Tradeoff between Data-, Instruction-, and Thread-Level Parallelism in Stream Processors

Jung Ho Ahn
Hewlett-Packard Laboratories
Palo Alto, California

Mattan Erez
University of Texas at Austin
Austin, Texas

William J. Dally
Stanford University
Stanford, California [*]

ABSTRACT

This paper explores the scalability of the Stream Processor architecture along the instruction-, data-, and thread-level parallelism dimensions. We develop detailed VLSI-cost and processor-performance models for a multi-threaded Stream Processor and evaluate the tradeoffs, in both functionality and hardware costs, of mechanisms that exploit the different types of parallelism. We show that the hardware overhead of supporting coarse-grained independent threads of control is $15 - 86\%$ depending on machine parameters. We also demonstrate that the performance gains provided are of a smaller magnitude for a set of numerical applications. We argue that for stream applications with scalable parallel algorithms the performance is not very sensitive to the control structures used within a large range of area-efficient architectural choices. We evaluate the specific effects on performance of scaling along the different parallelism dimensions and explain the limitations of the ILP, DLP, and TLP hardware mechanisms.

Categories and Subject Descriptors

C.1.3 [**Processor Architectures**]: Other Architecture Styles

General Terms

Performance

Keywords

Stream Processors, Aspect Ratio, DLP, ILP, TLP

1. INTRODUCTION

The increasing importance of numerical applications and the properties of modern VLSI processes have led to a resurgence in the development of architectures with a large number of ALUs and extensive support for parallelism (e.g., [7, 17, 19, 21, 23, 28, 32, 37]). In particular, *Stream Processors* achieve area- and energy-efficient high performance by relying on the abundant parallelism, multiple levels of locality, and predictability of data accesses common to media, signal processing, and scientific application domains. The term *Stream Processor* refers to the architectural style exemplified by Imagine [17], the ClearSpeed CSX600 [6], Merrimac [7], and the Cell Broadband Engine (Cell) [28].

Stream Processors are optimized for the *stream execution model* [16] and not for programs that feature only instruction-level parallelism or that rely on fine-grained interacting threads. Stream Processors achieve high efficiency and performance by providing a large number of ALUs partitioned into processing elements (*PEs*), minimizing the amount of hardware dedicated to data-dependent control, and exposing a deep storage hierarchy that is tuned for throughput. Software explicitly expresses multiple levels of parallelism and locality and is responsible for both concurrent execution and latency hiding [11].

In this paper we extend the stream architecture of Merrimac to support thread-level parallelism (*TLP*) on top of the mechanisms for data-level and instruction-level parallelism (*DLP* and *ILP* respectively). Previous work on media applications has shown that Stream Processors can scale to a large number of ALUs without employing TLP. However, we show that exploiting the TLP dimension can reduce hardware costs when very large numbers of ALUs are provided and can lead to performance improvements when more complex and irregular algorithms are employed.

In our architecture, ILP is used to drive multiple functional units within each PE and to tolerate pipeline latencies. We use *VLIW* instructions that enable a highly area- and energy-efficient register file organization [30], but a scalar instruction set is also possible as in [28]. DLP is used to achieve high utilization of the throughput-oriented memory system and to feed a large number of PEs operated in *SIMD* fashion. To enable concurrent threads, multiple instruction sequencers are introduced, each controlling a group of PEs in *MIMD* fashion. Thus, we support a *coarse-grained independent threads of control execution model.* Each instruction sequencer supplies SIMD-VLIW instructions to its group of PEs, yielding a fully flexible MIMD-SIMD-VLIW chip architecture. More details are given in Sec. 3.

Instead of focusing on a specific design point we explore the scaling of the architecture along the three axes of parallelism as the number of ALUs is increased and use detailed models to measure hardware cost and application performance. We then discuss and simulate the tradeoffs between the added flexibility of multiple control threads, the overhead of synchronization and load balancing between these threads, and features such as fine grained communication between PEs that are only feasible with lockstep execution. Note that our stream architecture does not support multiple execution models as described in [21, 23,

[*]This work was supported, in part, by the Department of Energy ASC Alliances Program, Contract Lawrence Livermore National Laboratory B523583, with Stanford University.

32, 37]. The study presented here is a fair exploration of the cost-performance tradeoff and scaling for the three parallelism dimensions using a single execution model and algorithm for all benchmarks.

The main hardware cost of TLP is in additional instruction sequencers and storage. Contrary to our intuition, the cost analysis indicates that the area overhead of supporting TLP is in some cases quite low. Even when all PEs are operated in MIMD, and each has a dedicated instruction sequencer, the area overhead is only 15% compared to our single threaded baseline. When the architectural properties of the baseline are varied, the cost of TLP can be as high as 86% Therefore, we advocate a mix of ILP, DLP, and TLP that keeps hardware overheads to less than 5% compared to a minimum-cost organization. The performance evaluation also yielded interesting results in that the benefit of the added flexibility of exploiting TLP is hindered by increased synchronization costs. For the irregular applications we study, performance improvement of the threaded Stream Processor over a non-threaded version is limited to 7.4%, far less than the 30% figure calculated in [10]. We also observe that as the number of ALUs is scaled up beyond roughly 64 ALUs per chip, the cost of supporting communication between the ALUs grows sharply, suggesting a design point with VLIW in the range of 2–4 ALUs per PE, SIMD across PEs in groups of 8–16 PEs, and MIMD for scaling up to the desired number of ALUs.

The main contributions of this paper are as follows:

- We develop detailed processor-performance and hardware-cost models for a multi-threaded Stream Processor.
- We study, for the first time, the hardware tradeoff of scaling the number of ALUs along the ILP, DLP, and TLP dimensions combined.
- We evaluate the performance benefits of the additional flexibility allowed by multiple threads of control, and the performance overheads associated with the various hardware mechanisms that exploit parallelism.

In Sec. 2 we present prior work related to this paper. We describe our architecture and the ILP/DLP/TLP hardware tradeoff in Sec. 3. Sec. 4 discusses application properties and corresponding architectural choices. We develop the hardware cost model and analyze scaling in Sec. 5. Our experiments and performance evaluation appear in Sec. 6 and we conclude in Sec. 7. This work is described in more detail in Chapter 6 of [1].

2. RELATED WORK

In this section we describe prior work related to this paper. We focus on research that specifically explored scalability.

A large body of prior work addressed the scalability of general purpose architectures targeted at the sequential, single-threaded execution model. This research includes work on ILP architectures, such as [13, 26, 27], and speculative or fine-grained threading as in [12, 31, 33]. Similar studies were conducted for cache-coherent chip multi-processors targeting a more fine-grained cooperative threads execution model (e.g., [4]). In contrast, we examine a different architectural space in which parallelism is explicit in the programming model offering a widely different set of tradeoff options.

Recently, architectures that target several execution models have been developed. The RAW architecture [37] scales along the TLP dimension and provides hardware for software controlled low overhead communication between PEs. Smart Memories [23] and the Vector-Thread architecture [21] support both SIMD and MIMD control, but the scaling of ALUs follows the TLP axis. TRIPS [32] can repartition the ALU control based on the parallelism axis utilized best by an application, however, the hardware is fixed and scalability is essentially on the ILP axis within a PE and via coarse-grained TLP across PEs. The Sony Cell Broadband Engine$^{\text{TM}}$ processor (Cell) [28] is a flexible Stream Processor that can only be scaled utilizing TLP. A feature common to the work mentioned above is that there is little evaluation of the hardware costs and tradeoffs of scaling using a combination of ILP, DLP, and TLP, which is the focus of this paper.

The hardware and performance costs of scaling the DLP dimension were investigated in the case of the VIRAM architecture for media applications [20]. Media applications with regular control were also the focus of [18], which evaluated the scalability of Stream Processors along the DLP and ILP dimensions. We extend this work by incorporating TLP support into the architecture and evaluating the tradeoff options of ALU control organizations for both regular- and irregular-control applications.

3. STREAM ARCHITECTURE

In this section we give an overview of the stream execution model and stream architecture and focus on the mechanisms and tradeoffs of controlling many ALUs utilizing a combination of ILP, DLP, and TLP. A more thorough description of stream architectures and their merits appears in [7, 17].

3.1 Stream Architecture Overview

We first present the generalized stream execution model, and then a stream architecture that is designed to exploit it.

3.1.1 Stream Execution Model

The stream execution model targets compute-intensive numerical codes with large amounts of parallelism, structured control, and memory accesses that can be determined well in advance of data use. Stream programs may follow a restricted synchronous data flow representation [22, 35] or a generalized *gather–compute–scatter* form [5, 16]. We use the generalized stream model that has been shown to be a good match for media, signal processing, and physical modeling scientific computing domains [7, 17]. In the generalized form, coarse-grained, complex *kernel operations* are executed on collections of data elements, referred to as *streams* or *blocks*, that are transfered using asynchronous bulk operations. The generalized stream model is not restricted to sequentially processing all elements of the input data; instead, streaming is on a larger scale and refers to processing a sequence of blocks (streams). Relying on coarse-grained control and data transfer allows for less complex and more power- and area- efficient hardware, as delegating greater responsibility to software reduces the need to dynamically extract parallelism and locality.

3.1.2 Stream Processor Architecture

Fig. 1 depicts a canonical Stream Processor that consists of a simple general purpose control core, a throughput-oriented streaming memory system, on-chip local storage in the form of the *stream register file* (SRF), a set of ALUs with their associated *local register files* (LRFs), and instruction sequencers. This organization forms a hierarchy of bandwidth, locality, and control mitigating the detrimental effects of distance in modern VLSI processes, where bandwidth drops and latency and power

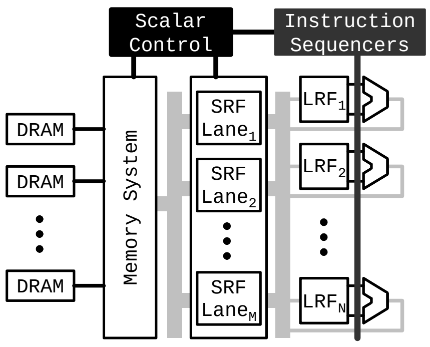


Figure 1: Canonical Stream Processor architecture

rise as distance grows.

The bandwidth hierarchy [30] consists of the DRAM interfaces that process off-chip communication, the SRF that serves as a staging area for the bulk transfers to and from DRAM and utilizes high bandwidth on-chip structures, and the LRFs that are highly partitioned and tightly connected to the ALUs in order to support their bandwidth demands. The same organization supports a hierarchy of locality. The LRFs exploits short term producer-consumer locality within kernels while the SRF targets producer-consumer locality between kernels and is able to capture a working set of the application. To facilitate effective utilization of the bandwidth/locality hierarchy each level has a separate name space. The LRFs are explicitly addressed as registers by ALU instructions. The SRF is arbitrarily addressable on-chip memory that is banked to support high bandwidth and may provide several addressing modes. The SRF is designed with a wide single ported SRAM for efficiency, and a set of *stream buffers* (SBs) are used to time-multiplex the SRF port [30]. The memory system maintains the global DRAM address space.

The general purpose core executes scalar instructions for overall program control and dispatches coarse-grained stream instructions to the memory system and instruction sequencer. Stream Processors deal with more coarsely grained instructions in their control flow than both superscalar and vector processors, significantly alleviating the von Neumann bottleneck [3]. The DMA engines in the memory system and the ALU instruction sequencers operate at a finer granularity and rely on decoupling for efficiency and high throughput. In a Stream Processor the ALUs can only access their private LRF and the SRF and rely on the higher level stream program to transfer data between memory and the SRF. Therefore, the SRF decouples the ALU pipeline from the unpredictable latencies of DRAM accesses enabling aggressive and effective static scheduling and a design with lower hardware cost compared to out-of-order architectures. Similarly, the memory system only handles DMA requests and can be optimized for throughput rather than latency.

In addition to the ALUs, a Stream Processor provides non-ALU functional units to support specialized arithmetic operations and to handle inter-PE communication. We assume that at least one *ITER* unit is available in each PE to accelerate iterative divide and square root operations; and that one or more *COMM* functional units are provided for inter-PE communication when applicable. The focus of this paper is on the organization and scaling of the ALUs along the ILP, DLP, and TLP axes of control as described below.

3.2 ALU Organization

We now describe how control of the ALUs can be structured along the different dimensions of parallelism and discuss the implications on synchronization and communication mechanisms.

3.2.1 DLP

A DLP organization of ALUs takes the form of a single instruction sequencer issuing SIMD instructions to a collection of ALUs. The ALUs within the group execute the same instructions in lockstep on different data. Unlike with vectors or wide-word arithmetic, each ALU can potentially access different SRF addresses. In this organization an optional switch can be introduced to connect the ALUs. Fig. 2(a) shows how all ALUs receive the same instruction (the “white” instruction) from a single instruction issue path and communicate on a global switch shared between the group of ALUs. Because the ALUs within the group operate in lockstep, simple control structures can utilize the switch for direct exchange of words between the ALUs, to implement the conditional streams mechanism for efficient data-dependent control operations [15], and to dynamically access SRF locations across several SRF banks [14].

Another possible implementation of DLP in a Stream Processor is to use short-vector ALUs that operate on wide words as with SSE [34], 3DNOW! [25], and Altivec [8]. This approach was taken in Imagine for 8-bit and 16-bit arithmetic within 32-bit words, on top of the flexible-addressing SIMD, to provide two levels of DLP. We chose not to evaluate this option in this paper because it significantly complicates programming and compilation and we explore the DLP dimension using clustering.

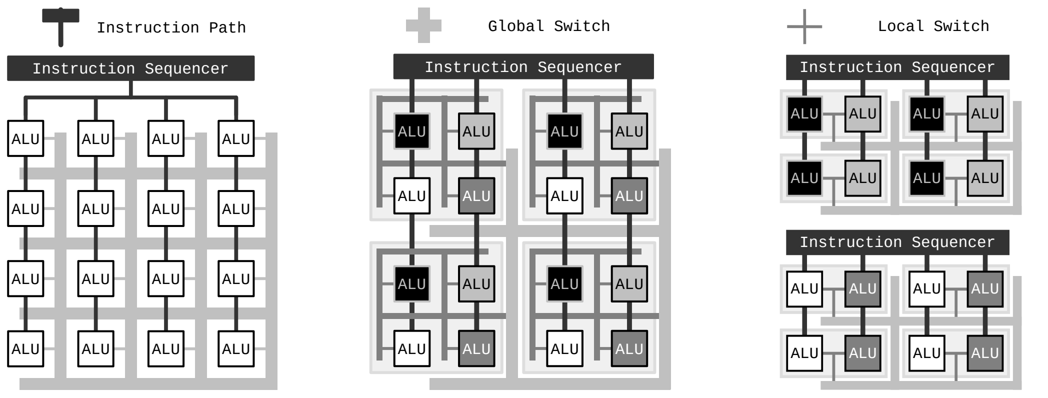
3.2.2 ILP

To introduce ILP into the DLP configuration the ALUs are partitioned into *clusters* (in this organization clusters correspond to PEs). Within a cluster each ALU receives a different operation from a VLIW instruction. DLP is used across clusters by using a single sequencer to issue SIMD instructions as before, but with each of these instructions being VLIW. Fig. 2(b) shows an organization with four clusters of four ALUs each. In each cluster a VLIW provides a “black”, a “dark gray”, a “light gray”, and a “white” instruction separately to each of the four ALUs. The same set of four instructions feeds the group of ALUs in all clusters (illustrated with shading in Fig. 2(b)).

In this SIMD/VLIW clustered organization the global switch described above becomes hierarchical. An *intra-cluster* switch connects the ALUs and LRFs within a cluster, and is statically scheduled by the VLIW compiler. The clusters are connected with an *inter-cluster* switch that is controlled in the same manner as the global DLP switch. This hierarchical organization provides an area-efficient and high bandwidth interconnect.

3.2.3 TLP

To address TLP we provide hardware MIMD support by adding multiple instruction sequencers and partitioning the inter-cluster switch. As shown in Fig. 2(c), each sequencer controls a *sequencer group* of clusters (in this example four clusters of two ALUs each). Within each cluster the ALUs share an intra-cluster switch, and within each sequencer group the clusters can have an optional inter-cluster switch. The inter-cluster switch can be extended across multiple sequencers, however, doing so requires costly synchronization mechanisms to ensure that a transfer across the switch is possible and that the data is coherent. A discussion of such mechanisms is beyond the



(a) SIMD (b) SIMD+VLIW (c) SIMD+VLIW+MIMD

Figure 2: ALU organization along the DLP, ILP, and TLP axes with SIMD, VLIW, and MIMD mechanisms respectively.

scope of this paper. It is possible to design an interconnect across multiple sequencer groups and partition it in software to allow a reconfigurable TLP option, and the hardware cost tradeoff of a full vs. a partitioned switch is discussed in Sec. 5. Another option is to communicate between sequencer groups using coarser-grained messages that amortize synchronization costs. For example, the Cell processor uses DMA commands to transfer data between its 8 sequencer groups (SPEs in Cell terminology) [28].

We do not evaluate extending a Stream Processor in the TLP dimension by adding virtual contexts that share the existing hardware as suggested for other architectures in [21, 36]. Unlike other architectures a Stream Processor relies heavily on explicitly expressing locality and latency hiding in software and exposes a deep bandwidth/locality hierarchy with hundreds of registers and megabytes of software controlled memory. Replicating such a large context is infeasible and partitioning the private LRFs reduces software's ability to express locality.

4. IMPACT OF APPLICATIONS ON ALU CONTROL

In order to understand the relationship between application properties and the mechanisms for ALU control and communication described in Sec. 3 we characterize numerically-intensive applications based on the following three criteria. First, whether the application is throughput-oriented or presents real-time constraints. Second, whether the parallelism in the application scales with the dataset or is fixed by the numerical algorithm. Third, whether the application requires *regular* or *irregular* flow within the numerical algorithm. Regular control corresponds to loops with statically determined bounds, and irregular control implies that the work performed is data dependent and can only be determined at runtime.

4.1 Throughput vs. Real-Time

In throughput-oriented applications, minimizing the time to solution is the most important criteria. For example, when multiplying large matrices, simulating a complex physical system as part of a scientific experiment, or in offline signal and image processing, the algorithms and software system can employ a batch processing style. In applications with real-time constraints, on the other hand, the usage model restricts the latency allowed for each sub-computation. In a gaming environment, for instance, the system must continuously respond to user input while performing video, audio, physics, and AI tasks.

The Imagine and Merrimac Stream Processors, as well as the similarly architected CSX-600, are designed for throughput-oriented applications. The ALUs in these processors are controlled with SIMD-VLIW instructions along the ILP and DLP dimensions only. The entire on-chip state of the processor, including the SRF (1MB/576KB in Merrimac/CSX600) and LRF (64KB/12KB), is explicitly managed by software and controlled with a single thread of execution. The software system exploits locality and parallelism in the application to utilize all processor resources. This organization works well for a large number of applications and scales to hundreds of ALUs [7,18]. However, supporting applications with real-time constraints can be challenging. Similarly to a conventional single-threaded processor, a software system may either preempt a running task due to an event that must be processed, or partition tasks into smaller subtasks that can be interleaved to ensure real-time goals are achieved.

Performing a preemptive context switch requires that the SRF be allocated to support the working set of both tasks. In addition, the register state must be saved and restored potentially requiring over 5% of SRF capacity and consuming hundreds of cycles due to the large number of registers. It is possible to partition the registers between multiple tasks to avoid this overhead at the expense of a smaller register space that can be used to exploit locality.

Subdividing tasks exposes another weakness of throughput-oriented designs. The aggressive static scheduling required to control the Stream Processor's ALUs often results in high task startup costs when software pipelined loops must be primed and drained. If the amount of work a task performs is small, because it was subdivided to ensure interactive constraints for example, these overheads adversely affect performance.

Introducing MIMD to support multiple concurrent threads of control can alleviate these problems, by allocating tasks spatially across the sequencer groups. In this way resources are dedicated to specific tasks leading to predictable timing. The Cell processor, which is used in the Sony PlayStation 3 gaming console, takes this approach. Using our terminology a Cell is organized as 8 sequencer groups with one cluster each, and one non-ALU functional unit and one FPU per cluster. The functional units operate on 128-bit wide words representing short vectors of four 32-bit operands each. As mentioned in Sec. 3, we do not directly evaluate vector-style SIMD in this paper and focus on an architecture where SIMD clusters can access independent SRF addresses.

4.2 Scaling of Parallelism

Many numerical algorithms and applications have parallelism that scales with the dataset size. Examples include n-body system simulation where interactions are calculated for each particle, processing of pixels in an image, images within a video stream, and points in a sampled signal. In such applications the parallelism can typically be cast into all three parallelism dimensions. Multiple threads can process separate partitions of the dataset, multiple clusters can work simultaneously on different elements, and loop unrolling and software pipelining transform data parallelism into ILP. The resulting performance depends on multiple factors such as the regularity of the control, load balancing issues, and inherent ILP as will be discussed in Sec. 6.

On the other hand, some numerical algorithms place a limit on the amount of parallelism, reducing scalability. For example, the deblocking filter algorithm of H.264 is limited in parallelism to computations within a 4×4 block because of a data dependency between blocks [29]. This leads to fine-grained serialization points of control, and the application may be more amenable to a space-multiplexed TLP mapping. The mapping of this type of limited-parallelism algorithm to highly parallel systems is a topic of active research and its evaluation is beyond the scope of this paper.

4.3 Control Regularity

All control decisions within a *regular control* portion of an algorithm can be determined statically. A typical case of regular control is a loop nest with statically known bounds. Regular control applications are very common and examples include convolution, FFT, dense matrix multiplication, and fixed degree meshes in scientific codes. Such algorithms can be easily mapped onto the DLP dimension and executed under SIMD-type control, and can also be cast into ILP and TLP. Converting DLP into TLP incurs overheads related to synchronization of the threads due to load imbalance. Casting DLP as ILP increases pipeline scheduling overheads related to software pipelining and loop unrolling, that are required to effectively utilize many ALUs. As a result, we expect that a high SIMD degree is necessary to achieve best performance on regular control applications.

In code with *irregular control*, decisions on control flow are made based on runtime information and cannot be determined statically. For example, when processing an element mesh the number of neighbors each element has may vary, leading to irregular control. Mapping irregular algorithms to the TLP dimension is simple as each sequencer follows its own control path. Mapping along the DLP dimension onto SIMD hardware is more challenging, and is discussed in detail in prior work [10, 15]. In [10] results based on an analytic performance model of a threaded stream architecture suggest that the potential of utilizing TLP can be as high as 30% over a DLP-ILP only configuration.

We evaluate the benefits and overheads of utilizing parallelism along the three axes for both regular and irregular applications in Sec. 6.

5. VLSI AREA MODELS

Our hardware cost model is based on an estimate of the area of each architectural component for the different organizations normalized to a single ALU. We focus on area rather than energy and delay for two reasons. First, as shown in [18], the energy of a stream processor is highly correlated to area. Second, a Stream Processor is designed to efficiently tolerate latencies by relying on software and the locality hierarchy, thereby reducing the sensitivity of performance to pipeline delays. Furthermore, as we will show below, near optimal configurations fall within a narrow range of parameters limiting the disparity in delay between desirable configurations. In this paper we only analyze the area of the structures relating to the ALUs and their control. The scalar core and memory system performance must scale with the number and throughput of the ALUs and do not depend on the ALU organization. Based on the implementation of Imagine [18] and the design of Merrimac we estimate that the memory system and the scalar core account for roughly 40% of a Stream Processor's die area.

5.1 Cost Model

The area model follows the methodology developed in [18] adapted to the design of the Merrimac processor [7,14] with the additional MIMD mechanism introduced in this paper. Tab. 1 and Tab. 2 summarize the parameters and equations used in the area cost model. The physical sizes are given in technology independent *track* and *grid* units, and are based on an ASIC flow. A *track* corresponds to the minimal distance between two metal wires at the lowest metal layer. A *grid* is the area of a (1×1) track block. Because modern VLSI designs tend to be wire limited, areas and spans measured in grids and tracks scale with VLSI fabrication technology.

We only briefly present the parameters and equations of the area model and a more detailed description can be found in [18].

The first parameter in Tab. 1 is the data width of the architecture, and we show results for both 64-bit and 32-bit arithmetic. The second set of parameters corresponds to the per-bit areas of storage elements, where the G_{SRF} parameter accounts for the multiplexers and decoders necessary for the SRF structures. The third group relates to the datapath of the Stream Processor and gives the widths and heights of the functional units and LRF.

Because the stream architecture is strongly partitioned, we only need to calculate the area of a given sequencer group to draw conclusions on scalability, as the total area is simply the product of the area of a sequencer group and the number of sequencers.

The main components of a sequencer group are the sequencer itself, the SRF, the clusters of functional units, LRF, and intra-cluster switch. A sequencer may also contain an inter-cluster switch to interconnect the clusters.

The sequencer includes the SRAM for storing the instructions and a datapath with a simple and narrow ALU (similar in area to a non-ALU numerical functional unit) and registers. The instruction distribution network utilizes high metal layers and does not contribute to the area.

In our architecture, the SRF is banked into lanes such that each cluster contains a single lane of the SRF, as in Imagine and Merrimac. The capacity of the SRF per ALU is fixed (S_{SRF}), but the distribution of the SRF array depends on C and N. The SRF cost also includes the hardware of the stream buffers. The amount of buffering required depends on both the number of ALUs within a cluster, and the bandwidth supplied by the wide SRF port. The number of SBs is equal to the number of external ports in a cluster P_e. The number of ports

Parameter	Value	Description (unit)
b	64/32	Data width of the architecture (bits)
A_{SRAM}	16	Area of a single ported SRAM bit used for SRF and instruction store (grids)
A_{sb}	128	Area of a dual ported stream-buffer bit (grids)
G_{SRF}	0.18	Area overhead of SRF structures relative to SRAM bit
w_{ALU}	1754	Datapath width of a 64-bit ALU (tracks)
w_{nonALU}	350	Datapath width of non-ALU supporting functional unit (FU) (tracks)
w_{LRF}	281	Datapath width of 64-bit LRF per functional unit (tracks)
h	2800	Datapath height for 64-bit functional units and LRF (tracks)
G_{COMM}	0.25	COMM units required per ALU
G_{ITER}	0.5	ITER units required per ALU
G_{sb}	1	Capacity of a half stream buffer per ALU (words)
I_0	64	Minimal width of VLIW instructions for control (bits)
I_N	64	Additional width of VLIW instructions per ALU/FU (bits)
L_C	4	Initial number of cluster SBs
L_N	1	Additional SBs required per ALU
L_{AG}	8	Bandwidth of on-chip memory system (words/cycle)
S_{SRF}	2048	SRF capacity per ALU (words)
S_{SEQ}	2048	Instruction capacity per sequencer group (VLIW words)
C	—	Number of clusters
N	—	Number of ALUs per cluster
T	—	Number of sequencers

Table 1: Summary of VLSI area parameters (ASIC flow). *Tracks* are the distance between minimal pitch metal tracks; *grids* are the area of a (1×1) track block.

has a minimal value L_C required to support transfers between the SRF and the memory system and a term that scales with N. The capacity of each SB must be large enough to effectively time multiplex the single SRF port, which must support enough bandwidth to both feed the ALUs (scales as $G_{sb}N$) and saturate the memory system (scales as $2L_{AG}/C$). The SRF bandwidth required per ALU (G_{sb}) will play a role in the performance of the MATMUL application described in Sec. 6.

The intra-cluster switch that connects the ALUs within a cluster and the inter-cluster switch that connects clusters use two dimensional grid structures to minimize area, delay, and energy as illustrated in Fig. 2. We assume fully-connected switches, but sparser interconnects may also be employed. The intra-cluster switch area scales as N^2, but for small N the cluster area is dominated by the functional units and LRF, which scale with N but have a larger constant factor.

The inter-cluster switch can be used for direct communication between the clusters, and also allows the clusters within a sequencer group to share their SRF space using the cross-lane indexed SRF mechanism [14]. We use a full switch to interconnect the clusters within a sequencer group and its grid organization is dominated by the C^2 scaling term. However, the dependence on the actual area of a cluster, the SRF lane, the SBs, and the number of COMM units required (N_{COMM} scales linearly with N) plays an important role when C is smaller.

5.2 Cost Analysis

As the analysis presented above describes, the total area cost scales linearly with the degree of TLP, and is equal to the number of sequencer groups (T) multiplied by the area of a single sequencer group. Therefore, to understand the tradeoffs of ALU organization along the different parallelism axes we choose a specific number of threads, and then evaluate the DLP and ILP dimension using a heatmap that represents the area of different (C, N) design space points relative to the area-optimal point. Fig. 3 presents the tradeoff heatmap for the baseline configuration specified in Tab. 1 for both a 32-bit and a 64-bit datapath Stream Processor. The horizontal axis corresponds to the number of clusters in a sequencer group (C), the vertical axis to the number of ALUs within each cluster (N), and the shading represents the relative cost normalized to a single ALU of the optimal-area configuration. In the 64-bit case, the area optimal point is (8-DLP,4-ILP) requiring 1.48×10^7 grids per ALU accounting for all stream execution elements. This corresponds to 94.1mm^2 in a 90nm ASIC process with 64 ALUs in a (2-TLP,8-DLP,4-ILP) configuration. For a 32-bit datapath, the optimal point is at (16-DLP,4-ILP).

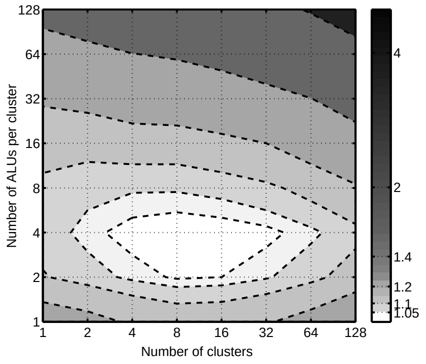
Fig. 3 indicates that the area dependence on the ILP dimension (number of ALUs per cluster) is much stronger than on DLP scaling. Configurations with an ILP of $2-4$ are roughly equivalent in terms of hardware cost, but further scaling along the ILP axis is not competitive because of the N^2 term of the intra-cluster switch and the increased instruction store capacity required to support wider VLIW instructions. Increasing the number of ALUs utilizing DLP leads to better scaling. With a 64-bit datapath (Fig. 3(a)), configurations in the range of $2-32$ and 4 ALUs are within about 5% of the optimal area. When scaling DLP beyond 32 clusters, the inter-cluster switch area significantly increases the area per ALU. A surprising observation is that even with no DLP, the area overhead of adding a sequencer for every cluster is only about 15% above optimal.

Both trends change when looking at a 32-bit datapath (Fig.3(b)). The cost of a 32-bit ALU is significantly lower, increasing the relative cost of the switches and sequencer. As a result only configurations within a $2-4$ ILP and $4-32$ DLP are competitive. Providing a sequencer to each cluster in a 32-bit architecture requires 62% more area than the optimal configuration.

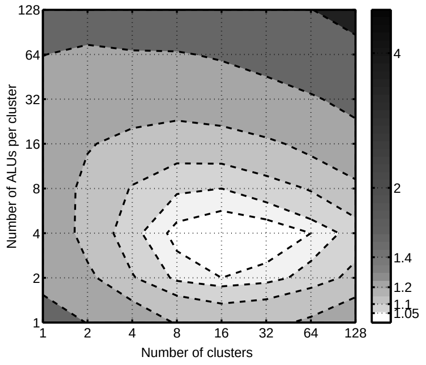
Scaling along the TLP dimension limits direct cluster to cluster communication to within a sequencer group, reducing the cost of the inter-cluster switch, which scales as C^2. The inter-cluster switch, however, is used for performance optimizations and is not necessary for the architecture because communica-

Component	Equation
COMMs per cluster	$N_{COMM} = (C = 1\ ?\ 0 : \lceil G_{COMM} N \rceil)$
ITERs per cluster	$N_{ITER} = \lceil G_{ITER} N \rceil$
FUs per cluster	$N_{FU} = N + N_{ITER} + N_{COMM}$
External Cluster Ports	$P_e = L_C + L_N N$
COMM bit width	$b_{COMM} = (b + log_2(S_{SRF} N C)) N_{COMM}$
Sequencer area	$A_{SEQ} = S_{SEQ}(I_0 + I_N N_{FU}) A_{SRAM} + h(w_{nonALU} + w_{LRF})$
SRF area per cluster	$A_{SRF} = (1 + G_{SRF}) S_{SRF} N A_{SRAM} b + 2A_{sb} b P_e max(G_{sb} N, L_{AG}/C)$
Intra-cluster switch area	$A_{SW} = N_{FU}(\sqrt{N_{FU}} b)(2\sqrt{N_{FU}} b + h + 2w_{ALU} + 2w_{LRF}) + \sqrt{N_{FU}}(3\sqrt{N_{FU}} b + h + w_{ALU} + w_{LRF}) P_e b$
Cluster area	$A_{CL} = N_{FU} w_{LRF} h + (N w_{ALU} + (N_{ITER} + N_{COMM}) w_{nonALU}) h + A_{SW}$
Inter-cluster switch area	$A_{COMM} = C b_{COMM} \sqrt{C} (b_{COMM} \sqrt{C} + 2\sqrt{A_{CL} + A_{SRF}})$
Total area	$A_{TOT} = T(C(A_{SRF} + A_{CL}) + A_{COMM} + A_{SEQ})$

Table 2: Summary of VLSI area cost models

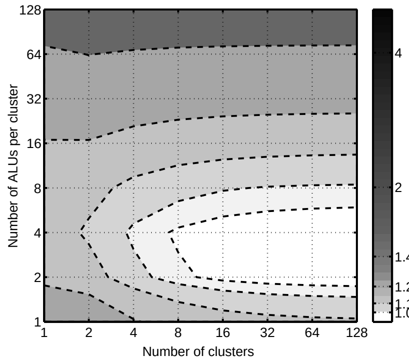


(a) 64-bit datapath

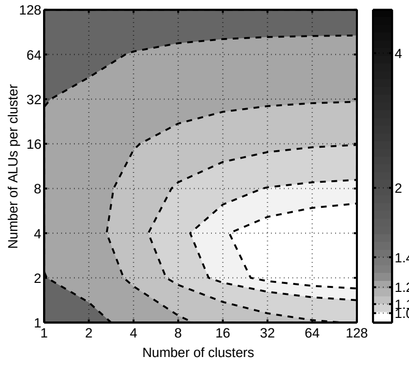


(b) 32-bit datapath

Figure 3: Relative area per ALU normalized to optimal ALU organization with the baseline configuration: (8-DLP,4-ILP) and (16-DLP,4-ILP) for 64-bit and 32-bit datapaths respectively.



(a) 64-bit datapath



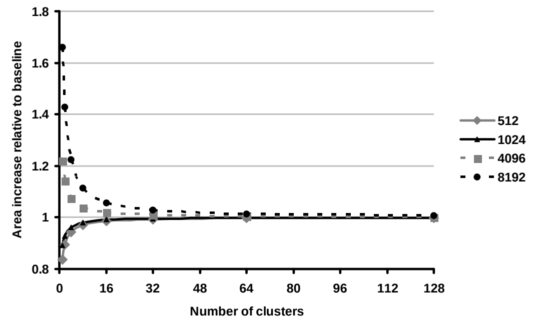
(b) 32-bit datapath

Figure 4: Relative area per ALU normalized to optimal ALU organization for configurations with no inter-cluster switch: (128-DLP,4-ILP).

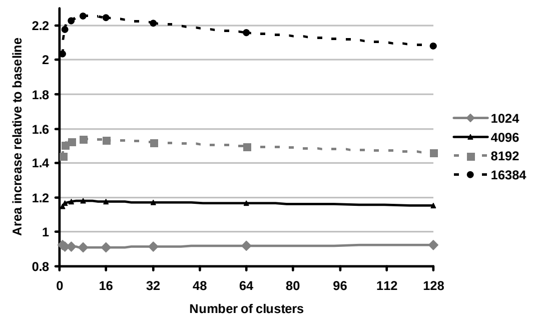
tion can always be performed through memory. Fig. 4 shows the area overhead heatmaps for configurations with no inter-cluster communication. The area per ALU without a switch improves with the amount of DLP utilized, but all configurations with more than 8 clusters fall within a narrow 5% area overhead range. The relative cost of adding sequencers is larger when no inter-cluster switch is provided because partitioning the control does not reduce the inter-cluster switch area. Thus, the single-cluster sequencer group configurations have an overhead of 27% and 86% for a 64-bit and a 32-bit datapath respectively. Note that the area-optimal configurations with no inter-cluster switch are 9% and 13% smaller than the area-optimal baseline configurations for 64-bit and 32-bit datapaths respectively.

We evaluate two extreme configurations of no inter-cluster communication and a fully-connected switch. Intermediate trade-off points include lower bandwidth and sparse interconnect structures, which will result in overhead profiles that are in between the heatmaps shown in Fig. 3 and Fig. 4.

Fig. 5 shows the sensitivity of the results presented above to the amount of on-chip storage in the sequencer instruction store (Fig. 5(a)) and SRF (Fig. 5(b)). The results are presented



(a) Sensitivity to instruction store capacity



(b) Sensitivity to SRF capacity

Figure 5: Relative increase of area per ALU as the number of clusters and capacity of storage structures is varied. Area is normalized to the baseline 64-bit configuration.

as a multiplicative factor relative to the baseline configuration shown in Fig. 3(a). In the case of instruction storage, there is a minor correlation between the area overhead and the degree of ILP due to VLIW word length changes. This correlation is not shown in the figure and is less than 1% for the near-optimal $2-4$-ILP configurations. The area overhead of increasing the instruction store is a constant because it only changes when sequencer groups are added along the TLP dimension. Therefore, the overhead decreases sharply for a larger than baseline instruction store, and quickly approaches the baseline. Similarly, when the instruction store is smaller than the baseline, the area advantage diminishes as the number of clusters increases. The trends are different for changing the amount of data storage in the SRF as the SRF capacity scales with the number of ALUs and, hence, the number of clusters. As a result, the area overhead for increasing the SRF size is simply the ratio of the added storage area relative to the ALU area for each configuration. The area per ALU has an optimal point near $8-16$ clusters, and the SRF area overhead is maximal.

6. EVALUATION

In this section we evaluate the performance tradeoffs of utilizing ILP, DLP, and TLP mechanisms using regular and irregular control applications that are all throughput-oriented and have parallelism on par with the dataset size. The applications are summarized in Tab. 3.

6.1 Experimental Setup

We modified the Merrimac cycle accurate simulator [7] to support multiple sequencer groups on a single chip. The memory system contains a single wide address generator to transfer data between the SRF and off-chip memory. We assume a 1GHz core clock frequency and memory timing and parameters conforming to XDR DRAM [9]. The memory system also includes a 256KB stream cache for bandwidth amplification, and scatter-add units, which are used for fast parallel reductions [1]. Our baseline configuration uses a 1MB SRF and 64 multiply-add floating point ALUs arranged in a (1-TLP,16-DLP,4-ILP) configuration along with 32 supporting iterative units, and allows for inter-cluster communication within a sequencer group at a rate of 1 word per cluster on each cycle.

The applications of Tab. 3 are written using a two-level programming model: kernels, which perform the computation referring to data resident in the SRF, are written in KernelC and are compiled by an aggressive VLIW kernel scheduler [24], and stream level programs, which control kernel invocation and bulk stream memory transfers, are coded using an API that is executed by the scalar control processor. The scalar core dispatches the stream instructions to the memory system and sequencer groups.

6.2 Performance Overview

The performance characteristics of the applications on the baseline configuration with the 1-TLP, 16-DLP, 4-ILP (1, 16, 4) configuration of Merrimac are reported in Tab. 4. Both CONV2D and MATMUL are regular control applications that are compute bound and achieve a high fraction of the 128 GFLOPS peak performance, at 85.8 and 117.3 GFLOPS respectively. FFT3D is also characterized by regular control and regular access patterns but places higher demands on the memory system. During the third FFT stage of computing the Z dimension of the data, the algorithm generates long strides (2^{14} words) that limit the throughput of DRAM and reduce the overall application performance. StreamFEM has regular control within a kernel but includes irregular constructs in the stream-level program. This irregularity limits the degree to which memory transfer latency can be tolerated with software control. StreamFEM also presents irregular data access patterns while processing the finite element mesh structure. This irregular access limits the performance of the memory system and leads to bursty throughput, but the application has high arithmetic intensity and achieves over half of peak performance. StreamMD has highly irregular control and data access as well as a numerically complex kernel with many divide and square root operations. It achieves a significant fraction of peak performance at 50.2 GFLOPS. Finally, the performance of StreamCDP is severely limited by the memory system. StreamCDP issues irregular access patterns to memory with little spatial locality, and also performs only a small number of arithmetic operations for every word transferred from memory (it has low arithmetic intensity). In the remainder of this section we will report run time results relative to this baseline performance.

Fig. 6 shows the run time results for our applications on 6 different ALU organizations. The run time is broken down into four categories: all sequencers are busy executing a kernel corresponding to compute bound portion of the application; at least one sequencer is busy and the memory system is busy, indicating load imbalance between the threads, but performance bound by memory throughput; all sequencers are idle and the memory system is busy during memory bound portions of the execution; at least one sequencer is busy and the memory system is idle due to load imbalance between the execution control threads.

Application	Type	Description
CONV2D	regular	2-dimensional 5×5 convolution on a 2048×2048 dataset [14].
MATMUL	regular	blocked 512×512 matrix matrix multiply.
FFT3D	regular	$128 \times 128 \times 128$ 3D complex FFT performed as three 1D FFTs.
StreamFEM	(ir)regular	streaming finite-element-method code for magnetohydrodynamics flow equations operating on a $9,664$ element mesh [10].
StreamMD	irregular	molecular dynamics simulation of a $4,114$ molecule water system [11].
StreamCDP	irregular	finite volume large eddy flow simulation of a $29,096$ element mesh [10].

Table 3: Applications used for performance evaluation.

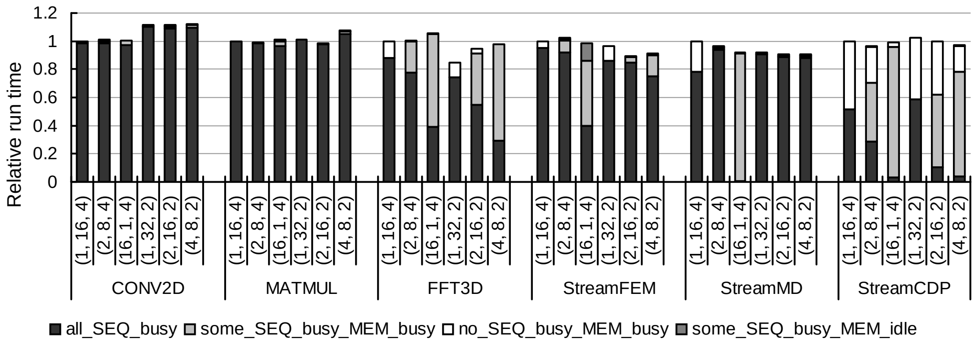


Figure 6: Relative run time normalized to the (1, 16, 4) configuration of the 6 applications on 6 ALU organizations.

Application	Computation	Memory BW
CONV2D	85.8 GFLOPS	31.1 GB/s
MATMUL	117.3 GFLOPS	15.6 GB/s
FFT3D	48.2 GFLOPS	44.2 GB/s
StreamFEM	65.5 GFLOPS	20.7 GB/s
StreamMD	50.2 GFLOPS	36.3 GB/s
StreamCDP	10.5 GFLOPS	39.1 GB/s
Peak	128 GFLOPS	64 GB/s

Table 4: Summary of application performance on baseline (1-TLP, 16-DLP, 4-ILP) configuration

The six configurations were chosen to allow us to understand and compare the effects of controlling the ALUs with different parallelism dimensions. The total number of ALUs was kept constant at 64 to allow direct performance comparison. The degree of ILP (ALUs per cluster) is chosen as either 2 or 4 as indicated by the analysis in Sec. 5, and the amount of TLP (number of sequencer groups) was varied from 1 to 16. The DLP degree was chosen to bring the total number of ALUs to 64.

Overall, we see that with the baseline configuration, the amount of ILP utilized is more critical to performance than the number of threads used. In FFT3D choosing a 2-ILP configuration improves performance by 15%, while the gain due to multiple threads peaks at 7.4% for StreamFEM.

We also note that in most cases, the differences in performance between the configurations are under 5%. After careful evaluation of the simulation results and timing we conclude that much of that difference is due to the sensitivity of the memory system to the presented access patterns [2]. Below we give a detailed analysis of the clear performance trends that are independent of the memory system fluctuations. We also evaluate the performance sensitivity to the SRF size and availability of inter-cluster communication.

6.3 ILP Scaling

Changing the degree of ILP (the number of ALUs per cluster) affects performance in two ways. First, the SRF capacity per cluster grows with the degree of ILP, and second, the VLIW kernel scheduler performs better when the number of ALUs it manages is smaller.

Because the total number of ALUs and SRF capacity is constant, a 2-ILP configuration has twice the number of clusters of a 4-ILP configuration. Therefore, a 2-ILP configuration has half the SRF capacity per cluster. The SRF size per cluster can influence the degree of locality that can be exploited and reduce performance. We see evidence of this effect in the run time of CONV2D. CONV2D processes the 2048×2048 input set in 2D blocks, and the size of the blocks affects performance. Each 2D block has a boundary that must be processed separately from the body of the block, and the ratio of boundary elements to body elements scales roughly as $1/N$ (where N is the row length of the block). The size of the block is determined by the SRF capacity in a cluster and the number of clusters. With a 128KB SRF, and accounting for double buffering and book-keeping data, the best performance we obtained for a 2-ILP configuration uses 16×16 blocks. With an ILP of 4 the blocks can be as large as 32×32 leading to an 11.5% performance advantage.

Looking at Fig. 7 we see that increasing the SRF size to 8MB reduces the performance difference to 1%, since the proportion of boundary elements decreases and it is largely due to imperfect overlap between kernels and memory transfers at the beginning and the end of the simulation.

The effectiveness of the VLIW kernel scheduler also plays an important role in choosing the degree of ILP. Our register organization follows the stream organization presented in [30], and uses a distributed VLIW register file with an LRF attached directly to each ALU port. As a result, the register file fragments when the number of ALUs is increased, reducing the effectiveness of the scheduler. Tab. 5 lists the software pipelined loop

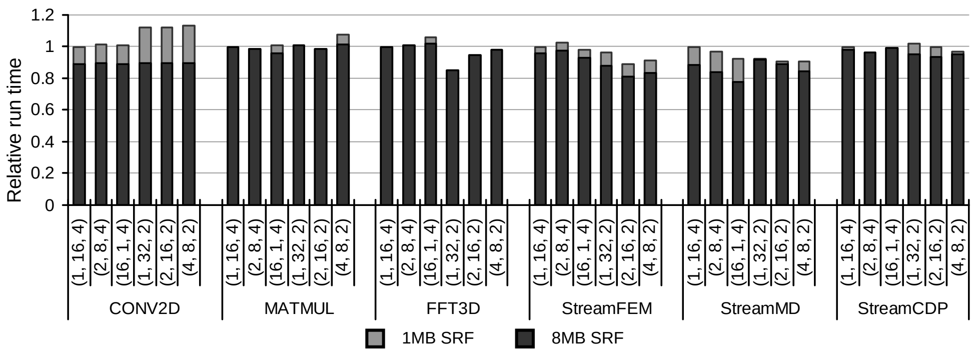


Figure 7: Relative run time normalized to the (1, 16, 4) configuration of the 6 applications comparing a 1MB and 8MB SRF.

initiation interval (II) achieved for 3 critical kernels. We can clearly see the scheduling advantage of the 2-ILP configuration as the II of the 2-ILP configurations is significantly lower than twice the II achieved with 4 ALUs. This leads to the 15% and 8.3% performance improvements observed for FFT3D and StreamMD respectively.

6.4 TLP Scaling

The addition of TLP to the Stream Processor has both beneficial and detrimental effects on performance. The benefits arise from performing irregular control for the applications that require it, and from the ability to mask transient memory system stalls with useful work. The second effect plays an important role in improving the performance of the irregular stream-program control of StreamFEM. The detrimental effects are due to the partitioning of the inter-cluster switch into sequencer groups, and the synchronization overhead resulting from load imbalance.

First, we note that the addition of TLP has no impact on CONV2D beyond memory system sensitivity. We expected synchronization overhead to reduce performance slightly, but because the application is regular and compute bound, the threads are well balanced and proceed at the same rate of execution. Therefore, the synchronization cost due to waiting on threads to reach the barrier is minimal and the effect is masked by memory system performance fluctuation.

The performance of MATMUL strongly depends on the size of the sub-matrices processed. For each sub-matrix, the computation requires $O(N_{sub}^3)$ floating point operations and $O(N_{sub}^2)$ words, where each sub-matrix is $N_{sub} \times N_{sub}$ elements. Therefore, the larger the blocks the more efficient the algorithm. With the baseline configuration, which supports direct inter-cluster communication, the sub-matrices can be partitioned across the entire SRF within each sequencer group. When the TLP degree is 1 or 2, the SRF can support 64×64 blocks, but only 32×32 blocks for a larger number of threads. In all our configurations, however, the memory system throughput was sufficient and the performance differences are due to a more subtle reason. The kernel computation of MATMUL requires a large number of SRF accesses in the VLIW schedule. When the sub-matrices are partitioned across the SRF, some references are serviced by the inter-cluster switch instead of the SRF. This can most clearly be seen in the (4,8,2) configuration, in which the blocks are smaller and therefore data is reused less requiring more accesses to the SRF. The same problem does not occur in the (16,1,4) configuration because it provides higher SRF bandwidth in order to support the memory system throughput (please refer to discussion of stream buffers in Sec. 3). The left side of Fig. 8 presents the run time results of MATMUL when the inter-cluster switch is removed, such that direct communication between clusters is not permitted, and for both a 1MB and a 8MB SRF. Removing the switch prevents sharing of the SRF and forces 16×16 sub-matrices with the smaller SRF size, with the exception of (16,1,4) that can still use a 32×32 block. As a result, performance degrades significantly. Even when SRF capacity is increased, performance does not match that of (16,1,4) because of the SRF bandwidth issue mentioned above. The group of bars with $Gsb = 2$ increases the SRF bandwidth by a factor of two for all configurations, and coupled with the larger SRF equalizes the run time of the applications if memory system sensitivity is ignored.

We see two effects of TLP scaling on the performance of FFT3D. First, FFT3D demonstrates the detrimental synchronization and load imbalance effect of adding threading support. While both the (1,16,4) and (1,32,2) configurations, which operate all clusters in lockstep, are memory bound, at least one sequencer group in all TLP-enabled configurations is busy at all times.

The stronger trend in FFT3D is that increasing the number of threads reduces the performance of the 2-ILP configurations. This is a result of the memory access pattern induced, which includes a very large stride when processing the Z dimension of the 3D FFT. Introducing more threads changes the blocking of the application and limits the number of consecutive words that are fetched from memory for each large stride. This effect has little to do with the control aspects of the ALUs and is specific to FFT3D.

StreamFEM demonstrates both the effectiveness of dynamically masking unexpectedly long memory latencies and the detrimental effect of load imbalance. The structure of the StreamFEM application limits the degree of double buffering that can be performed in order to tolerate memory latencies. As a result, computation and memory transfers are not perfectly overlapped, leading to performance degradation. We can see this most clearly in the (1, 32, 2) configuration of StreamFEM in Fig. 6, where there is a significant fraction of time when all clusters are idle waiting on the memory system (white portion of bar). When multiple threads are available this idle time in one thread is masked by execution in another thread, reducing the time to solution and improving performance by 7.4% for the (2, 16, 2). Increasing the number of threads further reduces performance by introducing load imbalance between the

Application	Kernel	Fraction of Run Time (1, 32, 2)	Fraction of Run Time (1, 16, 4)	Initiation Interval (1, 32, 2)	Initiation Interval (1, 16, 4)
FFT3D	FFT_128_stage1_to_stage3	20.7%	23.1%	28	16
	FFT_128_stage4_to_stage6	45.6%	47.2%	53	33
StreamMD	mole_DR_XL_COND	98.2%	98.7%	125	68

Table 5: Scheduler results of software-pipeline initiation interval for critical kernels.

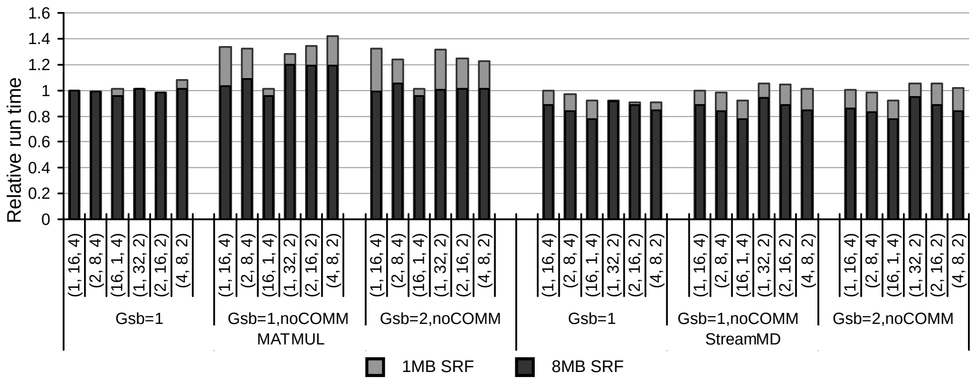


Figure 8: Relative run time normalized to the (1, 16, 4) configuration of MATMUL and StreamMD with and without inter-cluster communication.

threads. StreamFEM contains several barrier synchronization points, and load imbalance increases the synchronization time. This is most clearly seen in the extreme (16,1,4) configuration.

Based on the results presented in [10] we expect the benefit of TLP for irregular control to be significant (maximum 20% and 30% for StreamMD and StreamCDP respectively). Our results, however, show much lower performance improvements. In the baseline configuration, with 1MB SRF and inter-cluster communication, we see no advantage at all to utilizing TLP.

For StreamMD the reason relates to the effective inter-cluster communication enabled by SIMD. In the baseline configuration with 2-ILP, the algorithm utilizes the entire 1MB of the SRF for exploiting locality (cross-lane duplicate removal method of [10]) and is thus able to match the performance of the MIMD enabled organizations. As discussed earlier, the 4-ILP configurations perform poorly because of a less effective VLIW schedule. If we remove the inter-cluster switch for better area scaling (Sec. 5), the performance advantage of MIMD grows to a significant 10% (Fig. 8). Similarly, increasing the SRF size to 8MB provides enough state for exploiting locality within the SRF space of a single cluster and the performance advantage of utilizing TLP is again 10% (Fig. 7).

In the case of StreamCDP, the advantage of TLP is negated because the application is memory throughput bound. Even though computation efficiency is improved, the memory system throughput limits performance as is evident by the run time category corresponding to all sequencers being idle while the memory system is busy (white bars in Fig. 6).

7. CONCLUSION

We extend the stream architecture to support and scale along the three main dimensions of parallelism. VLIW instructions utilize ILP to drive multiple ALUs within a cluster, clusters are grouped under SIMD control of a single sequencer exploiting DLP, and multiple sequencer groups rely on TLP. We explore the scaling of the architecture along these dimensions as well as the tradeoffs between choosing different values of ILP, DLP, and TLP control for a given set of ALUs. Our methodology provides a fair comparison of the different parallelism techniques within the scope of applications with scalable parallelism, as the same execution model and basic implementation were used in all configurations.

We develop a detailed hardware cost model based on area normalized to a single ALU, and show that adding TLP support is beneficial as the number of ALUs on a processor scales above $32 - 64$. However, the cost of increasing the degree of TLP to an extreme of a single sequencer per cluster can be significant and ranges between $15 - 86\%$. The smaller part of this large overhead span is for configurations in which introducing sequencer groups partitions global switch structures on the chip, whereas high overhead is unavoidable in cases where the datapath is narrow (32-bits) and no inter-cluster communication is available.

Our performance evaluation shows that a wide range of numerical applications with scalable parallelism are fairly insensitive to the type of parallelism exploited. This is true for both regular and irregular control algorithms, and overall, the performance speedup is in the $0.9 - 1.15$ range. We provide detailed explanation to understand the many subtle sources of the performance difference and discuss the sensitivity of the results.

Continuous increases of transistor count force all the major parallelism types – DLP, ILP, and TLP – to be exploited in order to provide performance scalability as the number of ALUs per chip grows. The multi-threaded stream processor introduced in this paper is a step toward expanding the applicability and performance scalability of stream processing. The addition of hardware MIMD support allows efficient mapping of applications with a limited amount of fine-grain parallelism. In addition our cost models enable the comparison of stream processors with other chip-multiprocessors in both performance and efficiency.

8. REFERENCES

[1] J. Ahn, "Memory and Control Organizations of Stream Processors," Ph.D. dissertation, Stanford University, 2007.

[2] J. Ahn, M. Erez, and W. J. Dally, "The Design Space of Data-Parallel Memory Systems," in *SC'06*, Tampa, Florida, November 2006.

[3] J. Backus, "Can Programming be Liberated from the von Neumann Style?" *Communications of the ACM*, vol. 21, no. 8, pp. 613–641, August 1978.

[4] L. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese, "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing," in *Proceedings of the 27th International Symposium on Computer Architecture*, Vancouver, Canada, June 2000.

[5] 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.*, vol. 23, no. 3, pp. 777–786, 2004.

[6] ClearSpeed, "CSX600 datasheet," http://www.clearspeed.com/downloads/CSX600Processor.pdf, 2005.

[7] W. J. Dally, P. Hanrahan, M. Erez, T. J. Knight, F. Labonte, J. Ahn, N. Jayasena, U. J. Kapasi, A. Das, J. Gummaraju, and I. Buck, "Merrimac: Supercomputing with streams," in *SC'03*, Phoenix, Arizona, November 2003.

[8] K. Diefendorff, P. K. Dubey, R. Hochsprung, and H. Scales, "Altivec extension to PowerPC accelerates media processing," *IEEE Micro*, vol. 20, no. 2, pp. 85–95, 2000.

[9] ELPIDA Memory, Inc, "512M bits XDR^{TM} DRAM," http://www.elpida.com/pdfs/E0643E20.pdf.

[10] M. Erez, "Merrimac - High-Performance and High-Efficient Scientific Computing with Streams," Ph.D. dissertation, Stanford University, 2006.

[11] M. Erez, J. Ahn, A. Garg, W. J. Dally, and E. Darve, "Analysis and Performance Results of a Molecular Modeling Application on Merrimac," in *SC'04*, Pittsburgh, Pennsylvaniva, November 2004.

[12] L. Hammond, B. A. Hubbert, M. Siu, M. K. Prabhu, M. Chen, and K. Olukotun, "The Stanford Hydra CMP," *IEEE Micro*, vol. 20, no. 2, pp. 71–84, 2000.

[13] M. S. Hrishikesh, D. Burger, N. P. Jouppi, S. W. Keckler, K. I. Farkas, and P. Shivakumar, "The optimal logic depth per pipeline stage is 6 to 8 FO4 inverter delays," in *Proceedings of the 29th International Symposium on Computer Architecture*, Anchorage, Alaska, 2002, pp. 14–24.

[14] N. Jayasena, M. Erez, J. Ahn, and W. J. Dally, "Stream Register Files with Indexed Access," in *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, Madrid, Spain, February 2004.

[15] U. J. Kapasi, "Conditional Techniques for Stream Processing Kernels," Ph.D. dissertation, Stanford University, March 2004.

[16] U. J. Kapasi, S. Rixner, W. J. Dally, B. Khailany, J. Ahn, P. Mattson, and J. D. Owens, "Programmable Stream Processors," *IEEE Computer*, pp. 54–62, August 2003.

[17] B. Khailany, W. J. Dally, S. Rixner, U. J. Kapasi, P. Mattson, J. Namkoong, J. D. Owens, B. Towles, and A. Chang, "Imagine: Media Processing with Streams," *IEEE Micro*, pp. 35–46, March/April 2001.

[18] B. Khailany, W. J. Dally, S. Rixner, U. J. Kapasi, J. D. Owens, and B. Towles, "Exploring the VLSI Scalability of Stream Processors," in *Proceedings of the 9th Symposium on High Performance Computer Architecture*, Anaheim, California, February 2003.

[19] C. Kozyrakis, *et al.*, "Scalable Processors in the Billion-Transistor Era: IRAM," *Computer*, vol. 30, no. 9, pp. 75–78, 1997.

[20] C. Kozyrakis and D. Patterson, "Overcoming the Limitations of Conventional Vector Processors," in *Proceedings of the 30th International Symposium on Computer Architecture*, San Diego, California, June 2003, pp. 399–409.

[21] R. Krashinsky, C. Batten, M. Hampton, S. Gerding, B. Pharris, J. Casper, and K. Asanovic, "The Vector-Thread Architecture," in *Proceedings of the 31st International Symposium on Computer Architecture*, Munich, Germany, June 2004, pp. 52–63.

[22] E. A. Lee and D. G. Messerschmitt, "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing," *IEEE Transactions on Computers*, January 1987.

[23] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. Dally, and M. Horowitz, "Smart Memories: A Modular Recongurable Architecture," in *Proceedings of the 27th International Symposium on Computer Architecture*, Vancouver, BC, Canada, June 2000, pp. 161–171.

[24] P. Mattson, W. J. Dally, S. Rixner, U. J. Kapasi, and J. D. Owens, "Communication Scheduling," in *9th International Conference on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, November 2000, pp. 82–92.

[25] S. Oberman, G. Favor, and F. Weber, "AMD 3DNow! Technology: Architecture and Implementations," *IEEE Micro*, vol. 19, no. 2, pp. 37–48, 1999.

[26] S. Palacharla, N. P. Jouppi, and J. E. Smith, "Complexity-Effective Superscalar Processors," in *Proceedings of the 24th International Symposium on Computer Architecture*, Denver, Colorado, 1997, pp. 206–218.

[27] Y. N. Patt, S. J. Patel, M. Evers, D. H. Friendly, and J. Stark, "One Billion Transistors, One Uniprocessor, One Chip," *Computer*, vol. 30, no. 9, pp. 51–57, 1997.

[28] D. Pham, *et al.*, "The Design and Implementation of a First-Generation CELL Processor," in *Proceedings of the IEEE International Solid-State Circuits Conference*, San Francisco, CA, February 2005, pp. 184–185.

[29] I. E. Richardson, *H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia*. John Wiley & Sons, Ltd, 2003.

[30] S. Rixner, W. J. Dally, B. Khailany, P. Mattson, U. J. Kapasi, and J. D. Owens, "Register Organization for Media Processing," in *Proceedings of the 6th International Symposium on High Performance Computer Architecture*, Toulouse, France, January 2000, pp. 375–386.

[31] E. Rotenberg, Q. Jacobson, Y. Sazeides, and J. Smith, "Trace Processors," in *Proceedings of the 30th ACM/IEEE International Symposium on Microarchitecture*, North Carolina, 1997, pp. 138–148.

[32] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. W. Keckler, and C. R. Moore, "Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture," in *Proceedings of the 30th International Symposium on Computer Architecture*, San Diego, California, June 2003, pp. 422–433.

[33] G. S. Sohi, S. E. Breach, and T. N. Vijaykumar, "Multiscalar Processors," in *Proceedings of the 22nd International Symposium on Computer Architecture*, S. Margherita Ligure, Italy, 1995, pp. 414–425.

[34] S. T. Thakkar and T. Huff, "The Internet Streaming SIMD Extensions," *Intel Technology Journal*, no. Q2, p. 8, May 1999.

[35] W. Thies, M. Karczmarek, and S. P. Amarasinghe, "StreamIt: A Language for Streaming Applications," in *Proceedings of the 11th International Conference on Compiler Construction*, Grenoble, France, April 2002, pp. 179–196.

[36] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous multithreading: Maximizing on-chip parallelism," in *Proceedings of the 22th International Symposium on Computer Architecture*, June 1995, pp. 392–403.

[37] E. Waingold, *et al.*, "Baring it all to Software: Raw Machines," pp. 86–93, September 1997.