Register Organization for Media Processing

Scott Rixner[†]*, William J. Dally[†], Brucek Khailany[†], Peter Mattson[†], Ujval J. Kapasi[†], and John D. Owens[†]

[†]Computer Systems Laboratory
Stanford University
Stanford, CA 94305

*Dept. of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

{rixner, billd, khailany, pmattson, ujk, jowens}@cva.stanford.edu

Abstract

Processor architectures with tens to hundreds of arithmetic units are emerging to handle media processing applications. These applications, such as image coding, image synthesis, and image understanding, require arithmetic rates of up to 10^{11} operations per second. As the number of arithmetic units in a processor increases to meet these demands, register storage and communication between the arithmetic units dominate the area, delay, and power of the arithmetic units. In this paper we show that partitioning the register file along three axes reduces the cost of register storage and communication without significantly impacting performance. We develop a taxonomy of register architectures by partitioning across the data-parallel, instruction-level parallel, and memory hierarchy axes, and by optimizing the hierarchical register organization to operate on streams of data. Compared to a centralized global register file, the most compact of these organizations reduces the register file area, delay, and power dissipation of a media processor by factors of 195, 20, and 430, respectively. This reduction in cost is achieved with a performance degradation of only 8% on a representative set of media processing benchmarks.

1. Introduction

Media processing applications, such as video compression and decompression, image synthesis, and image understanding, demand very high arithmetic rates. To operate in real-time on large images, these applications currently demand 10^{10} to 10^{11} operations per second [2]. As applications become more sophisticated, even higher rates are expected. To achieve these high operation rates, processor architectures with tens to hundreds of arithmetic units are required. A 32-bit integer ALU takes less than 0.05mm^2 in a 0.18μm CMOS process, so we can economically fabricate several hundred arithmetic units on a relatively small 1cm^2 integrated-circuit chip. In the near future, we will be able to integrate thousands of arithmetic units on a chip.

As the number of arithmetic units in a media processor increases to these levels, register storage and communication between arithmetic units become critical factors dominating the area, cycle time, and power dissipation of the processor. The single central register file that has traditionally been used to interconnect ALUs and provide short-term storage does not scale well with large numbers of arithmetic units. For N arithmetic units, the area of the register file grows as N^3, the delay as $N^{3/2}$, and the power dissipation as N^3. The area of the register file dominates the area of the ALUs for more than 7 ALUs, the delay of the register file dominates the latency of a floating-point multiply for more than 58 ALUs, and the power dissipation of the register file dominates the power dissipation of a floating-point multiply for more than 8 ALUs.[1]

A central register file that interconnects every arithmetic unit is costly because it provides both storage for and communication among arithmetic units in a very general manner: any ALU can read from or write to any storage location. The area, delay, and power of the registers can be significantly reduced by restricting the communication between ALUs and registers, so that each arithmetic unit can only read and write a limited subset of registers. By restricting the communication in a manner that matches the parallelism and access patterns of the applications, a more efficient storage structure can be realized without a significant performance penalty. The most partitioned organization that we consider reduces the area, delay, and power dissipation of 48 ALUs by factors of 195, 20, and 430, respectively, while only degrading performance by 8% on a representative set of media processing kernels.

The research described in this paper was supported by the Defense Advanced Research Projects Agency under ARPA order E254 and monitored by the Army Intelligence Center under contract DABT63-96-C-0037.

1. This assumes that the register file is backed by a cache with a hit time of one cycle. If the register file must cover memory (or cache) latency of more than a few cycles, then the register file always dominates the area of the ALUs, dominates latency for more than 22 ALUs, and dominates power dissipation for more than 2 ALUs.

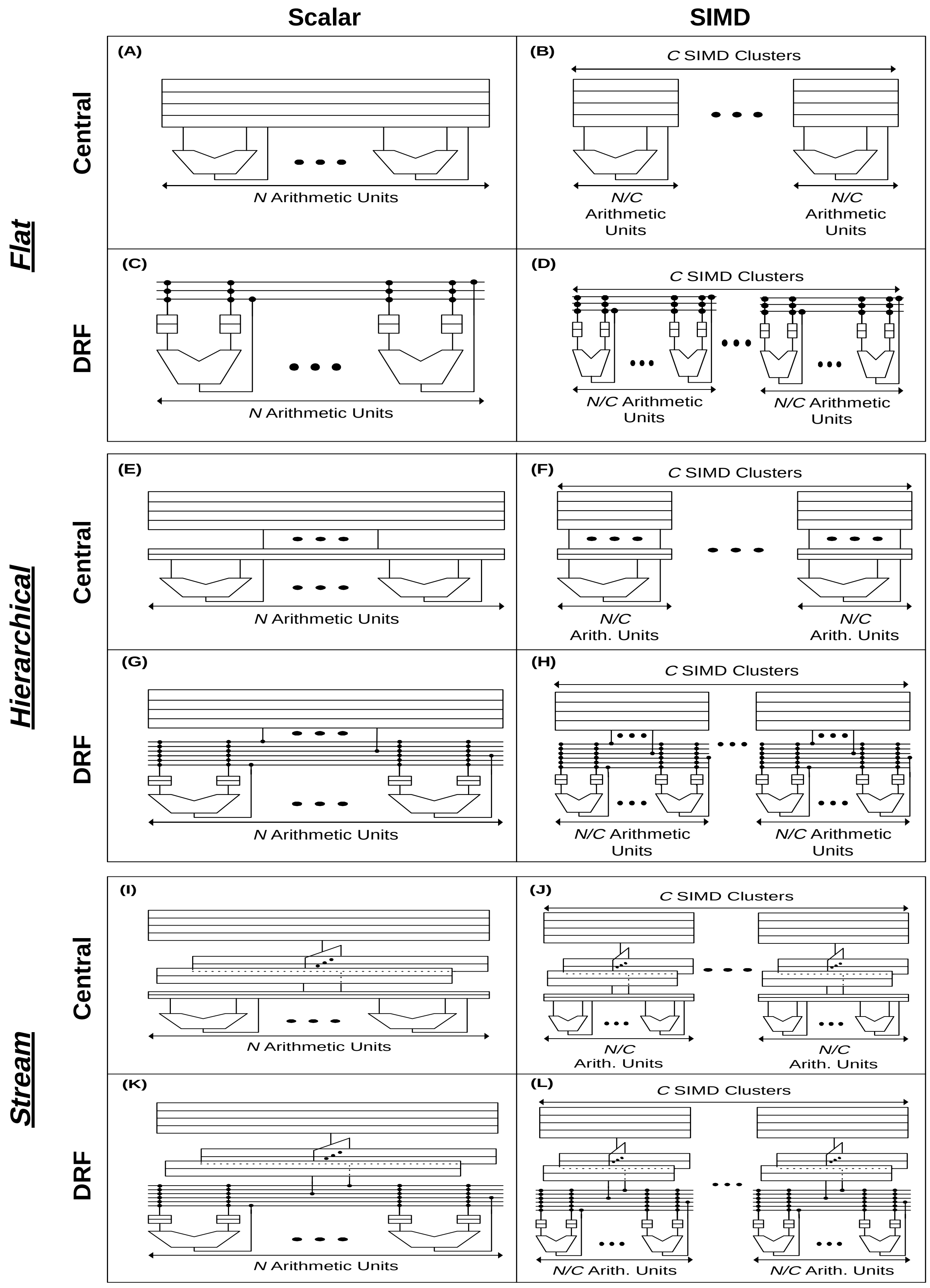

Figure 1. Register File Architecture Classifications.

Table 1. Summary of Parameters.

Parameter	Empirical Value	Description
α	0.25	Activity factor (probability that a node changes from 0 to 1 on a given cycle)
A_{FU}	7200*b*	Average area of a functional unit in grids
C_{bit}	0.22	Ratio of a register cell's bit line transistor capacitance to the capacitance of a minimum size inverter
C_w	0.05	Ratio of capacitance of one track of wire to the capacitance of a minimum size inverter
C_{word}	0.33	Ratio of a register cell's word select transistor capacitance to the capacitance of a minimum size inverter
E_0	12	Energy required to charge a minimum size inverter (in fJ)
f_{cyc}	1/20	Clock frequency (in 1/fan-out-of-four inverter delays): ~500MHz in a 0.18µm CMOS process
h	4	Register cell height (wire tracks) without ports
P_{FU}	25	Average power dissipated in an ALU when performing an arithmetic operation every cycle (in mW)
T	40	Memory latency in cycles
v_0	1350	Wire propagation velocity in tracks per fan-out-of-four inverter (FO4) delays [1]
w	3	Register cell width (wire tracks) without ports
b	32	Data width of the architecture
C	8	Number of SIMD clusters
G	1/4	Number of ports between the hierarchical register files per ALU
M	1/16	Number of memory ports per ALU
r_a	10	Number of registers per ALU
r_i	1.6	Load imbalance factor of a DRF architecture
r_m	4	Number of registers needed by each ALU for each cycle of memory latency
r_r	1.9	Replication factor of a DRF architecture
r_s	2	Number of registers needed in a stream buffer to hide the access latency of the memory staging register file
d	---	Length of a wire (distance)
N	---	Number of arithmetic units in a configuration
p	---	Total number of ports into a register file
p_e	*G* or *M*	Number of external ports connected to memory, a cache, or a memory staging register file per ALU
r	---	Number of registers per arithmetic unit
R	---	Total number of registers in a register file

In this paper, we investigate register organizations for media and signal processors with tens to hundreds of arithmetic units. We introduce a taxonomy of these register file organizations by partitioning the traditional central register file along three axes. Partitioning the register file along the data-parallel axis yields a SIMD (or vector) organization. Partitioning the register file along the instruction-level parallel axis yields a distributed register file (DRF) organization. Partitioning the register file along the memory hierarchy axis yields a hierarchical register organization. Finally, we optimize the hierarchical organization to handle long streams of continuous data resulting in a stream organization.

The remainder of this paper discusses register organization for media processors in more detail. Section 2 explains our analytical models of a register file's area, delay and power. Section 3 introduces twelve register organizations generated by applying the four partitions described above and compares their area, delay, and power. Section 4 examines the impact of these partitioned register organizations on media processing performance. The Appendix explains the models in more detail, and includes the complete set of formulae for the register organizations.

2. Register File Models

Figure 1 illustrates a taxonomy of 12 register organizations that starts with a central register file in the upper left corner and progresses through four transformations to a stream architecture in the lower right corner. The four transformations are: SIMD, dividing the register file across the data stream; DRF, dividing the register file across the functional units; Hierarchical, separating the portion of the register file that stages data for the ALUs from the portion that stages data for the memory; and Stream, organizing the memory portion of the register file to operate on long streams of continuous data.

In order to compare these 12 organizations, we introduce models for the area, delay, and power dissipation of a register file for large numbers of arithmetic units. Our complete models of area, delay, and power dissipation are presented in more detail in the Appendix. For reference, Table 1 provides a summary of the parameters used throughout the paper. The first section of values are for architectures targeted at a CMOS process with a minimum drawn gate length of 0.18µm. The second section of values were empirically determined using a variety of media processing kernels. The final section of values are parameters that are varied throughout the paper.

2.1. Area

The area of a register file is the product of the number of registers, R, the number of bits per register, b, and the size of a register cell. The schematic and layout of a register cell, given in Figure 2, show that each cell is $w+p$ wire tracks wide and $h+p$ wire tracks high, or $(w+p)(h+p)$ grids: p word lines in one dimension, p bit lines in the other, and $w \times h$ grids for the storage cell, power, and ground. Each port requires at least one wire track for a word line to address it and one wire track for a bit line to access the data. Configurations with a larger number of grids could be used for delay or power reasons, but would consume more area.

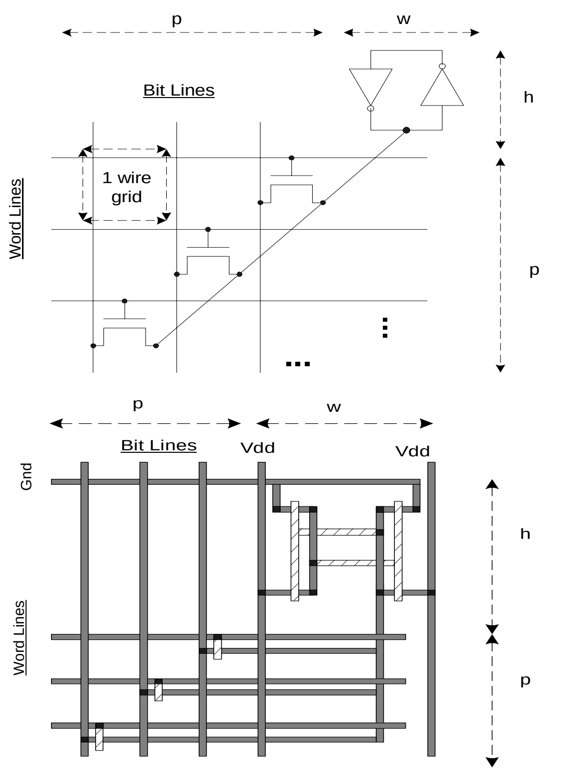

Figure 2. Schematic and layout of a register cell.

Several optimizations can reduce the area of a register file. For example, register cells can be replicated to distribute the read ports among several copies of each cell. However, all write ports must still access each copy in order to keep the data identical. For large numbers of read ports, duplicating the register file will reduce the total area, as in the Alpha 21264 [6]. Other area optimizations could also be applied to this basic register cell, such as time-multiplexing the ports, but these optimizations, like replication, merely decrease the register file size by a constant factor. Therefore, a register file with a large number of ports has area that grows with Rp^2.

A central register file organization (Figure 1a), with a single register file and N arithmetic units, requires r registers per ALU and $p_e + 3$ ports per ALU (two read ports for the operands, one write port for the result, and p_e external ports[2]). Thus, $R = rN$ and $p = (p_e+3)N$, so the area of the central register file will grow with N^3.

To reduce the number of ports into the central register file, it can be partitioned into interconnected clusters of arithmetic units, each with their own smaller centralized register file [3] [17]. The extreme case of this clustering is a distributed register file (DRF) organization where each

2. The external ports interface the register file with memory, a cache, or a larger hierarchical register file.

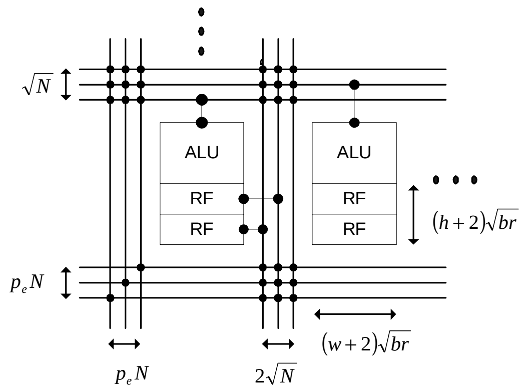

Figure 3. Configuration of the DRF organization.

arithmetic unit has a dedicated local register file connected directly to each input, as depicted in Figure 1c. In a DRF organization, each local register file has a single read port, a single write port, and contains $r/2$ registers. To connect the outputs of the N functional units, the $p_e N$ external ports, and the $2N$ distributed register files, a switch is needed. For more than a few ALUs, it is most efficient to organize the ALUs in two dimensions and utilize a two-level switch, as shown in Figure 3. Each row of ALUs requires $N^{1/2}$ b-bit buses to transfer their results to the columns and each column of ALUs requires $2N^{1/2}$ b-bit buses to transfer at most one data value to each of the distributed register files in that column. There must be an additional row and column of $p_e N$ buses to transfer data to and from the external ports.[3] The area of this switch dominates the area of the distributed register files, and grows with N^2. The switch in a DRF architecture need not provide a complete connection from every ALU to every register file, but the impact of an incomplete switch is beyond the scope of this paper.

2.2. Delay

The delay of a register file access is composed of *wire propagation delay* and *fan-in/fan-out delay.* The wire propagation delay is the minimum time of flight across a wire, which grows linearly with distance, assuming optimally spaced repeaters [1]. The fan-in/fan-out delay is the minimum drive delay of a lumped capacitive load using a buffer chain, which grows logarithmically with capacitance.

Wire propagation delay dominates the delay for long wires with little fan-out, as found in the word and bit lines of a register file with a large number of ports. To access a register cell, as shown in Figure 4, a signal must traverse a word line of length $(w+p)(bR)^{1/2}$ and then a bit line of length $(h+p)(bR)^{1/2}$, resulting in a delay that is propor-

3. The bidirectional external ports could be divided into read and write ports, splitting the $p_e N$ buses between the two dimensions.

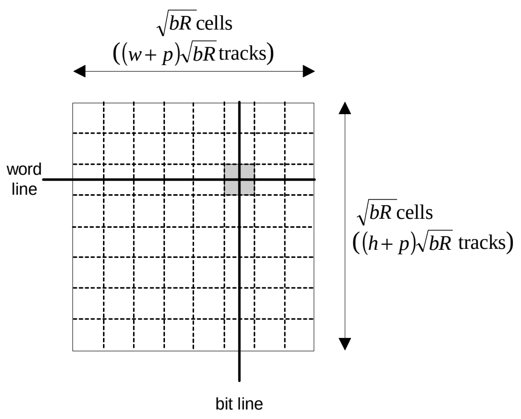

Figure 4. Register file access.

tional to $pR^{1/2}$. For register files with a small number of ports, the access time is dominated by the fan-out of the word lines and the fan-in of the bit lines, which is a function of the number of registers in the file, R. Bypassing the register file can reduce the delay of a register file access; however, the bypass structures are large and cannot eliminate the wire propagation delay between the arithmetic units.

The delay of the centralized register file is dominated by wire propagation delay when it must support more than 10 arithmetic units. A central register file organization has $R = rN$ registers and $p = (p_e+3)N$ ports, so beyond 10 arithmetic units, the delay grows with $N^{3/2}$. In a distributed register file organization, the two-port register files have a fixed size, $r/2$, so their delay is constant. The delay of traversing the DRF switch, however, includes the wire propagation delay of traversing both a horizontal and vertical wire in the switch, as can be seen in Figure 3. The delay of the switch is therefore linear in N and dominates the register file access time for more than 35 ALUs.

2.3. Power

The energy dissipated in a register file is proportional to the capacitance that must be switched for each access. Since every bit line and only a single word line must switch for each register file access, the power dissipation is dominated by the bit lines' capacitance. For a register file with a large number of ports, this capacitance is mostly wire capacitance. As shown in Figure 4, each port has $(bR)^{1/2}$ bit lines that connect $(bR)^{1/2}$ register cells, resulting in a wire capacitance proportional to $bR(h+p)C_w$. For a register file with a small number of ports, the bit line capacitance is mostly transistor capacitance, which is proportional to bRC_{bit}. Optimizations to reduce power dissipation, such as the use of hierarchical bit lines or differential bit lines with reduced-swing reads and full-swing writes, merely reduce it by a constant factor.

The number of ports in a central register file increases with the number of ALUs, N, by the relation $p = (p_e+3)N$. As can be seen in Figure 2, increasing the number of ports in a register file will increase the length of the wires in each port, while the number of transistors on those wires remains the same. Therefore, for register files with a large number of ports, the power dissipation of the p ports is dominated by wire capacitance and grows as p^2R. Since there are $R = rN$ registers in a central register file, the total power dissipation of the central register organization therefore grows as N^3. In a distributed register file organization, each two-port register file dissipates constant power, regardless of N. However, the power dissipation of the wires in the switch, which grows as N^2, dominates the power dissipation of the register files for more than 20 arithmetic units. Therefore, the power dissipation of the DRF organization grows quadratically with N.

3. Register Organizations

In this section we describe, quantify, and compare the 12 register organizations of Figure 1. Our analysis is further substantiated by the detailed models presented in the Appendix. Throughout this section, graphs are used to illustrate the important intuitions and trends of the area, delay and power dissipation of these register organizations.

3.1. Central Register Organization

The central register file architecture, shown in Figure 1a, serves as a baseline for our comparison. The basic model of a central register file organization was presented in Section 2. The number of registers in this organization grows linearly with the number of ALUs, N, by the relation $R = (r_a+r_mT)N$: each ALU requires r_a registers to stage its input and output operands and r_mT registers to cover the latency of memory, T. In order to provide sufficient data bandwidth to each ALU and enough additional bandwidth to transfer data back and forth to memory, the register file must have $p = (M+3)N$ ports: M ports to memory and 3 ports to source and sink data for each ALU. The area of a central register file is $A \approx Rp^2 \approx (r_a+r_mT)N((M+3)N)^2$, so the ratio of register area to ALU area grows as N^2. The delay of a central register file is dominated by wire propagation delay, leading to $t \approx pR^{1/2} \approx (M+3)N((r_a+r_mT)N)^{1/2}$. Therefore, the delay of the central register file grows as $N^{3/2}$. The power dissipation of the central register file is dominated by the wire capacitance of the bit lines, leading to $P \approx p^2R \approx ((M+3)N)^2(r_a+r_mT)N$, so the power overhead, $(NP_{FU}+P)/NP_{FU}$ (the total power dissipated in the register files, switches, and functional units divided by the power dissipated in the functional units), increases as N^2. Note

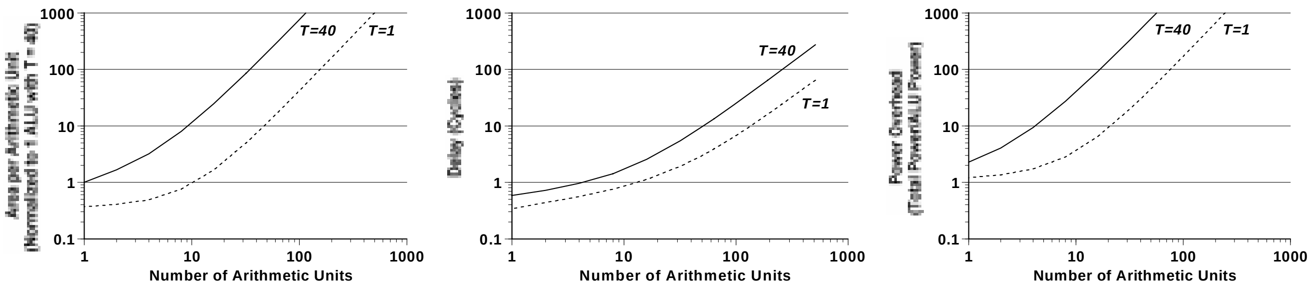

Figure 5. Area per ALU, delay, and power overhead of the central register file organization.

that increased power overhead corresponds to decreased power efficiency.

Figure 5 plots the area per ALU, register file delay, and power overhead of the central register organization for a memory system with latency $T = 1$ and for a memory system with latency $T = 40$. The low-latency case corresponds to a processor with a very effective on-chip cache memory, while the high-latency case reflects a processor without a cache or with a high miss ratio. Many media applications do not cache well, so the high-latency case is more indicative of the media processing application domain. The high-latency case increases the number of registers needed per ALU to hold the results of T outstanding memory operations per port. In both the high and low latency case, for more than about four ALUs, the area per ALU grows as N^2, the delay of the register file grows as $N^{3/2}$, and the power efficiency of the register file diminishes as N^2.

3.2. SIMD Register Organization

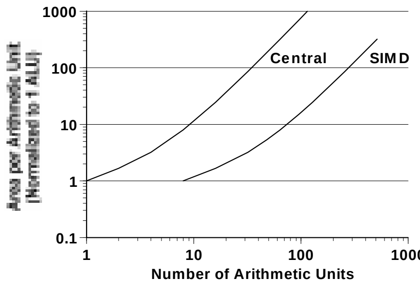

Figure 6. Area per ALU of the SIMD and Central organizations.

The cost of the register file can be substantially reduced for data-parallel media applications by partitioning the registers across groups of arithmetic units, as illustrated in Figure 1b. When a data-parallel loop is unrolled C times there is little or no communication between the iterations. Thus we can partition the register file into C identical clusters by removing the expensive, but little-used, communication paths between clusters.[4] This type of register organization is widely used in SIMD and vector processors [7] [9] [10] [16]. In a C-way SIMD register organization, there are C clusters, each with N/C ALUs, leading to an area per ALU ratio that grows as N^2/C^2.

Figure 6 compares the area per ALU for an eight-wide SIMD organization to that of our baseline organization. The figure shows that the SIMD organization starts out at eight ALUs with a register area that is only 8% of the central register organization. As the number of ALUs increases, the SIMD organization approaches an asymptote of 1.56% ($1/C^2 = 1/64$) of the area of the baseline organization. Delay and power dissipation are similarly improved asymptotically by constant factors to 4.44% ($(1/C)^{3/2} = (1/8)^{3/2}$) and 1.56%, respectively.

3.3. Distributed Register Organization

The register file can also be partitioned across the ALU inputs so that each ALU input has its own dedicated register file, as illustrated in Figure 1c. Each distributed register file only needs a single read port for its associated ALU input and a single write port to store results, decreasing the overall register area by a factor of N^2 from the central organization. However, an N x $2N$ crossbar switch must be introduced outside of the register files to connect each ALU to all of the two-port register files. This switch grows as N^2, so the register files and the switch combined grow linearly relative to ALU area. This type of organization can be found in the polycyclic architecture [13] and the Cydra [12].

The restricted communication of the DRF organization increases the number of registers required per arithmetic unit. Some data values must be replicated in multiple register files and all register files must provide enough storage to hold the peak number of live data values needed in any one register file. The increased register demand caused by replication and load imbalance can be approximated by the

4. A limited-bandwidth communication path separate from the register file is usually provided to allow for some communication between loop iterations. Such a *communication unit* greatly extends the range of applications that a SIMD architecture can handle without substantially increasing its area.

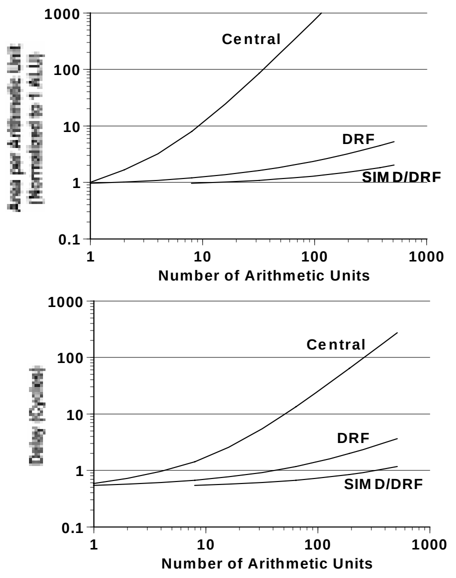

Figure 7. Area per ALU and delay of the Central, DRF, and SIMD/DRF organizations.

constant factors $r_r = 1.9$ and $r_i = 1.6$, giving a combined increase in register demand of 3.04.[5] However, because the switch is much larger than the register files in a DRF organization, the additional registers only increase the total area of a DRF organization with eight ALUs by 2.5%.

The SIMD organization of Figure 1b exploits register locality associated with data parallelism, and the DRF organization of Figure 1c exploits register locality associated with instruction-level parallelism. Combining these two orthogonal partitions gives the SIMD/DRF organization of Figure 1d. Figure 7 compares the area of the DRF and SIMD/DRF configurations to that of our baseline architecture. The figure illustrates that the area per ALU of the DRF organization grows linearly with the number of ALUs, because the switch area dominates the register files. The delay of the small two-port local register files is a small constant and the delay of the switch grows as N, whereas the delay of the large, many-ported central register file grows as $N^{3/2}$. The power overhead per ALU of the distributed register files increases linearly with N, compared to N^2 for the central register file organization. In the SIMD/DRF organization, the size of the DRF switch is decreased, thereby improving the area, delay, and power dissipation of the switch over the DRF organization by an amount that asymptotically approaches $1/C$.

5. These factors were empirically determined by measuring the difference in register demand between a central and DRF architecture. We expect that we could largely eliminate register imbalance by adding a heuristic to our compiler to balance register usage. However, we have not yet tested this conjecture.

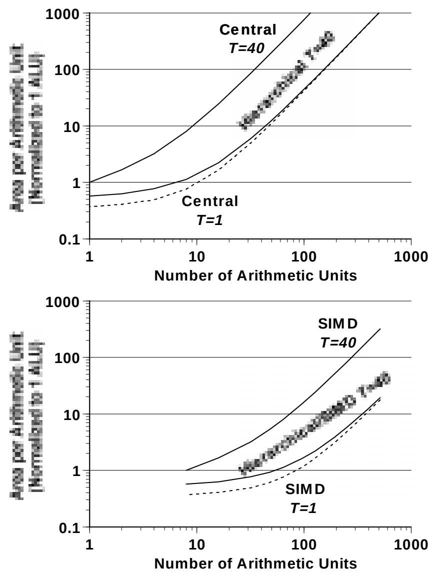

Figure 8. Area per ALU of the hierarchical organizations.

3.4. Hierarchical Register Organization

Registers provide short term storage to hold the results of both arithmetic operations and memory operations. As memory latency, T, expressed in CPU cycles, increases, the demand for register capacity, R, is primarily driven by the requirement to stage the results of large numbers of load operations. The number of registers per ALU required to handle memory loads increases linearly with T, by the relation $r_m T$, while the number needed to handle arithmetic operations remains constant at r_a.

A cache is used to cover the memory latency in conventional architectures, but many media applications do not have the temporal locality to successfully take advantage of a cache. Media applications exhibit little or no data reuse and have extremely large data sets, so a cache could potentially degrade performance rather than improve it [8]. Even though media processing data types do not cache well, using the cache as a staging area for prefetched data can improve performance [11]. However, this is an inefficient way to provide storage to stage memory results because address translation is required on every reference, accesses are made with long addresses, tag overhead is incurred in the cache, and conflicts may evict previously fetched data. It is more efficient to provide sufficient registers to cover the memory latency by prefetching data directly into registers rather than into the cache.

The hierarchical register organization, Figure 1e, partitions the central register file into a large register file with just a few ports to stage memory operations and a smaller register file with many ports to interconnect the arithmetic units. The port ratio is related to the ratio of arithmetic to memory instructions. This type of register organization can be found in the S and T registers of Cray vector machines [15]. This organization reduces the number of registers in the register file connected to the arithmetic units by $r_m T$ per arithmetic unit, because the central register file, with area $A \approx Rp^2 \approx (r_a + r_m T)N((M+3)N)^2$, is split into two smaller files, with total area $A \approx r_a N(3N)^2 + r_m TN(MN)^2$. GN extra ports must be introduced into both files to allow data to pass between them, giving a final area of $A \approx r_a N((G+3)N)^2 + r_m TN((G+M)N)^2$. Even with the extra ports, the combined area of these two register files is substantially less than that of the centralized register file with both a large number of ports and a large number of registers, as shown by the high latency ($T = 40$) curves in Figure 8.

The hierarchical register organization can also be combined with the SIMD organization (Figure 1f), the DRF organization (Figure 1g), or both (Figure 1h). Figure 8 shows how a hierarchical register organization reduces register area compared to the central and SIMD architectures with a memory latency of $T=40$. The area, delay, and power dissipation of hierarchical central and hierarchical SIMD organizations with memory latency of 40 cycles approaches the area, delay, and power dissipation of the original organizations with memory latency of 1 cycle. In effect, partitioning out the memory registers masks the effect of memory latency on register area, delay, and power dissipation. As the DRF and SIMD/DRF organizations have all of their registers contained in small two-port register files, moving to a hierarchical organization actually increases the overall area of these configurations by introducing a large, many-ported register file that dominates the area. However, by taking advantage of the sequential stream accesses inherent in media applications and moving to a stream register organization, the area of these two configurations improves as well.

3.5. Stream Register Organizations

Most media processing applications operate on a small number of sequential streams of data records (e.g. pixels in a video stream). Our final transformation replaces each port into the memory staging register file that we introduced in the hierarchical organization with a *stream buffer.* A stream buffer automatically prefetches sequential data for its associated stream out of this register file, similar to stream buffers used to prefetch streams from memory [4]. All stream buffers share a single port into the memory staging register file, allowing that single physical port to act as many logical ports. Each stream buffer must have $r_s W$ registers, where r_s is the amount of buffering required to hide the latency of the single port and W is the access width of that port. Since the single port of the memory staging register file must provide the same bandwidth as all of the ports in the hierarchical organization, $W = (G+M)N$. Therefore, the $(G+M)N$ stream buffers that replace the ports into the memory staging register file, each with $r_s(G+M)N$ registers, have a total area that grows with N^2.

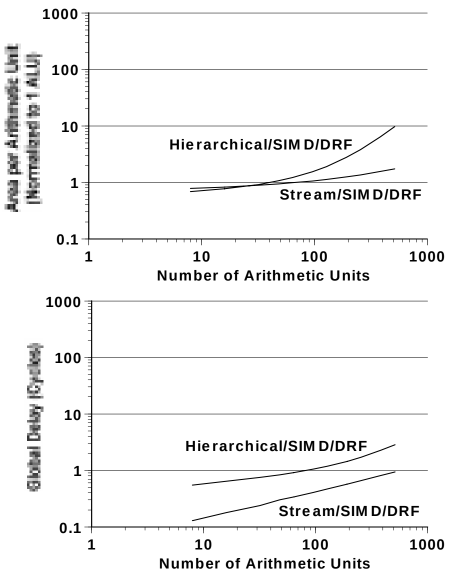

Figure 9. Area per ALU and global delay for hierarchical and stream SIMD/DRF organizations.

By replacing the $(G+M)N$ ports into the memory staging register file with a single port, the stream organization reduces the area of that register file by a factor of N^2, since the area of that register file grows as Rp^2. For very large N, the stream register file area is dominated by the stream buffers, which grow quadratically in N. In contrast, the hierarchical register file area grows cubically in N, because the number of ports into that register file grows with N. For the central (Figure 1i) and SIMD (Figure 1j) register organizations, the many ported register files that feed the arithmetic units dominate the area of the hierarchical organization, so the stream organization only improves their area slightly. For the DRF (Figure 1k) and SIMD/DRF (Figure 1l) register organizations, the memory staging register file dominates the area of the hierarchical organization, so the stream organization offers a significant savings in area, as shown in Figure 9. The Imagine stream processor [14], with a stream register file and SIMD/DRF arithmetic clusters, is an example of an architecture that combines all of these concepts.

The small stream buffers also hide the access time of the memory staging register file, as seen in Figure 9. The stream transformation decreases the memory staging register file access delay by a factor of $N^{1/2}$, but does not decrease the delay of the local register files connected to the ALUs.

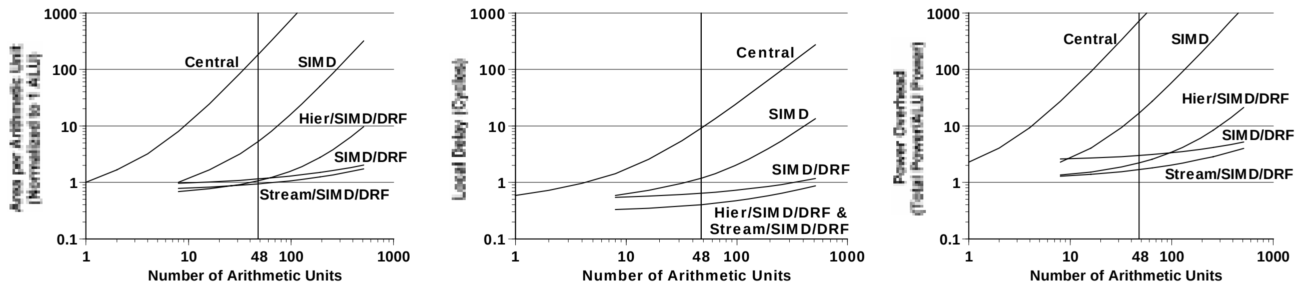

Figure 10. Area per ALU, delay, and power overhead of architectures spanning the space from a flat, scalar, central register file architecture to a stream, SIMD, distributed register file architecture.

4. Evaluation

To evaluate the various register architectures, we simulated the set of media processing kernels enumerated in Table 2 on a progression of architectures from the conventional central register file architecture to a stream architecture with SIMD clusters and distributed register files (Stream/SIMD/DRF). For the experiments, 48 arithmetic units were used in all of the configurations, and the rest of the parameter values were the empirically determined values from Table 1. All arithmetic units can perform 32-bit and dual 16-bit parallel-subword operations. The kernels were compiled for each organization using software pipelining and communication scheduling.

Table 2. Media Processing Kernels. Performance is given for a 48 arithmetic unit central architecture at 500MHz without accounting for additional register file latency.

Benchmark Kernel	Description	Performance (GOPS)
Convolution	Performs a 7x7 floating-point convolution over an image.	15.5
DCT (Discrete Cosine Transform)	Transforms an 8x8 matrix of 16-bit fixed-point numbers.	28.4
Triangle Transform	Performs a 3-D perspective transformation on a stream of triangles.	14.7
Pixel Shading	Shades a stream of pixels for a polygon rendering application.	15.8
FIR (Finite Impulse Response Filter)	Computes the response of a 13-tap 16-bit fixed-point FIR filter producing 1024 outputs.	22.4
FFT (Fast Fourier Transform)	Performs a 1024-point floating-point FFT.	10.0

Figure 10 shows the area per ALU, delay, and power dissipation of the five simulated organizations. There is a local maximum in performance per unit area at 48 ALUs, which is why that configuration was chosen. At 48 ALUs not all of the asymptotic effects have taken place, but the more partitioned organizations already provide significant area and power savings.

Figure 11 plots the performance of the media processing kernels on the five architectures. Figure 11a shows the raw performance of the kernels (normalized to their performance on the central organization given in Table 2) assuming that all of the organizations have the same register file latency. The kernels perform well across the architectures and show a performance degradation of only 16% on the stream architecture. The compiler can exploit enough data parallelism in most of the kernels that there is very little performance degradation. However, FIR and FFT require interactions between adjacent elements of the data stream, and therefore exhibit the largest performance degradation, most noticeably at the transition from a centralized to a SIMD organization.

Figure 11b shows the performance of the kernels when the additional latencies of the large register files are taken into account by increasing the pipeline depth of each arithmetic operation while holding the clock frequency constant. Performance is normalized to performance on the central organization without the additional latency. These media kernels have enough data parallelism that they are largely insensitive to operation latency, so modulo scheduling is able to hide the extra 8 cycles of latency of the large central register file. Registers are allocated when a result is produced, rather than when an operation is issued, so the increased operation latency does not increase register demand. Accounting for register file latency, the performance of the kernels on the stream organization is only 8% less than on the central organization.

Figure 11c shows the performance per unit area of the kernels on the four partitioned organizations normalized to their performance per unit area on the central organization. The performance per unit area of the stream organization is 180 times better than the centralized register file organization and 5 times better than the SIMD organization. At 48 ALUs, the stream organization is 430 times more power efficient than the central organization and 10 times more power efficient than the SIMD organization. Our metric of power efficiency is independent of kernel performance because it does not account for the additional power overhead of structures outside of the register file, arithmetic units, and switch.

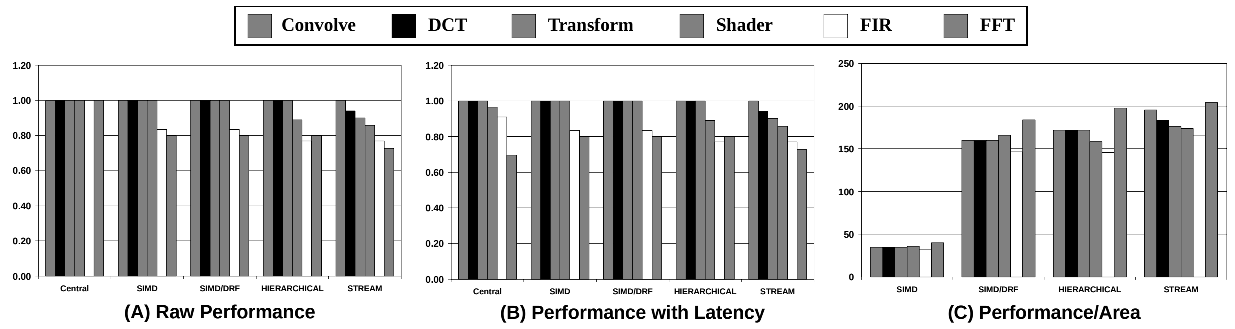

Figure 11. Performance of media processing kernels across five register organizations.

5. Conclusions

Partitioned register organizations dramatically reduce area, delay, and power dissipation in comparison to the traditional central register file organization, allowing them to scale efficiently to sustain the high arithmetic rates demanded by media processing applications. We present a taxonomy of partitioned register file architectures across three axes. Register files can be split along the data-parallel axis resulting in a SIMD organization, or along the instruction-level parallel axis resulting in a DRF organization. Register files can also be split between registers that stage memory operations and registers that provide short term storage for arithmetic units, resulting in a Hierarchical organization. Finally, restricting the access pattern of the hierarchical organization to streams of data results in a Stream organization.

We have developed models of the area, delay, and power dissipation of a register file organization to analyze the various partitioned register organizations. These models shows that the central register file organization has area and power dissipation that grow as N^3 and delay that grows as $N^{3/2}$. By separating data-parallel computations, the C-way SIMD organization reduces area, delay, and power dissipation by $1/C^2$, $1/C^{3/2}$, and $1/C^2$, respectively. By separating inter-ALU communication from data storage, the DRF organization has area and power dissipation that grow with N^2 and delay that grows with N. The Hierarchical organization masks the effect of memory latency on register area, delay, and power dissipation.

We introduce the Stream organization which improves the Hierarchical organization by reducing the number of ports into the memory staging register file. Experiments on organizations with 48 arithmetic units show that the Stream organization offers an area, delay, and power savings of 195, 20, and 430 over the central register file organization that can be realized with a performance degradation of only 8%.

As wires come to dominate the area and performance of processors, efficient execution demands that the movement of data and instructions on a chip be explicitly controlled and optimized. The partitioned register file architectures described in this paper are a first step toward such *explicit communication architectures*. Future work must address issues of distributed control and compilation to exploit locality on such explicit communication architectures.

References

[1] Dally, William J. and Poulton, John W. *Digital Systems Engineering*, Cambridge University Press: New York, NY, 1998.

[2] Diefendorff, K., Sony's Emotionally Charged Chip: Killer floating-point "Emotion Engine" to power PlayStation 2000. *Microprocessor Report* (April 19, 1999), pp. 1, 6-11.

[3] Farkas, Keith I., et. al., The Multicluster Architecture: Reducing Cycle Time Through Partitioning. In *Proceedings of the International Symposium on Microarchitecture* (November 1997), pp. 149-159.

[4] Jouppi, Norman P., Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers. In *Proceedings of the International Symposium on Computer Architecture* (May 1990),pp. 364-373.

[5] Keckler, Stephen W., et. al., The MIT Multi-ALU Processor. *Hot Chips IX*, (August 1997) Stanford, CA, pp 1-8.

[6] Kessler, R. E., The Alpha 21264 Microprocessor. *IEEE Micro* (March-April 1999), pp. 24-36.

[7] Lee, Ruby B., Subword Parallelism with MAX-2. *IEEE Micro* (August 1996), pp. 51-59.

[8] Lee, Ruby B. and Smith, Michael D., Media Processing: A new design target. *IEEE Micro* (August 1996), pp. 6-9.

[9] Oed, Wilfried, Cray Y-MP C90: System features and early benchmark results. *Parallel Computing* (August 1992), pp. 947-954.

[10] Peleg, Alex and Weiser, Uri, MMX Technology Extension to the Intel Architecture. *IEEE Micro* (August 1996), pp. 42-50.

[11] Ranganathan, Parthasarathy, et. al., Performance of Image and Video Processing with General-Purpose Processors and Media ISA Extensions. In *Proceedings of the International Symposium on Computer Architecture* (May 1999), pp. 124-135.

[12] RAU, B. RAMAKRISHNA, ET. AL., The Cydra 5 Departmental Supercomputer: Design philosophies, decisions, and trade-offs. In *Computer* (January 1989), pp. 12-35.

[13] RAU, B. RAMAKRISHNA, GLAESER, CHRISTOPHER D., AND PICARD, RAYMOND L., Efficient Code Generation for Horizontal Architectures: Compiler techniques and architectural support. In *Proceedings of the International Symposium on Computer Architecture* (April 1982), pp. 131-139.

[14] RIXNER, SCOTT, ET. AL., A Bandwidth-Efficient Architecture for Media Processing. In *Proceedings of the International Symposium on Microarchitecture* (December 1998), pp. 3-13.

[15] RUSSELL, RICHARD M., The Cray-1 Computer System. In *Communications of the ACM* (January 1978), pp. 63-72.

[16] TREMBLAY, MARC, ET. AL., VIS Speeds New Media Processing. *IEEE Micro* (August 1996), pp. 10-20.

[17] TURLEY, JIM AND HAKKARAINEN, HARRI, TI's New 'C6x DSP Screams at 1,600 MIPS. *Microprocessor Report* (February 17, 1997), pp. 14-17.

Appendix

Table 3 gives the equations for the area, delay, and power dissipation of a register file used to derive the formulae for the 12 register file organizations introduced in this paper. These equations give a lower bound approached by actual register files. Table 4 presents the area, delay, and power models for the 12 register file organizations included in this study, based upon the equations in Table 3.

The area equations for the register files and switches given in Table 3 are explained in Section 2.1. The DRF switch area can be derived directly from Figure 3. In addition to the register cells themselves, the register file must have a decoder to drive the word lines, and, for large enough files, sense amplifiers to recover the bit line signals after they have traversed the cells in the register file. In practice, these structures comprise roughly 20% of the register file area. For example, in a multithreaded microprocessor [5], the storage cells comprise 81% of the area of a 7-ported, 64-bit, 75 entry register file and the storage cells comprise 78% of the area of a one port, 128 Kb RAM.

The total delay of a register file can be broken down into two components: wire propagation delay and fan-in/fan-out delay, as explained in Section 2.2. The wire propagation delay component of total delay is given by $t_w(d) = d/v_0$, where d is distance and v_0 is velocity. The fan-in/fan-out delay is the sum of the word-line and bit-line delays. For the word-line delay, the register file address bits must fan out to every cell in the register file, bR, where each cell has transistor gate capacitance, C_{word}, and wire capacitance, $(w+p)C_w$. In addition, the worst case wire delay for a word line is $(w+p)(bR)^{1/2}$. For the bit-line delay, bits must fan in from all register cells, R, where each cell has transistor capacitance, C_{bit}, and wire capacitance, $(h+p)C_w$. The

Table 3. Base Area, Delay, and Power Equations.

Element	Equation
Register Area	$A_R(p) = b(w+p)(h+p)$
Central Architecture Area	$A_C(N, r, p_e) = rN \cdot A_R((p_e+3)N) + A_{FU} \cdot N$
DRF Switch Area	$A_{SW}(N, r, p_e) = (p_e+2)(p_e+1)(bN)^2 + [(2p_e+3)\sqrt{A_{FU}} + ((p_e+2)w+(p_e+1)h)\sqrt{rb}]bN^{3/2}$
DRF Architecture Area	$A_{DRF}(N, r, p_e) = rN \cdot A_R(2) + A_{SW}(N, r, p_e) + A_{FU} \cdot N$
Wire Delay	$t_w(d) = d/v_0$
Word Line Delay	$t_{word}(R, p) = \log_4((C_{word} + (w+p)C_w)bR) + t_w((w+p)\sqrt{bR})$
Bit Line Delay	$t_{bit}(R, p) = \log_4((C_{bit} + (h+p)C_w)R) + t_w((h+p)\sqrt{bR})$
RF Access Time	$t_a(R, p) = t_{word}(R, p) + t_{bit}(R, p)$
Wire Power Dissipation	$P_w(d) = C_w E_0 d f_{cyc}$
Word Line Power Dissipation	$P_{word}(R, p) = (C_{word} + (w+p)C_w)E_0\sqrt{bR})f_{cyc}$
Bit Line Power Dissipation	$P_{bit}(R, p) = [(C_{bit} + (h+p)C_w)E_0\sqrt{bR}] \cdot \alpha\sqrt{bR}f_{cyc}$
RF Access Power Dissipation	$P_a(R, p) = P_{word}(R, p) + P_{bit}(R, p)$

wire delay is a function of the length of one bit line, $(h+p)(bR)^{1/2}$. These equations give a lower bound for a register file's access time. All delays in Table 3 are given in units of fan-out-of-four inverter (FO4) delays. An FO4 delay is less than 100ps for a modern 0.18µm process. A cycle time of 20 FO4 delays is assumed, which corresponds to a clock frequency, f_{cyc}, of greater than 500MHz.

Power dissipation at a node is given by $P=C_{node}V\Delta V\alpha f_{cyc}$. The activity factor, α, is the probability that a node changes from 0 to 1 on a given cycle, and is 1/4 for random data. E_0 is the energy required to charge a minimum sized inverter, $C_{min}V\Delta V$. Therefore, power dissipation at a node can be computed as $P=C_xE_0\ \alpha f_{cyc}$, where $C_x=C_{node}/C_{min}$. The power dissipation of driving a wire, $P_w(d)$, is directly proportional to the length of the wire, d. During a register-file access, the power dissipation per port in the word lines, $P_{word}\,(R,\ p)$, is due to switching the capacitance of $(bR)^{1/2}$ cells on one word line. The power dissipation per port in the bit lines, $P_{bit}\,(R,\ p)$, is due to switching the capacitance of $(bR)^{1/2}$ cells on $(bR)^{1/2}$ bit lines with probability α. These power equations do not differentiate between reads and writes of the register files.

Table 4. Area (A), Delay (t), and Power (P) of the 12 Register Organizations. The 3 letter subscripts denote the particular architecture: (F)lat/(H)ierarchical/(S)tream, (S)calar/(V)ector-SIMD, and (C)entral/(D)istributed. Additional subscripts in the delay and power equations denote global (G), stream buffer (SB), and local (L) delay and power.

Central (A) $A_{FSC}(N) = A_C(N, r_a + r_m T, M)$ $t_{FSC}(N) = t_a((r_a + r_m T)N, (M+3)N) + 2t_w(\sqrt{A_{FU}N})$ $P_{FSC}(N) = (M+3)N \cdot P_a((r_a + r_m T)N, (M+3)N) + 3Nb\alpha \cdot P_w(\sqrt{A_{FU}N})$	**SIMD (B)** $A_{FVC}(N) = C \cdot A_{FSC}(N/C)$ $t_{FVC}(N) = t_{FSC}(N/C)$ $P_{FVC}(N) = C \cdot P_{FSC}(N/C)$
DRF (C) $A_{FSD}(N) = A_{DRF}(N, r_r r_i r_a + r_m T, M)$ $t_{FSD}(N) = t_a((1/2)(r_r r_i r_a + r_m T), 2) + t_w(2\sqrt{A_{FSD}(N)})$ $P_{FSD}(N) = (r_r + 2)N \cdot P_a((1/2)(r_r r_i r_a + r_m T), 2) + (2M+3)Nb\alpha \cdot P_w(\sqrt{A_{FSD}(N)})$	**SIMD/DRF (D)** $A_{FVD}(N) = C \cdot A_{FSD}(N/C)$ $t_{FVD}(N) = t_{FSD}(N/C)$ $P_{FVD}(N) = C \cdot P_{FSD}(N/C)$
Hierarchical/Central (E) $A_{HSC}(N) = r_m TN \cdot A_R((M+G)N) + A_C(N, r_a, G)$ $t_{HSCG}(N) = t_a(r_m TN, (M+G)N)$ $t_{HSCL}(N) = t_a(r_a N, (G+3)N) + 2t_w(\sqrt{A_{FU}N})$ $P_{HSCG}(N) = (M+G)N \cdot P_a(r_m TN, (M+G)N)$ $P_{HSCL}(N) = (G+3)N \cdot P_a(r_a N, (G+3)N) + 3Nb\alpha \cdot P_w(\sqrt{A_{FU}N})$	**Hierarchical/SIMD (F)** $A_{HVC}(N) = C \cdot A_{HSC}(N/C)$ $t_{HVCG}(N) = t_{HSCG}(N/C)$ $t_{HVCL}(N) = t_{HSCL}(N/C)$ $P_{HVCG}(N) = C \cdot P_{HSCG}(N/C)$ $P_{HVCL}(N) = C \cdot P_{HSCL}(N/C)$
Hierarchical/DRF (G) $A_{HSD}(N) = r_m TN \cdot A_R((M+G)N) + A_{DRF}(N, r_r r_i r_a, G)$ $t_{HSDG}(N) = t_a(r_m TN, (M+G)N) + t_w(2\sqrt{A_{DRF}(N, r_r r_i r_a, G)})$ $t_{HSDL}(N) = t_a((1/2) r_r r_i r_a, 2) + t_w(2\sqrt{A_{DRF}(N, r_r r_i r_a, G)})$ $P_{HSDG}(N) = (M+G)N \cdot P_a(r_m TN, (M+G)N) + 2GNb\alpha \cdot P_w(\sqrt{A_{DRF}(N, r_r r_i r_a, G)})$ $P_{HSDL}(N) = (r_r + 2)N \cdot P_a((1/2) r_r r_i r_a, 2) + 3Nb\alpha \cdot P_w(\sqrt{A_{DRF}(N, r_r r_i r_a, G)})$	**Hierarchical/SIMD/DRF (H)** $A_{HVD}(N) = C \cdot A_{HSD}(N/C)$ $t_{HVDG}(N) = t_{HSDG}(N/C)$ $t_{HVDL}(N) = t_{HSDL}(N/C)$ $P_{HVDG}(N) = C \cdot P_{HSDG}(N/C)$ $P_{HVDL}(N) = C \cdot P_{HSDL}(N/C)$
Stream/Central (I) $A_{SSC}(N) = r_m TNA_R(1) + A_C(N, r_a, G) + (G+M)N(r_s(G+M)N \cdot A_R(2))$ $t_{SSCG}(N) = t_a(r_m TN, 1)$ $t_{SSCSB}(N) = t_a(r_s(G+M)N, 2)$ $t_{SSCL}(N) = t_{HSCL}(N)$ $P_{SSCG}(N) = P_a(r_m TN, 1)$ $P_{SSCSB}(N) = 2(G+M)N \cdot P_a(r_s(G+M)N, 2)$ $P_{SSCL}(N) = P_{HSCL}(N)$	**Stream/SIMD (J)** $A_{SVC}(N) = C \cdot A_{SSC}(N/C)$ $t_{SVCG}(N) = t_{SSCG}(N/C)$ $t_{SVCSB}(N) = t_{SSCSB}(N/C)$ $t_{SVCL}(N) = t_{HSCL}(N/C)$ $P_{SVCG}(N) = C \cdot P_{SSCG}(N/C)$ $P_{SVCSB}(N) = C \cdot P_{SSCSB}(N/C)$ $P_{SVCL}(N) = C \cdot P_{HSCL}(N/C)$
Stream/DRF (K) $A_{SSD}(N) = r_m TNA_R(1) + A_{DRF}(N, r_r r_i r_a, G) + (G+M)N(r_s(G+M)N \cdot A_R(2))$ $t_{SSDG}(N) = t_{SSCG}(N)$ $t_{SSDSB}(N) = t_{SSCSB}(N) + t_w(2\sqrt{A_{DRF}(N, r_r r_i r_a, G)})$ $t_{SSDL}(N) = t_{HSDL}(N)$ $P_{SSDG}(N) = P_{SSCG}(N)$ $P_{SSDSB}(N) = P_{SSCSB}(N) + 2GNb\alpha \cdot P_w(\sqrt{A_{DRF}(N, r_r r_i r_a, G)})$ $P_{SSDL}(N) = P_{HSDL}(N)$	**Stream/SIMD/DRF (L)** $A_{SVD}(N) = C \cdot A_{SSD}(N/C)$ $t_{SVDG}(N) = t_{SSCG}(N/C)$ $t_{SVDSB}(N) = t_{SSDSB}(N/C)$ $t_{SVDL}(N) = t_{HSDL}(N/C)$ $P_{SVDG}(N) = C \cdot P_{SSCG}(N/C)$ $P_{SVDSB}(N) = C \cdot P_{SSDSB}(N/C)$ $P_{SVDL}(N) = C \cdot P_{HSDL}(N/C)$