Register Organization and Raw Hardware

Lecture #7: Thursday, 25 April 2002
Lecturer: Prof. Bill Dally
Scribe: Suzanne Rivoire, Yangjin Oh
Reviewer: Mattan Erez

Logistics :

- Handout : Project Topics (prepare for brainstorming on Tues)
- Project is in teams of up to 4
- Assignment step 2 due on Friday at midnight

During this lecture, we discussed two papers. The first paper, which constituted the majority of the lecture, is a comparison of the area requirements of different register file organizations. The second paper is an introduction to the hardware organization of MIT's Raw microprocessor.

1 Register Organization for Media Processing

S. Rixner, W. Dally, B. Khailany, P. Mattson, U. Kapasi, and J. Owens. "Register Organization for Media Processing." HPCA 6, 2000.

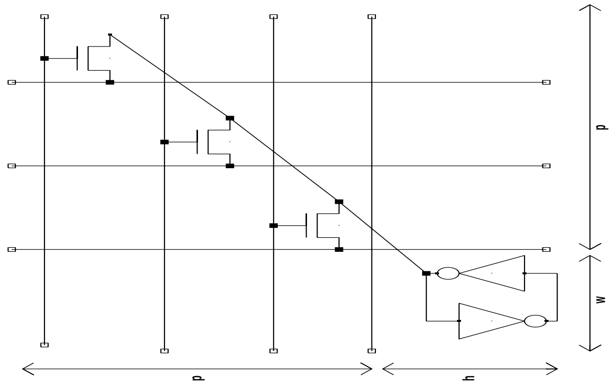

Figure 1: Schematic of register cell

Register files are commonly thought of as storage elements, but they also serve the purpose of communication. In fact, for multiport register files above a threshold size, the area of the communication switch dominates the area of the register file as a whole.

This paper recognizes the dual purpose of the register file and looks at ways to rearrange and decouple the storage and communication functionalities. Starting with a centrally organized flat register file, it describes transformations from this central organization into a single instruction multiple data (SIMD) organization, a distributed organization, and a distributed SIMD organization. Then it describes further transformations from the flat register file space to a hierarchical organization and finally a stream organization.

While the area, delay, and power dissipation of a register file are all important metrics for measuring performance, in this lecture we decided to focus on area for simplicity. The paper presents a variety of parameters for measuring area, described in detail on its last page. The area of a single register cell is proportional to $(p+w)(p+h)$, where p is the number of word lines in one dimension and bit lines in the other, and h and w are the height and width in wire tracks of the cell. A register file with a large number of ports has an area that grows with Rp^2, where R is the total number of registers and p is the total number of ports.

1.1 Central Register Organization

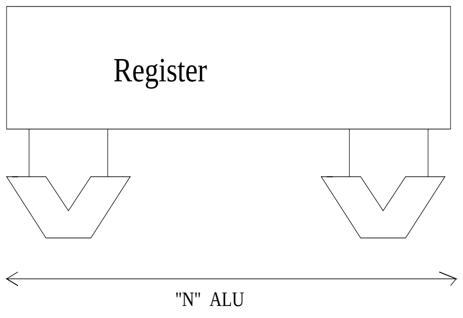

Figure 2: Central register organization

As a baseline, the paper first analyzes the central, flat register file architecture, shown in Figure 2. In this architecture, every ALU has two read ports and one write port to the register file.

The area of the central register file architecture is proportional to N^3, where N is the number of ALUs served by the register file.

The number of ports in a central register file is given by the equation $p = (3 + p_e)N$, where p_e is the number of external connections needed by the register file. Later in the paper, it is assumed that p_e can be effectively replaced by M, the number of connections to memory. The value of M is usually $\frac{1}{16}$, indicating that the register file needs three connections per ALU and $\frac{1}{16}$ that for memory. This further suggests that the ratio of local register file bandwidth to memory bandwidth is around 48:1. In Imagine, this ratio is actually much higher - around 250:1.

The number of registers in a central register file is given by $R = (r_a + T * r_m)N$, where r_m is the per-ALU number of memory references per cycle, T is the memory latency, and r_a is the number of registers per ALU, set equal to 10 here.

In this paper, r_m, an empirically determined constant, is set equal to 4. T, the memory latency, is set equal to 40. Figure 3 shows the meaning of r_m and T. As the memory latency grows, so does the number of outstanding requests to memory and so, therefore, does the number of registers to hold these requests. Additionally, the strip size is proportional to T, since we need longer strips as the memory latency gets longer.

In summary, the area of the central register file is proportional to N^3, since $p^2R = (3 + M)^2N(r_a + Tr_m)N^2$.

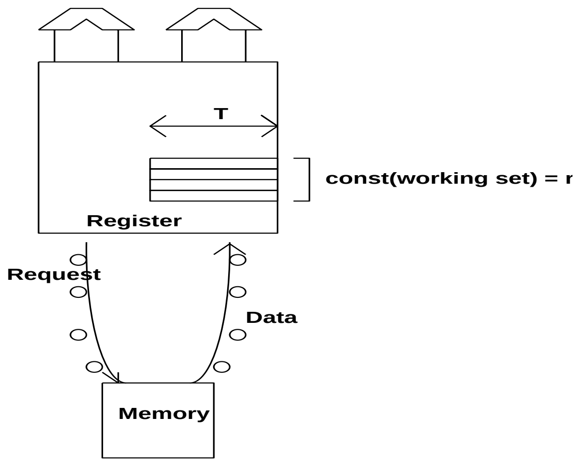

Figure 3: Relationship between T and r_m

1.2 SIMD Register Organization

The SIMD transformation splits the central register file into C separate clusters. The area needed by the SIMD register file is proportional to $(\frac{N}{C})^3$.

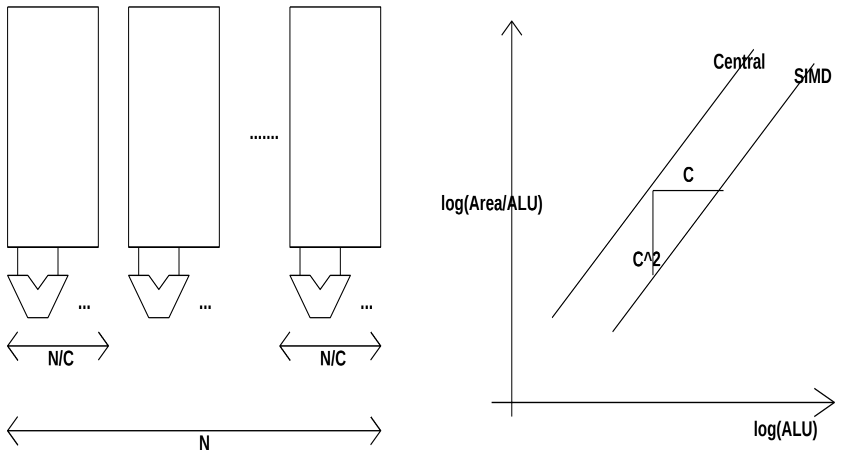

Figure 4: Comparison of structure and area for SIMD and central register files

If we plot the area per ALU versus number of ALUs for both the central and SIMD architectures, as in Figure 4, we see that both plots are linear with a slope of 2 (since they are dominated by N^2). This means that, although a lower constant is associated with the SIMD curve, the area of both register files is still proportional to N^3. However, the SIMD curve is shifted to the right of the central curve by a distance of C, showing that a SIMD architecture yields essentially C free ALUs over the central architecture.

1.3 Distributed Register Organization

The next transformation is to a distributed register file (DRF), which separates the storage and communication functions of the register file more than the architectures we just examined. In the DRF, each ALU input has its own dedicated register file; the distributed files are tied together by a $N * 2N$ (2 input, 1 output per ALU) crossbar switch that connects each ALU to all of the two-port register files. While the DRF transformation results in substantial area savings, its disadvantage is restricted communication; no longer can all ALUs communicate with all other ALUs as simply. Now some data values must be replicated in multiple register files, and register demand per ALU is increased.

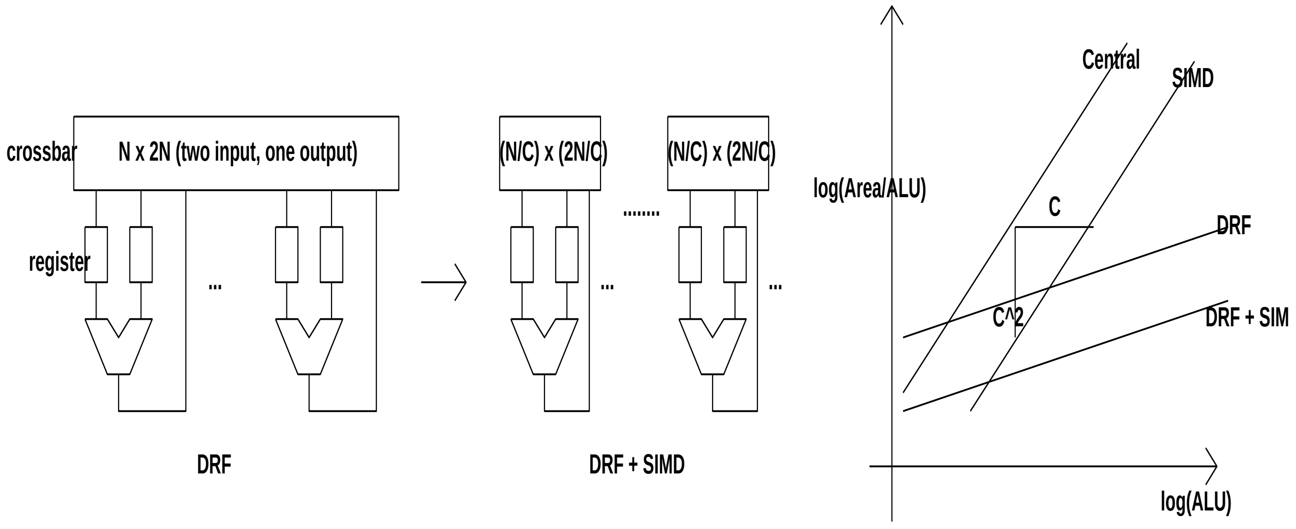

Figure 5: DRF and DRF+SIMD register files

The area of a DRF is dominated by the switch and is thus proportional to N^2. Plotting it on the same axes as the other register files shows a line with slope equal to 1. Transforming the DRF to a SIMD DRF is analogous to transforming a central register file to a SIMD register file; it improves the area by a constant factor, shifting the curve to the right and yielding C free ALUs.

We have explored four register file organizations: central, SIMD, DRF, and DRF SIMD. Any of these can be further transformed from a flat organization to a hierarchical organization or a stream organization. These transformations are described next.

1.4 Hierarchical Register Organization

The register file must interface with two different units with different performance demands: the ALUs, which need very high bandwidth (lots of ports), and the memory, which demands lots of registers to cover its latency but needs fewer ports. In the flat architecture, one central register file must deal with both of these conflicting demands, meaning that its area must be proportional to $(lots\ of\ registers) * (lots\ of\ ports)^2$. The hierarchical register organization partitions the register file into two levels, one to interface with memory and one with the ALUs.

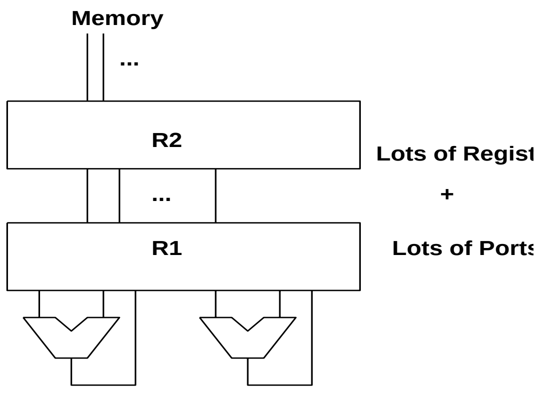

Figure 6: Hierarchical register organization

R_1 serves the ALUs, while R_2 is used to cover the memory latency. So, R_2 has lots of registers with just a few ports to stage memory operations, while R_1 has fewer registers but has many ports to interconnect the arithmetic units.

Conceptually, dividing the register file this way seems analogous to partitioning into L1 and L2 caches. There are two major differences: storage management and namespace. The partitioned register file allows higher bandwidth than its cache counterpart, but the disadvantage is that the namespace is not handled as neatly. Rather, programming complexity increases since it's harder to keep track of which register file a given data item is in.

The total area for a central, hierarchical register file is given by

$Area = [(3 + G)N]^2(r_aN) + (MN)^2(Tr_mN)$,

where $M * N$ is the number of ports to the Memory from R_2 and $G * N$ is the number of ports between the R_1 and R_2 register files.

This reduces the constant factor significantly over a flat architecture, but the overall tendency is still N^3. For the central register organization, R_1 dominates R_2 in size, meaning that R_2 is almost free. Curiously, for SIMD and DRF hierarchical structures, R_2 begins to dominate for values of N greater than 48. At this point, the area of the hierarchical register file actually exceeds the area of an equivalent flat register file.

There are two ways to get around this problem. One method, stream register organization, is discussed in the next section. The other method, which is not discussed in the paper, is to change R_2 into a clustered or distributed register file so that its area grows only as N^2 and not N^3.

1.5 Stream Register Organization

The stream transformation is applicable to SIMD, DRF, and DRF SIMD register files. It differs from the hierarchical organization by having R_2 perform only large, sequential

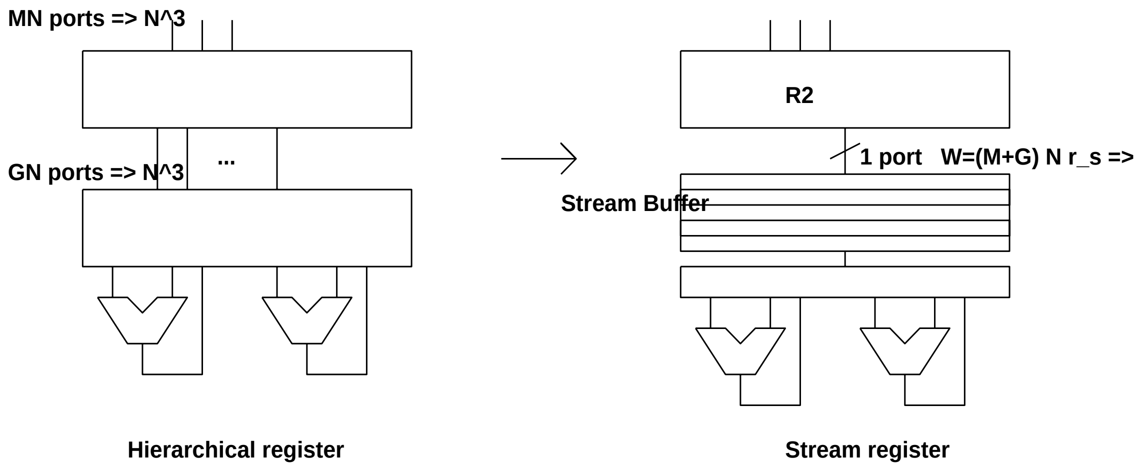

Figure 7: Comparison between hierarchical and stream register organization

transfers. R_2 is allocated a single port with a certain width $W = (M+G)Nr_s$, and a set of stream buffers. This width is needed to have the same bandwidth as a hierarchical register file. A stream buffer is a very wide register, pieces of which are accessed sequentially.

The payoff of the transformation into a stream architecture is that we can achieve an area proportional to N^2, since R_2 only needs 1 port. We also have to add in the area of the stream buffers, which grows as N^2 with a very small constant.

1.6 Comparison of Register Organizations

With each of the successive transformations described above, we are able to decrease the area of the register file at the cost of a decrease in flexibility. A review of the tradeoffs:

-Central to SIMD: we exploit reference locality in order to save register space. However, by going to a SIMD architecture, we also enforce locality; unfettered communication is no longer possible.

-SIMD to DRF: we save space but pay the price of replicating some data and arranging load balancing between register files. A DRF needs about three times as many registers as a SIMD register file to be efficient, which is usually a reasonable price to pay.

-Hierarchical to stream: stream architectures force memory references to be sequential. This constraint is being imposed by many memory systems today anyway.

1.7 Question and Answers

- *Question: How are stream register files different from multilevel caches?*

  Answer: Caching is in the memory namespace and is reactive on misses. On the other hand, stream data is explicitly managed in its own namespace and pro-actively gets data. The major plus of this method is bandwidth savings, and the disadvantage is the additional programming complexity caused by aliasing, which potentially presents a major coherence problem.

- *Question: What are variable lifetimes in the R_1 and R_2 levels?*

  Answer: In local register files, variable lifetimes are approximately one loop iteration. However, variable lifetimes in the SRF are very long.

2 The Raw Microprocessor : Hardware

M. Taylor, J. Kim, J. Miller, et al. "The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs." Laboratory for Computer Science, MIT.

2.1 Raw Microprocessor Organization

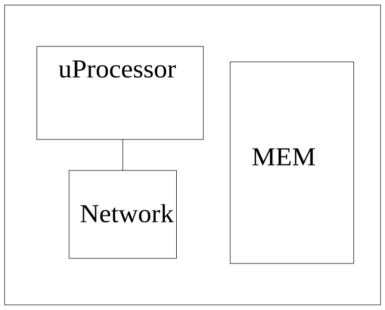

Figure 8: Tile of Raw Microprocessor

A Raw processor consists of sixteen identical programmable tiles (Figure 8). Each has its own microprocessor, data cache, memory, and interface to the interconnection network among the tiles. Raw has only 16 ALUs total, or two Imagine clusters' worth.

2.2 Application: Wavelets

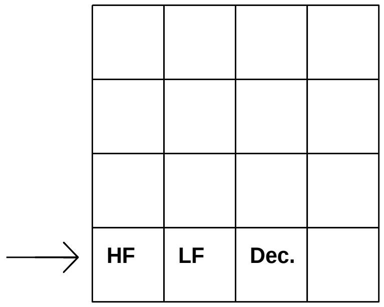

Figure 9: Wavelet running on Raw

Raw is both space- and time-multiplexed, so programming the Fast Wavelet Transform, as we did in the assignment, would involve allocating different tiles to different kernels and directing the flow of communication among tiles. Figure 9 is a diagram of how some of the data flow might be partitioned.

Even in this small example, the problem of load balancing becomes apparent: after the FIR/decimation phase is complete, some tiles may have much more to do than others depending on the size of the input. The compiler for StreamIt, the programming language developed by MIT, uses kernel fission and fusion to perform some of this load balancing between tiles; we'll read about the language and the compiler in another paper.

2.3 Parallelism: Raw vs. Imagine

One of the most important contrasts between Raw and Imagine is in the types of parallelism they exploit. Imagine invests most of its energy in exploiting data-level parallelism (DLP) and instruction-level parallelism (ILP), the cheapest forms of parallelism to exploit. The Raw processor, however, is almost exclusively concerned with thread-level parallelism (TLP), which is far trickier. Synchronization slowdowns and deadlock become real issues in Raw, since at any given time some processors are likely to be blocked waiting for others. Buffers in each tile can amortize the synchronization problem, but the burden of scheduling the work is still difficult. Figure 10 is a plot of the degree to which Raw and Imagine exploit each of these three types of parallelism.

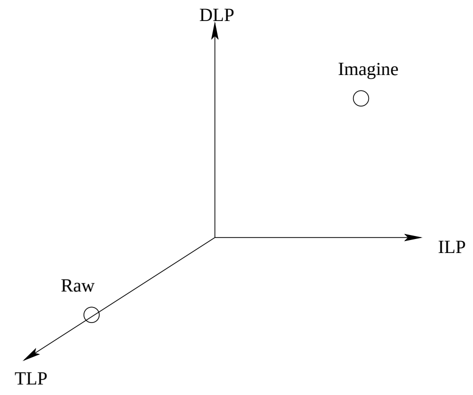

Figure 10: Parallelism on Imagine and Raw

An interesting problem would be to plot a surface of constant cost on these three axes. For instance, perhaps the cost of 8-way DLP is equivalent to 4-way ILP or 2-way TLP.

2.4 Raw's Static and Dynamic Networks

Communication in Raw has a latency of one cycle per hop between tiles; a turn counts as two hops. Scheduling communicating tiles close together is thus very important, since local communication is much faster than communication between opposite tiles on sides of the Raw processor.

The latencies of the static and dynamic networks are basically equivalent, but the routing tables for the static network take up a significant amount of die area. The advantage of the static network is that contention is arbitrated at compile time and is under full control. Additionally, the dynamic network can deadlock by over-commitment of buffering resources.

From the Raw floorplan, about 40 percent of the area of Raw is interconnect. The static and dynamic networks in Raw are flow-controlled, and that the processor and the networks proceed in lockstep. To communicate with the network, the processor writes to or reads from special registers that are mapped to the network.