Stream Processors and their Applications for the Wireless Domain

Binu Mathew and Ali Ibrahim
{mbinu | ibrahim} @ {cs.utah.edu}

Abstract

A stream is a sequence of similar data records with real-time throughput or bandwidth constraints attached to it. Examples include link-level encryption in networks, video trans-coding, video compression, cellular telephony as well as the image and speech pA stream is a sequence of similar data records with real-time throughput or bandwidth constraints attached to it. Examples include link-level encryption in networks, video trans-coding, video compression, cellular telephony as well as the image and speech processing. Stream programs consist of a data-flow network where the nodes called kernels represent simple algorithms that transform an input block to an output block with access to a limited amount of history. Stream processors are decoupled access/execute processors whose architecture has been optimized for the repeated application of stream computations to high bandwidth data-streams at very high levels of performance and energy efficiency.

Stream processors have been extensively studied in academia by projects such as Stanford Imagine and MIT RAW. They made their commercial debut with the Cell, a broadband processor from IBM that is the computing engine for the Sony PlayStation 3. The cellular telephony market has experienced rapid growth around the world and represents a significant opportunity for stream processors because this domain requires very high computation rates to reduce the bit error rate and to support high data rates, full motion video and multimedia applications, and a variety of wireless standards. Simultaneously, they must also be energy efficient and flexible, have a low time to market, and be low cost.

This article starts by providing an overview of the fundamental concepts behind stream processors, their applications to perception, media, wireless and scientific workloads, major research projects etc. It will elaborate on the nature of 3G and 4G wireless algorithms, architectural approaches to optimize these algorithms as well as commercial processors that have been optimized for the wireless domain.

1 Introduction

Many embedded, media and digital signal processing applications involve simple repeated computations on a very long or never ending sequence of data. There is limited access or no access at all to past data. A good example is an image processing system such as the simplified version of a face recognition system shown in Figure 1. This surveillance system accepts a video stream from a camera, identifies the pixels that have human skin color, segments the image into regions

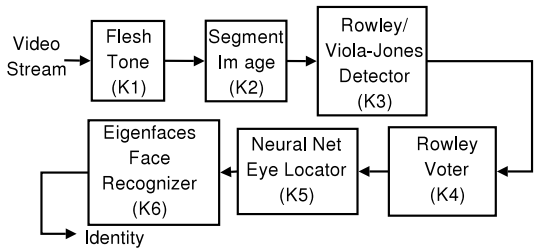

Figure 1: Face Recognizer: An Example Stream Application

that contain skin or no-skin, uses a neural network based algorithm to identify regions that may contain a face, uses another neural network based algorithm to locate the eyes and then tries to match the face against a database of known faces to obtain a persons identity. Details of the system may be found in [12]. The application represents a well structured assembly of simple compute intensive algorithms. Data-flow between the component blocks is regular and predictable. The whole computation may be abstracted as a data-flow graph consisting of a few key procedures and an input stream and an output stream. We say that applications with such simple regular structures are "stream-able" and the style of computation is called "stream processing". Other examples include link-level encryption in networks, video trans-coding, video compression, cellular telephony as well as the image and speech processing. Even though stream optimized processor hardware is a relatively new area, stream oriented techniques are ubiquitous in the software world with Unix pipes being a prime example.

1.1 Rationale

Most current research in processor architecture revolves around optimizing four criteria: energy (or power), delay, area and reliability. For scientific applications, memory bandwidth is also a precious commodity that critically affects delay. One or more of these criteria can often be traded off for the sake of another. For processors, it is often the case that the product of energy and delay required to process a given work load is relatively constant across different architectures after normalizing for the CMOS process [2]. To achieve a significant improvement in the energy-delay product, the architecture needs to be optimized to exploit the characteristics of the target application. Stream applications exhibit three forms of parallelism that can be taken advantage of: instruction level parallelism (execute independent instructions in parallel), data parallelism (operate on multiple data elements at once, often using SIMD) and task parallelism (execute different tasks in parallel on different processors).

Current high-performance dynamically scheduled out-of-order processors are optimized for applications that have a limited amount of instruction level parallelism (ILP). They do not depend heavily on task level parallelism as evidenced by the fact that most processors support only one or two way SMT and one or two cores per chip. Deep sub-micron CMOS processes offer the opportunity to fabricate several thousands of 32-bit adders or multipliers on a $10 \times 10\,mm$ microprocessor die. Yet, because of the limited ILP and irregular nature of typical applications most micro-processors have six or fewer function units. The bulk of the area and power is consumed by caches, branch predictors, instruction windows and other structures associated with identifying

and exploiting ILP and speculation. Stream applications on the other hand have very regular structures, are loop oriented and exhibit very high levels of ILP. In fact, Kapasi et al report being able to achieve 28 to 53 instructions per cycle for a set of stream applications on the Imagine stream processor [9]. To achieve such high levels of parallelism, stream processors typically utilize a large number of function units that are fed by a hierarchy of local register files, stream register files and stream caches that decouple memory access from program execution. The resulting bandwidth hierarchy can exploit data parallelism and main memory bandwidth much more efficiently than traditional processors resulting in better performance per unit area or unit power. In addition, it is possible to construct multiple stream processors on the same chip and use stream communication mechanisms to ensure high bandwidth data flow and exploit task level parallelism. The trade-off when compared to general purpose processors is a much more restrictive programming model and applicability that is limited to the streaming domain

1.2 Terminology

Definitions of common stream programming terms follow.

Record — A record is either an atomic data type such as an integer or a floating point number or an aggregate of records.

Stream — A stream is a directed sequence of records. Depending on their direction, streams may be either input streams or output streams. While most streams are very long or never ending, stream programming systems also support some specialized types of streams. Constant streams have a fixed length are often repeatedly reused. Conditional streams access a record from the stream only when a condition code that is computed in a processor or a function unit cluster of a processor is true [10]. They may be input or output streams depending on the direction of access. Most streams are sequential, i.e., for input streams the algorithm can read only the next record in the stream and write the sequentially next record to an output stream. Gather streams allow arbitrary indexing to fetch records [1]. Indexed streams allow accessing elements within a constrained range of one stream to be accessed using indices from another stream [13].

Derivation — A new stream defined to contain a subset of the records from another stream is called a derived stream. For example, if the variable *x* refers to an existing input stream, y is a constant stream containing the integers (0,2,4,8) an indexed stream z may be derived from x and y that is constrained to have the range (0,8) relative to the current position of x. If $S[i]$ denotes the i^{th} record relative to the current position of stream S, then $z[0] = x[0]$, $z[1] = x[2]$, and so on. Derivations may be done using other mechanisms such as strided or bit-reversed access. The key point is that the stream compiler should be able to reason about the relationship of records of the derived and base streams.

Kernel — A kernel is a function that transforms input streams to output streams. Kernel functions are typically loop oriented computations with few or non-existent conditional branches in the loop body. Simple conditional operations in the original algorithm

may often be converted into forms such as predicated execution/if-conversion, conditional moves and input/output to conditional streams.

Stream Graph A stream graph is a directed graph where the nodes are kernels and the edges correspond to streams that convey the data between the kernels.

SRF Stream Register Files (SRF) are specialized storage structures used to hold streaming data. They are essentially large blocks of SRAM under the control of hardware that knows how to fetch and store the sequentially next element of a particular stream, perform operations for gather streams, indexed streams etc.

LRF The complexity of multi-ported register files is one of the critical factors that limits issue width in a traditional microprocessor. Rather than use a single central register file, stream architectures attach Local Register Files (LRF) to execution clusters or function units. These LRFs typically serve only the units they are attached to. Inter-cluster communication is handled by a compiler controlled communication network.

Producer-Consumer Locality It is common in stream applications for a producer kernel to generate and save some results to an SRF and the results are immediately used by a consumer kernel straight out of the SRF without saving intermediate results to lower levels of the memory hierarchy. Such reuse of intermediate results is termed producer-consumer locality.

2 The Stream Virtual Machine

The Stream Virtual Machine (SVM) is an abstract machine model that has been proposed by Labonte et al to represent the important characteristics of stream architectures and to develop techniques to compile applications and analyze their performance across different implementation architectures [11]. Their compilation technique proceeds in two stages. First a high level compiler (HLC) reads a stream application written in a stream programming language such as StreamIt or ArrayC. The HLC also reads an abstract SVM model for a stream architecture such as the MIT RAW machine or Stanford Imagine. It then uses the abstract machine model to partition the application into kernels that will execute on particular processing resources and into data transfers between the kernels. This mapping may be described in terms of functions available in the SVM API. API functions provide for initializing local memory, scheduling kernels for execution, declaring dependence between kernels, co-ordinating DMA transfers between different units etc [11]. The kernels are then compiled into binary form by a low level compiler (LLC) that is specific to the particular architecture.

An SVM model for a stream architecture consists of three types of components: processors, memories and links. Processors in turn come in three varieties. Control processors decide the sequence of operations performed by the entire machine. Control processors offload the compute intensive task of stream kernel execution to kernel processors (stream processors). Lastly, DMA engines are considered as processors that execute specialized kernels that transfer data between the many different memories in the system. The parameters that describe an SVM processor are its type (control, kernel or DMA), its operating frequency, function unit mix, degree of SIMD

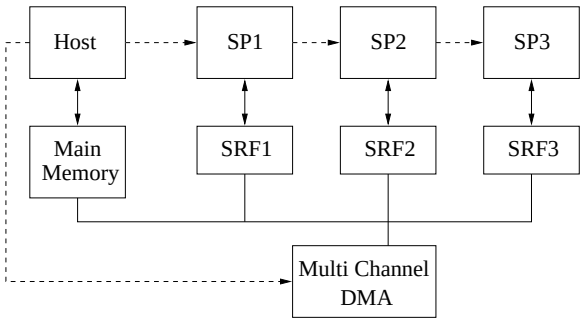

Figure 2: SVM Machine Model Example

parallelism, the number and capacity of register files etc. The memories in an SVM system may be classified depending on their access mechanism into RAMs (random access allowed), FIFOs (only sequential access allowed) and caches (associative lookup allowed). Since stream processors use a hierarchy of memories that capture producer-consumer locality to economize main memory bandwidth, a natural characterization parameter for SVM memories is the bandwidth and latency they offer to entities that are above and below them in the bandwidth hierarchy. Links allow processors and memories to communicate with each other and are characterized by their bandwidth and latency.

Figure 2 shows an SVM model to which we can map our face recognizer from Figure 1. It consists of three stream processors, a control processor and a multi-channel DMA engine that can move data between the SRFs and main memory. Solid lines indicate data paths and dotted lines indicate control paths. We next describe different types of task to resource mappings for such an application.

3 Time and Space Multiplexing

Consider a set of n kernels named $K = \{K_1, K_2, ..., K_n\}$ and m processors $P = \{P_1, P_2, ..., P_m\}$. Let $D(K_i, P_j)$ denote the execution time (delay) of kernel $K_i$when executed on processor (or processor set) P_j. When the set of processors is understood from the context, we will denote this as $D(K_i)$. Let $\mathcal{P}(P)$ denote the power set of P. A schedule S is a mapping $S(K) \rightarrow \mathcal{P}(P) \times t$ where t is the start time of the kernel. We will here after use the notation $K_i.p$ and $K_i.t$ to denote the set of processors and start time assigned to kernel K_i. Legal mappings obey the following rules:

1. $K_j.t \geq K_i.t + D(K_i)$ when ever $K_j$depends on K_i, i.e., dependences are not violated.
2. For all i, j such that $i \neq j$, and $K_i.t \leq K_j.t < K_i.t + D(K_i, K_i.p)$, it should be the case that $K_i.p \cap K_j.p = \phi$, i.e., only one kernel may be executing on a processor at any time.
3. For all i, $K_i.p \neq \phi$, i.e., every kernel should be allocated at least one processor.

A stream processing system whose schedules follow the rule, *for all* i, $K_i.p = P$ is called a time multiplexed system. In such a system, all available processing resources are allocated to

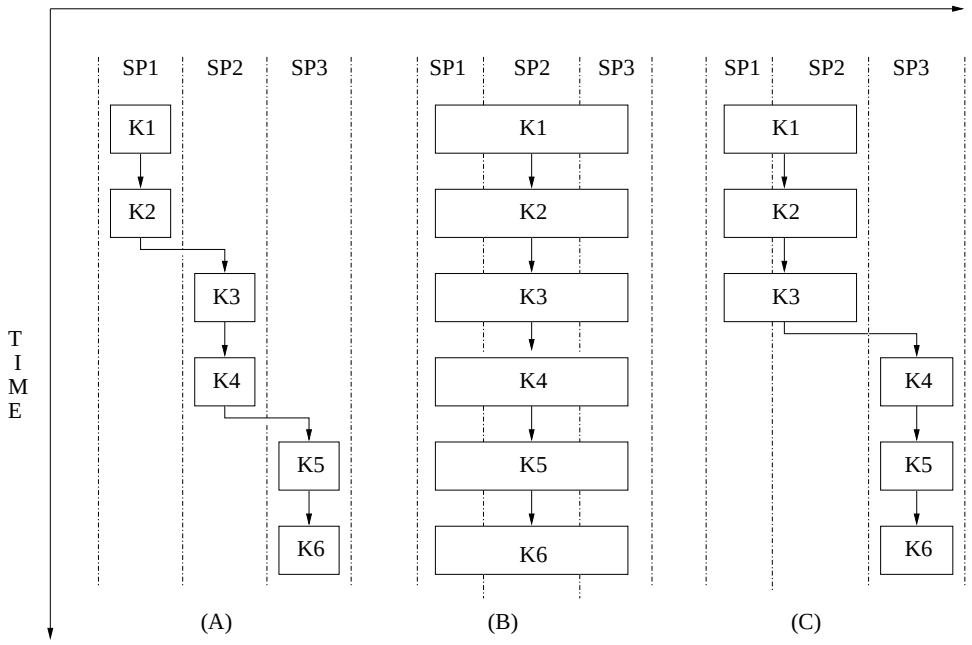

Figure 3: Example Schedules for the Application from Figure 1

one kernel and then to the next kernel and so on. A stream processing system whose schedules follow the rule, *for all* i, $|K_i.p| = 1$ is called a space multiplexed system. In such a system, only one processor is ever allocated to one kernel. If there is some i, j such that $i \neq j$, and $K_i.t \leq K_j.t < K_i.t + D(K_i, K_i.p)$ and $|K_i.p| > 1$, the system is said to be space-time multiplexed. In that case, multiple kernels execute simultaneously and some kernels are given more than one processor. Figure 3 shows three different mappings for the application from Figure 1 on to the architecture from Figure 2. By our definitions, Figure 3(A) is a space multiplexed schedule, Figure 3(B) is a time multiplexed schedule and Figure 3(C) is a space-time multiplexed schedule. To simplify the example, DMA transfers are not shown.

To consider the relative benefits of time and space multiplexing we present a simplified analysis that considers only two kernels $K_1$and K_2 where K_2depends on K_1. A more detailed theoretical framework has been developed by the authors, but the details will be deferred to a later publication. Consider the case where the amount of data parallelism available is much larger than the number of processors m. For the space multiplexed case, we follow the schedule:

$S(K_1) = (0, \{P_1, P_2, ..., P_{m/2-1}\})$ and $S(K_2)$=$(D(K_1), \{P_{m/2}, P_{m/2+1}, ..., P_m\})$.

For the time multiplexed case, we follow the schedule:

$S(K_1) = (0, \{P_1, P_2, ..., P_m\})$ and $S(K_2) = (D(K_1), \{P_1, P_2, ..., P_m\})$.

Because of abundant data parallelism, we can assume that execution time of a kernel is inversely proportional to the number of processors allocated to it. Then, $Throughput_{time\,mux} = \frac{1}{D(K_1)+D(K_2)}$ and $Throughput_{space\,mux}$=$\frac{1}{MAX(2\times D(K_1), 2\times D(K_2))}$=$\frac{1}{2\times D(K_1)}$ (Arbitrarily picking K_1 as the larger term).

Therefore, $Throughput\,ratio_{time\,mux/space\,mux} = \frac{MAX(2\times D(K_1), 2\times D(K_2))}{D(K_1)+D(K_2)}$. Since this ratio is greater than or equal to one, time multiplexing works better than space multiplexing in this case. Intuitively, the space multiplexed version works like a pipeline where the stage delays are un-

balanced and the pipeline shifts at the speed of the slowest stage while the space multiplexed version is perfectly load balanced. In addition, it is possible to convert the time multiplexed system into an m-way SIMD version where all m copies share the same control logic and instruction memory leading to lower area and higher power efficiency. When the stages are balanced $MAX(2 \times D(K_1), 2 \times D(K_2)) = 2 \times D(K_1) = 2 \times D(K_2)$. Then the throughput ratio becomes one, but time multiplexing is still better because of higher area and power efficiency.

The situation is quite different when the data/instruction level parallelism is limited. Let kernels K_1 and K_2 each require N_1 and N_2 instructions worth of computation respectively. Assume that all instructions require one cycle. Further, assume that dependences limit the peak IPC possible to I_1 and I_2 where $I_1, I_2 \leq m/2$. Then, $D(K_1) = N_1/I_1$ and $D(K_2) = N_2/I_2$. Then, $Throughput_{time\,mux} = \frac{1}{\frac{N_0}{I_0}+\frac{N_1}{I_1}}$ and $Throughput_{space\,mux} = \frac{1}{MAX(\frac{N_0}{I_0},\frac{N_1}{I_1})}$. Therefore,

$Throughput\,ratio_{time\,mux/space\,mux} = \frac{MAX(\frac{N_0}{I_0},\frac{N_1}{I_1})}{\frac{N_0}{I_0}+\frac{N_1}{I_1}}$.

Since this quantity is less than one, space multiplexing is the better alternative in this case. Intuitively, time multiplexing lets execution resources go waste while space multiplexing shares the resources leading to better performance. As before, when the load is perfectly balanced they have equivalent throughput, but time multiplexing is the better option in that case. On a practical note, when time multiplexed architectures are based on very wide SIMD execution, it is often cumbersome to re-formulate algorithms to match the wide SIMD model. Programmers might find it much more convenient to express an algorithm as a pipeline of tasks in a form suitable for space multiplexing. Programming languages like StreamIt attempt to automatically compile space-multiplexed code to space-time multiplexed binaries, but further research in this area is required to take advantage of the efficiency of time multiplexed architectures.

4 Stream Processor Implementations

Now that we have introduced a minimal framework to reason about various styles of stream processing, we present an overview of three specific implementations: Stanford Imagine, MIT RAW and IBM's Cell processor.

4.1 Imagine

The Imagine image and signal processor developed by Prof. William Dally and the Concurrent VLSI Architecture Group (CVA) at Stanford university was the pioneering project in stream processing [9]. Figure 4 shows the internal structure of an Imagine processor. The Imagine processor consists of eight execution clusters where each cluster contains six ALUs resulting in a peak execution rate of 48 arithmetic operations per second. Each cluster executes a VLIW instruction under the control of the micro-controller. The same VLIW instruction is issued to all clusters resulting in the instruction fetch and control overhead being amortized over 8-way SIMD execution. The bandwidth hierarchy consists of local register files (LRF) attached to each function unit that provide 435GB/s, a stream register file (SRF) that feeds the LRFs at 25.6 GB/s and a streaming memory system that feeds the SRF at 2.1 GB/s. The LRFs within each cluster are connected directly to function units but can accept results from other function units over an intra-cluster switching network. Each cluster also contains a communication unit that can send and receive results from

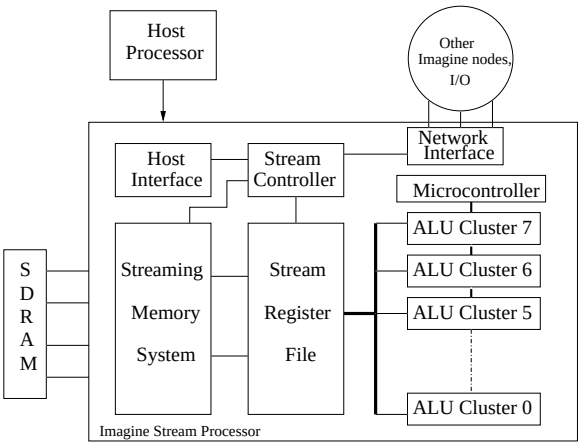

Figure 4: The Imagine Stream Processor

other units over an inter-cluster switching network. The SRF is internally split into banks that serve each cluster. In addition, each cluster also has a 256 word 32-bit scratchpad memory. The host processor queues kernels for execution with the stream controller via the host interface. The stream controller initiates stream loads and stores via the stream memory system, uses an internal scoreboard to ensure that dependences are satisfied, and then lets the micro-controller sequence the execution of the next kernel function whose dependences have all been satisfied. The Imagine processor was fabricated in a 0.18μCMOS process and achieved 7.96 GFLOPS and 25.4 GOPS at 200 MHz. Imagine was succeeded by the Merrimac project where the focus was on developing a streaming super computer for scientific computing.

Both Imagine and Merrimac were developed primarily around the concept of time multiplexing. Thus they are optimized for applications with very high levels of data parallelism. In a multi-chip configuration, it is possible to space-time multiplex these systems by making kernels on each node communicate with their counterparts on other nodes over a network interface.

4.2 RAW

The RAW processor is a wire delay exposed tiled architecture developed by Prof. Anant Agarwal and the Computer Architecture Group (CAG) at MIT as a part of the Oxygen ubiquitous computing project [17]. Increasing wire delays in sub-micron CMOS processes and the demand for high clock rates have created a need to decentralize control and resources and distribute resources as semi-autonomous clusters that avoid the need for single-cycle global communication. The RAW processor approaches this problem by splitting the die area into a square array of identical tiles and the tiles communicate with each other over a mesh network. Each tile contains an 8-stage in-order single issue MIPS-like processor with a pipelined FPU, 32 KB of instruction cache, 32 KB of data cache and routers for two static and two dynamic networks that transport 32-bit data. The routers have another 64 KB of instruction cache. Point to point transport of scalar values is done over the high performance static network that is fully compiler controlled and guarantees in-order operand delivery. The dynamic network routes operations such as I/O, main memory traffic and inter-tile message passing that are difficult to fully schedule statically. The static router controls

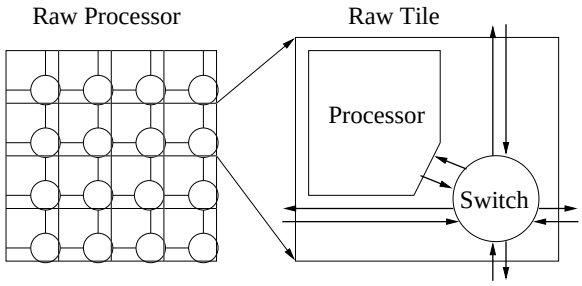

Figure 5: The RAW Processor

two cross bars each with seven inputs namely the four neighboring tiles in the square array, the router pipeline itself, the other crossbar and the processor. For tiles on the periphery of the chip, some of the links connect to external interfaces. The tiles and the static router are designed for single cycle latency between hops. The compiler encodes the routing decisions for the crossbars into a 64-bit instruction that is fetched from a 64KB instruction cache and executed by the static router. Inter-tile communication latency is reduced by integrating the network with the bypass paths of the processor. A 225 MHz implementation of a 16 tile RAW processor was fabricated in a 0.18μ CMOS process and achieved speedups of 4.9 to 15.4 over a 600 MHz Pentium 3 for a set of stream oriented benchmarks written in the StreamIt language.

Because of its independent threads of execution in each tile, the RAW processor is capable of performing time, space and space-time multiplexing. The StreamIt language mostly exposes a space multiplexed programming model even though the compiler is capable of partitioning kernels and load balancing them for space-time multiplexing.

4.3 The Cell

Stream processors made their commercial debut in 2005 with the Cell Broadband Engine Architecture (CBEA) from IBM that was developed under collaboration with Toshiba and Sony [14]. This architecture is optimized for a range of compute intensive applications varying from computer games, cryptography, graphics transformations and lighting to scientific workloads. The Cell consists of a 64-bit Power processor that serves a similar function to the host processor of Imagine and eight streaming units names Synergistic Processing Elements (SPE). Each SPE is capable of 128-bit SIMD operations that may be two 64-bit, four 32-bit, eight 16-bit or 16 byte-wide operations. An SPE consists of two pipelines. The even pipeline executes floating point and integer arithmetic while the odd pipeline handles branches, memory accesses and permutations. Up to two instructions may be issued in-order per cycle to a set of seven function units. The bandwidth hierarchy consists of a 128 word 128-bit LRF in each cluster that is filled from a 256 KB local store (SRF) that is in turn serviced by a globally coherent DMA engine. Interestingly, the SRF also serves as the instruction store for an SPE. Like in the case of the RAW processor, each SPE has its own thread of execution and the system is capable of performing time, space and space-time multiplexing. The Cell processor was fabricated in a $90nm$ CMOS process, has a peak operating frequency of 4 GHz and achieves a SIMD speedup of 9.9 times on a set of compiled benchmarks.

5 Stream Processing for Wireless Systems

Until now, most research in stream processing has addressed media and scientific applications. Wireless communication in forms such as cellular telephony systems and local and personal area networks has become a ubiquitous part of modern life. Mobile wireless systems have relatively high demands for processing and power efficiency and represent a new domain in which stream processors could be quite useful. With around one billion cell phones sold annually, the volume in this market provides economic justification for the development of specialized processor architectures. We present a brief overview of cellular communication technology and indicate avenues for the application of stream processors.

First generation (1G) cellular systems using analog technology were introduced in Scandinavia in 1981 and were followed by similar systems in the United states. They provided only voice transmission. The first digital cellular systems that made their appearance in 1990 and were termed second generation wireless (2G) systems. They provided better voice quality and added data services support with transmission rates up to 9.6kbits/s. To support high data rates and to be able to provide multimedia services anytime and anywhere, the International Telecommunications Union defined a family of systems for the third generation (3G) mobile telecommunications called IMT-2000. The 3G system provides data rates up to 2 Mbits/s for stationary users, 384 Kbits/s for pedestrians, and 144 Kbits/s for vehicular users. The services offered by 3G systems can be divided into different classes depending on their delay sensitivity. Voice, video telephony, and video games are delay sensitive. Email, short message service, and data downloads are not delay sensitive. In this article, we explain the Wide-band Code Division Multiple Access (WCDMA) system that is commonly used in 3G systems.

The quest to improve data rates and quality of service and to provide seamless roaming and global mobility for voice and data services, a new wireless standard (4G) is currently being formulated. The technologies that would most likely play an important role in 4G are Software Defined Radio (SDR) and Multiple Input Multiple Output (MIMO) antenna systems. 3G and 4G systems consist of several layers each providing a specific function. The following sections will only focus on the physical layer where all the compute intensive algorithms are located.

6 WCDMA Physical Layer

Figure 6 shows the block diagram of a receiver that uses WCDMA physical layer technology. We focus on the receiver rather than the transmitter, since the latter has a much lower computational complexity than the former.

At the output of the A/D converter, the signal is first filtered using a Root-Raised Cosine (RRC) filter that reduces inter-symbol interference. Subsequently, the signal is fed to a rake receiver and a searcher. Due to the possibility of multi-path propagation, the rake receiver has a number of fingers called correlators that individually process several multi-path components. In each of these fingers, the signal is unspread by multiplying it by a unique PN code used by the transmitter. This operation separates the desired data signal from interfering signals. The outputs from different correlators are then combined to achieve improved reliability and performance. Depending on the number of multi-path components, the number of correlators varies between two and six. The searcher provides an estimate of the number of multi-paths and their relative delays. The rake

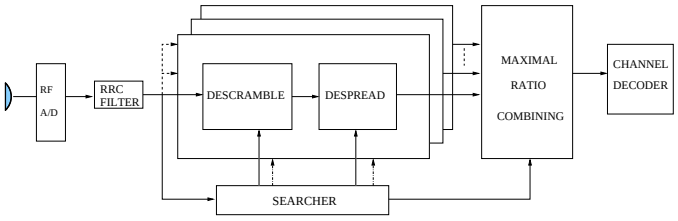

Figure 6: Functional diagram of the most compute intensive receiver algorithms

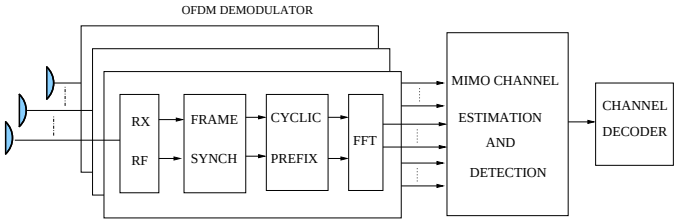

Figure 7: MIMO-OFDM receiver

receiver uses this information to control the number of correlators and the delay between them. Finally, the signal is passed through a turbo decoder for error correction. Turbo coding is used in 3G wireless cellular systems because of its outstanding error correction capabilities.

All of these algorithms are dominated by inner loops of low to moderate complexity that are applied to real time data streams. The loop bodies tend to have a high degree of parallelism and have operations such as complex-correlation arithmetic that could benefit from processor specialization. The simple regular data-flow and real-time requirements make this system a good candidate for acceleration using customized stream processors.

7 4G

As explained before, it is anticipated that MIMO and software radio technologies may play a key role in 4G systems. The use of a MIMO system could significantly increase the data rate but on compared to 3G it could also increase the computational complexity by 1-2 orders of magnitude, depending on the number of antennas. There are several different flavors of MIMO based systems such as MIMO-OFDM (Orthogonal Frequency Division Multiplexing) and MIMO MC-CDMA (Multi-Carrier Code Division Multiple Access). A simplified block diagram of a MIMO-OFDM receiver is shown in Figure 7. MIMO-OFDM combines OFDM and MIMO techniques to realize good spectral efficiency and high throughout. It can transmit OFDM modulated data from multiple antennae simultaneously. The receiver first performs OFDM demodulation then does MIMO decoding to extract data from all the transmit antennae and sub-channels.

In software defined radio technology, a large portion of the radio frequency functionality such as the intermediate frequency (IF) stage, bit-stream processing, modulation/demodulation, and source processing are performed in software running as opposed to the traditional approach of using RF circuits. SDR provides the opportunity to implement multi-mode terminals that can operate

Algorithm	Approximate MIPS
Digital Filter (RRC, Channelization)	3000
Searcher	1500
Rake	650
Maximal-ratio combining	24
Channel estimator	12
AGC, AFC	10
De-interleaving, rate matching	14
Turbo decoding	52
Sum	5262

Table 1: Computation Requirements for 3G WCDMA Receiver

at several different frequency ranges. New services and advanced signal processing techniques can be easily implemented and tested using the SDR approach. Its re-programmability allows it to download algorithms on demand, and intelligently adapt radio interfaces to different applications and environments. Both MIMO and SDR are computationally intensive techniques with well structured data-flow and are amenable to stream processing.

8 Computational Complexity and Power Consumption

The computational requirements imposed by cellular standards has increased exponentially from 1G to 4G. This is due to the increased complexity of algorithms introduced to reduce the bit-error rate use of the wireless spectrum more efficiently. These algorithms have been shown to require more performance than is currently available in embedded processors. Table 1, shows the computation requirements to handle 384 Kbits/s transmission rate on a 3G WCDMA receiver[3]. The problem will persist in the future since the computation requirements are growing faster than Moore's law[3]. Power dissipation is also a major problem in battery powered mobile computing and communication devices. While the computational requirements are increasing exponentially, battery capacity is only improving at the rate of 1.03 times per year[15]. Flexibility and low time to market require the use of programmable processors for the implementation of the increasingly sophisticated digital signal processing algorithms. Power efficiency on the other hand, requires the use of a customized solution. To make 4G systems usable, the performance per unit power consumption of processors needs to improve significantly – an area in which stream processors have a significant advantage.

9 Current Solutions

As explained in the previous sections, wireless applications can be partitioned into different components that can be space-time multiplexed efficiently. This structural simplicity has given rise to several multi-processor System-on-Chip (SoC) architectures with the OMAP from TI, SB3010 from Sandbridge technologies and the EVP processor from Philips being prominent examples.

9.1 Texas Instruments OMAP

The OMAPV2230 is one of the interesting SoCs from TI's OMAP-VOX product family[8]. It is an integrated Universal Mobile Telecommunications System (UMTS) solution for 3G handsets. It integrates a digital base-band system and an applications processor. The digital base-band system consists of an ARM processor for control purposes, a TI TMS320C55x processor for DSP algorithms, and a custom ASIC module that handles the compute intensive portions of 3G base-band processing. The applications processor includes a dedicated 2D/3D graphics accelerator and an Image Video Audio Accelerator (IVA).

9.2 Philips EVP

The Embedded Vector Processor (EVP) developed by Philips is a VLIW based scalable SIMD architecture designed to support multiple 3G standards[18]. The EVP includes several specialized function units such as a shuffle unit, an intra-vector unit that supports intra-vector operations, and a code generation unit that supports CDMA code generation. It also includes an address calculation unit that supports an extensive set of addressing modes. The EVP exploits VLIW parallelism by providing the ability to issue five vector operations, four scalar operations, address updates and loop control operations simultaneously.

9.3 Sandbridge Technologies SB3010

Sandbridge Technologies has implemented a 3G multimedia handset design using their SB3010 base-band processor[16]. The processor integrates an ARM 9 RISC core, four of Sandbridge's own Sandblaster DSP cores, on-chip instruction caches and data memories and a programmable RF interface. In most broadband communication systems data is streamed from an A/D converter. To accommodate for this, their design uses scratchpad memories rather than data caches. Each Sandblaster core delivers 2 billion MAC operations per second and supports eight hardware threads. With a total performance of the order of 10 billion MACs per second, the SB3010 is able to run different wireless protocols such as WCDMA, as well as multimedia codecs such as MPEG-4 H.264 and MP-3.

The DSP architecture can be partitioned into an instruction fetch and branch unit, an integer and load store unit, and a SIMD vector unit. This SIMD unit consists of four vector processing elements (VPE), an accumulator register file, a shuffle unit, and a reduction unit. Integer operations 16, 32 and 40-bit fixed-point data types is supported. The Sandblaster DSP implements an unorthodox multi-threading method to avoid the hassles posed by data dependences and hazards in pipelined processors. Eight hardware threads are supported, but they issue instructions round-robin. Effectively, each thread issues an instruction every eighth cycle and its dependences resolve while it waits for other threads to take their turn.

10 Stream Processor Based Wireless SoCs

As shown in Figure6 and explained in previous sections wireless systems usually consists of multiple DSP algorithm kernels connected in feed forward pipelines and data is streamed between

the kernels. These kernels are typically compute bound, exhibit high levels of data parallelism, typically require low precision fixed-point arithmetic and contain bit manipulation operations that could benefit from customized instructions and function units.

The ACT stream processor developed at the University of Utah has demonstrated that compute intensive 3G base-band algorithms stream architectures perform very well on stream architectures[7, 5, 6]. The energy-delay product of this stream processor was within one to two orders of magnitude of that of an ASIC. This research is a first step toward the goal of creating high efficiency wireless SoCs based on space-time multiplexed implementations of base-band algorithms on a network of customized stream processors. Unlike the general mesh network of the RAW processor, the on-chip interconnect between the stream processors can be customized to account for the data-flows observed in 3G and 4G systems. It is known that the input from the A/D converter stage to a WCDMA system requires at most 7.68 MB/s bandwidth [4]. Communication between later kernels in a WCDMA system requires even less bandwidth. This makes it possible to use low throughput interconnects between different stream processors. Once the data is received by a particular core, it may need to be buffered and accumulated. In the case of MIMO-OFMD this takes the form of a FIFO that is required between the OFDM modulator and MIMO detection. In the case of turbo codes, this is because the algorithm operates only on blocks of data. In either case, an SRF structure that supports sequential and indexed streams would be adequate to handle the buffering requirements. Some parts of wireless processing may not be amenable to space multiplexing because of load imbalance between stages, variability in data arrival times. A mixture of distributed control, space-time multiplexing, data re-arrangement units and programmable interconnect may be required to solve all the complex challenges posed by 4G algorithms.

11 Conclusions

Stream processors have been extensively studied in academia by projects such as Stanford Imagine and MIT RAW. They made their commercial debut with the Cell, a broadband processor from IBM that is the computing engine for the Sony PlayStation 3. Research has demonstrated that stream processors can achieve very high levels of energy efficiency and performance on a variety of speech and image processing as well as wireless communication tasks [12, 7]. The cellular telephony market has experienced rapid growth around the world and represents a significant opportunity for stream processors because this domain requires very high computation rates to reduce the bit error rate and to support high data rates, full motion video and multimedia applications, and a variety of wireless standards. Simultaneously, they must also be energy efficient and flexible, have a low time to market, and be low cost. The stringent power requirements of mobile wireless systems calls for different patterns of stream processor customization than previously studied for scientific stream processors. The possibility of customizing the instruction set architecture and on-chip interconnect and performing energy-delay trade-offs for wireless optimized stream processors merits further study.

References

[1] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan. Brook for gpus: stream computing on graphics hardware. *ACM Trans. Graph.*, 23(3):777–786, 2004.

[2] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9):1277–1284, September 1996.

[3] D. Greifendorf, J. Stammen, and P. Jung. The evolution of hardware platforms for mobile "software defined radio" terminals. *Proceeding IEEE Personal, Indoor, and Mobile Radio Conference*, Sept 2002.

[4] H. Holma and A. Toskala. WCDMA for UMTS: Radio access for third generation mobile communications, second edition, john wiley & sons, 2002.

[5] A. Ibrahim and A. Davis. Address acceleration mechanisms for an adaptive cellular telephony processor. *International Conference on Multimedia and Expo (ICME)*, 2005.

[6] A. Ibrahim and A. Davis. Exploiting data context switching in a low power VLIW coprocessor. *Workshop on Optimizations for DSP and Embedded Systems (ODES), in conjunction with IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2005.

[7] A. Ibrahim, M. Parker, and A. Davis. Energy efficient cluster coprocessors. *International Conference on Acoustics, Speech, and Signal Processing(ICASSP)*, May 2004.

[8] T. Instruments. Omapv2230. Technical report, http://focus.ti.com/pdfs/wtbu/ti_omapv2230.pdf.

[9] U. Kapasi, W. J. Dally, S. Rixner, J. D. Owens, and B. Khailany. The Imagine stream processor. In *Proceedings 2002 IEEE International Conference on Computer Design*, pages 282–288, Sept. 2002.

[10] U. J. Kapasi, W. J. Dally, S. Rixner, P. R. Mattson, J. D. Owens, and B. Khailany. Efficient conditional operations for data-parallel architectures. In *MICRO 33: Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, pages 159–170, New York, NY, USA, 2000. ACM Press.

[11] F. Labonte, P. Mattson, W. Thies, I. Buck, C. Kozyrakis, and M. Horowitz. The stream virtual machine. *pact*, 00:267–277, 2004.

[12] B. Mathew. *The Perception Processor*. PhD thesis, School of Computing, University of Utah, Aug. 2004.

[13] P. Mattson, U. Kapasi, J. Owens, and S. Rixner. Imagine programming system user's guide. http://cva.stanford.edu/classes/ee482s/docs/ips_user.pdf, 2002.

[14] D. Pham, T. Aipperspach, D. Boerstler, M. Bolliger, R. Chaudhry, D. Cox, P. Harvey, P. Harvey, H. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Pham, J. Pille, S. Posluszny, M. Riley, D. Stasiak, M. Suzuoki, O. Takahashi, J. Warnock, S. Weitzel, D. Wendel, and K. Yazawa. Overview of the architecture, circuit design, and physical implementation of a first-generation cell processor.

[15] J. M. Rabaey, M. Potkonjak, F. Koushanfar, S. Li, and T. Tuan. Challenges and opportunities in broadband and wireless communication designs. In *ICCAD*, pages 76–83, November 2000.

[16] M. Schulte, J. Glossner, S. Jinturkar, M. Moudgill, S. Mamidi, and S. Vassiliadis. A low-power multithreaded processor for software defined radio. *Journal of VLSI Signal Processing Systems*, 43, 2006.

[17] M. B. Taylor, W. Lee, J. Miller, D. Wentzlaff, I. Bratt, B. Greenwald, H. Hoffmann, P. Johnson, J. Kim, J. Psota, A. Saraf, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal. Evaluation of the raw microprocessor: An exposed-wire-delay architecture for ilp and streams. In *ISCA '04: Proceedings of the 31st annual international symposium on Computer architecture*, page 2, Washington, DC, USA, 2004. IEEE Computer Society.

[18] K. van Berkel, F. Heinle, P. P. Meuwissen, K. Moerman, and M. Weiss. Vector processor as an enabler for software defined radio in handheld devices. *EURASIP Journal on applied signal processing*, 2005.