocking, 0.44 W by turning off clusters during memory stalls and 0.808 W by turning off unutilized clusters when the parallelism falls below 32. Thus, the mux network addition lowers the power consumption of the processor to 1.71 W and provides a power savings of 42.54%. We can see also from the table that the multiplexer network yields

Table 4: Power savings using the MUX network for reconfiguration for various workloads from (4,7) to (32,9)

Workload	Freq (MHz)		Voltage (V)	Power Savings (W)			Power (W)		Savings
	needed	used		over-clocking	memory stalls	unutilized clusters	New	Base	
(4,7)	345.09	433	0.875	0.325	1.05	0.366	0.30	2.05	85.14 %
(4,9)	380.69	433	0.875	0.193	0.56	0.604	0.69	2.05	66.41 %
(8,7)	408.89	433	0.875	0.089	0.54	0.649	0.77	2.05	62.44 %
(8,9)	463.29	533	0.950	0.304	0.71	0.643	1.33	2.98	55.46 %
(16,7)	528.41	533	0.950	0.020	0.44	0.808	1.71	2.98	42.54 %
(16,9)	637.21	667	1.050	0.156	0.58	0.603	3.21	4.55	29.46 %
(32,7)	902.89	1000	1.300	0.792	1.18	1.375	7.11	10.46	32.03 %
(32,9)	1118.25	1200	1.400	0.774	1.41	0.000	12.38	14.56	14.98 %
Estimated cluster power consumption									78 %
Estimated SRF power consumption									11.5 %
Estimated microcontroller power consumption									10.5 %
Estimated chip area									45.7 mm^2

savings in power even in the no reconfiguration case (32,9) of a fully loaded base-station due to turning off clusters during memory stalls and over-clocking. Thus, the multiplexer network yields an additional 15–85% power savings over that provided by simple frequency and voltage scaling in the stream processor.

7. CONCLUSIONS

We provide mechanisms for improving the power efficiency of stream processors when the data parallelism falls below the number of clusters. Clusters form the dominant source of power consumption and turning off unutilized clusters can significantly improve the power efficiency. For a wireless base-station example, we show that 15–85% power savings can be achieved over and above a stream processor that uses conventional power saving techniques such as dynamic voltage and frequency scaling. We have shown 3 different methods by which reconfiguration support could be provided in stream processors to enhance their power efficiency. This dynamic reconfiguration support extends stream processors from traditional high performance applications to power-sensitive applications in which the data parallelism varies dynamically and falls below the number of clusters.

8. ACKNOWLEDGEMENTS

We are grateful to the Imagine team at Stanford University for their help and support with the simulator. Sridhar Rajagopal and Joseph Cavallaro were supported in part by Nokia Corporation, Texas Instruments, Inc., and by NSF under grants EIA-0224458, and EIA-0321266.

9. REFERENCES

[1] U. J. Kapasi, S. Rixner, W. J. Dally, B. Khailany, J. H. Ahn, P. Mattson, and J. D. Owens, "Programmable stream processors," *IEEE Computer*, vol. 36, no. 8, pp. 54–62, August 2003.

[2] S. Rajagopal, S. Bhashyam, J. R. Cavallaro, and B. Aazhang, "Real-time algorithms and architectures for multiuser channel estimation and detection in wireless base-station receivers," *IEEE Transactions on Wireless Communications*, vol. 1, no. 3, pp. 468–479, July 2002.

[3] S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice Hall, 1 edition, 1995.

[4] Paul R. La Monica, "Wireless gets blacked out too - North America Power Blackout," CNN Money News report: http://money.cnn.com/2003/08/15/technology/landlines/, August 16 2003.

[5] S. Rixner, W. Dally, B. Khailany, P. Mattson, U. Kapasi, and J. Owens, "Register organization for media processing," in *6th International Symposium on High-Performance Computer Architecture*, Toulouse, France, January 2000, pp. 375–386.

[6] N. Mera, "The wireless challenge," Telephony Online, http://telephonyonline.com/ar/telecom_wireless_challenge, December 9 1996.

[7] "Implementation of a WCDMA Rake receiver on a TMS320C62x DSP device," Tech. Rep. SPRA680, Texas Instruments, July 2000.

[8] "Channel card design for 3G infrastructure equipment," Tech. Rep. SPRY048, Texas Instruments, 2003.

[9] S. Rajagopal, S. Rixner, and J. R. Cavallaro, "Design-space exploration for real-time embedded stream processors," *IEEE Micro (special issue on embedded systems)*, vol. 24, no. 4, pp. 54–66, July-August 2004.

[10] U. Kapasi, W. Dally, S. Rixner, P. Mattson, J. Owens, and B. Khailany, "Efficient conditional operations for data-parallel architectures," in *33rd Annual International Symposium on Microarchitecture*, Monterey, CA, December 2000, pp. 159–170.

[11] B. Khailany, W. J. Dally, S. Rixner, U. J. Kapasi, J. D. Owens, and B. Towles, "Exploring the VLSI scalability of stream processors," in *International Conference on High Performance Computer Architecture (HPCA-2003)*, Anaheim, CA, February 2003, pp. 153–164.

[12] B. Khailany, *The VLSI implementation and evaluation of area- and energy-efficient streaming media processors*, Ph.D. thesis, Stanford University, Palo Alto, CA, June 2003.

[13] L.T. Clark et al., "An embedded 32-b microprocessor core for low power and high performance applications ," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 11, pp. 1599–1608, 2001.

[14] Transmeta, "Crusoe Processor Product Brief: Model TM5800," http://www.transmeta.com/pdf/specifications/TM5800_productbrief_030206.pdf.