Communication Scheduling and Conditional Operations

Lecture #6: Tuesday, 23th April 2002
Lecturer: Bill Dally, Mattan Erez
Scribe: Paul Wang Lee, Yeow Cheng Ong, Jean Suh
Reviewer: Mattan Erez

1 Logistics

One handout:

The Raw Microprocessor:
A Computational Fabric for Software Circuits and General Purpose Programs
Laboratory for Computer Science, Massachusetts Institute of Technology

Course Plan for next two weeks:

23 April
Discussion of the following research papers listed below:
Communication Scheduling (continue from last week)
Peter Mattson, Willian J. Dally, Scott Rixner, Ujval J. Kapasi, John D. Owens. Computer Systems Laboratory
Efficient Conditional Operations for Data-parallel Architectures
Ujval J. Kapasi, William J. Dally, Scott Rixner, Peter R. Mattson, John D. Owens, Brucek Khailany. Computer Systems Laboratory

25 April
Either discussion of the following two research papers listed below:
Register Organization for Media Processing
The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs
Or Discussion on Brook Tutorial

30 April
Project Brainstorm

2 Discussion: Communication Scheduling

2.1 Importance of Communcation Scheduling

Each operation needs to be scheduled in time space. The scheduler operates by scheduling an operation on a cycle and then assigning it to a functional unit using heuristic methods. Communication scheduling then allocates the interconnect resources needed by the operation to read its operands and write its result.

It allocates resources such that the result of an operation is accessible to all operations that use it as an operand. In the event that the operand is not used immediately, it has to be stored.

It enables the transfer of variables between the functional unit and register files. Copy operations may be performed to move the value between register files.

2.2 Scheduling Algorithm

First, an operation is tentatively scheduled on a cycle and assigned to a functional unit. The scheduler then attempts to schedule the requisite communication. The placement of the operation is still an open decision at this point, and may be changed later.

If communication scheduling succeeds, the operation is scheduled. If it fails, the placement of the operation is rejected, and the scheduler must reshedule the operation. The scheduler will then have to assign the operation to a different functional unit, or delay it until a later cycle, until it succeeds.

When the first of the two inter-communicating operations is scheduled, a communication is opened : communication scheduling determines the valid stubs and selects a stub that does not conflict with other stubs on the same cycle.

As each such operation is scheduled, the stub assigned to the open communication may be changed to allow stubs to be found for other communications. It should be noted that the number of permutations of valid stubs for a set of communications is exponential with the number of communications.

When the second communicating operation is being scheduled, the communication is closed : communication scheduling tries to find a write stub and a read stub that access the same register file to form a route. If neccessary, it inserts and schedules copy operations to connect the stubs and form a route.

In Figure 1, operation "d = f+z" has to be shifted down to the next cycle if communication scheduling fails. Copy operations have to be inserted to form the route.

Once a communication has been assigned to a route it is closed and the stubs and any copy operations that compose the route cannot be changed.

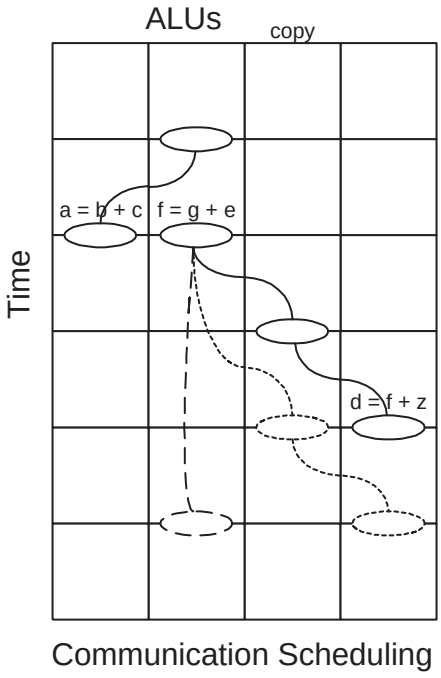

Figure 1: Rescheduling of operations

2.3 Optimization techniques in the scheduler

The scheduler schedules operations along the critical path first so that communication scheduling assigns communications between those operations early, providing them with preferential allocation of interconnect resources.

2.4 Techniques used in communication scheduling

Assigning priorities
When doing permutation of stubs, try to do closing communications first before opening others. Closing communications themselves are ordered by smallest copy range first, so that the communication that has the fewest cycles to schedule copy operations in have preference in choosing stubs to form routes.

Minimize communications
When assigning stubs, always find the nearest one available. This will minimize the

distance, the number of hops, and hence minimize the communication needed and reduce the chance of bottlenecks.

Backtracking
If the write operation is in a different basic block than the read operation, then the copy range is equal to the cycles in that basic block after the write operation. If the write operation is scheduled first, the read operation can be delayed indefinitely. If it proves impossible to schedule the required copy operations regardless of how the read operation is scheduled, the scheduler must backtrack to the write operation's basic block and force the write operation to be scheduled earlier to increase the copy range. The size of the basic block is fixed, and hence scheduler has to squeeze more operations into one cycle.

2.5 Other notes:

Communication scheduling deals with operations in the same kernel.

The scheduler can be improved by scheduling operation in the same cycle if other ALU's are available. It should attempt to optimize and not just make the operation possible.

3 Discussion: Efficient Conditional Operations for Data-parallel Architectures

3.1 Importance of conditionals for stream architecture

SIMD presents a number of problems. These problems and solutions using conditional streams will be discussed next.

Three ways of using conditional streams are explored.

3.2 Conditional switching

Consider the following code:

```
x >> a;
if (a>3)
   y << a*2;
else
   z << a/2;
```

for traditional SIMD, predication can be used as shown in figure 2.

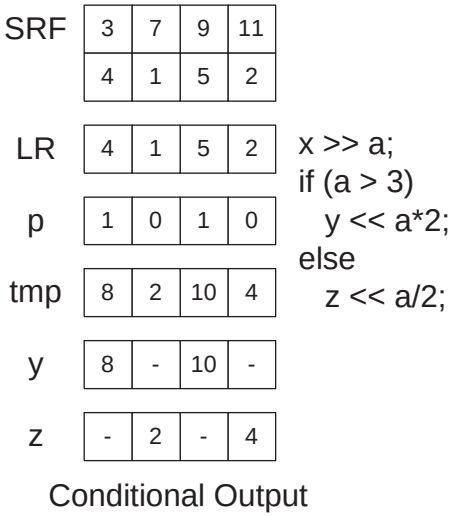

Figure 2: Traditional predication

This method results in waste of memory due to null values that are predicated out. Processing elements are wasted as well, since subsequent operations idle during null values until the data stream is compressed. However, compression requires an expensive

gather/scatter operation to memory.

Using conditional streams, invalid values are not written into the output stream at all, thereby avoid the above-mentioned costs.

3.3 Combining

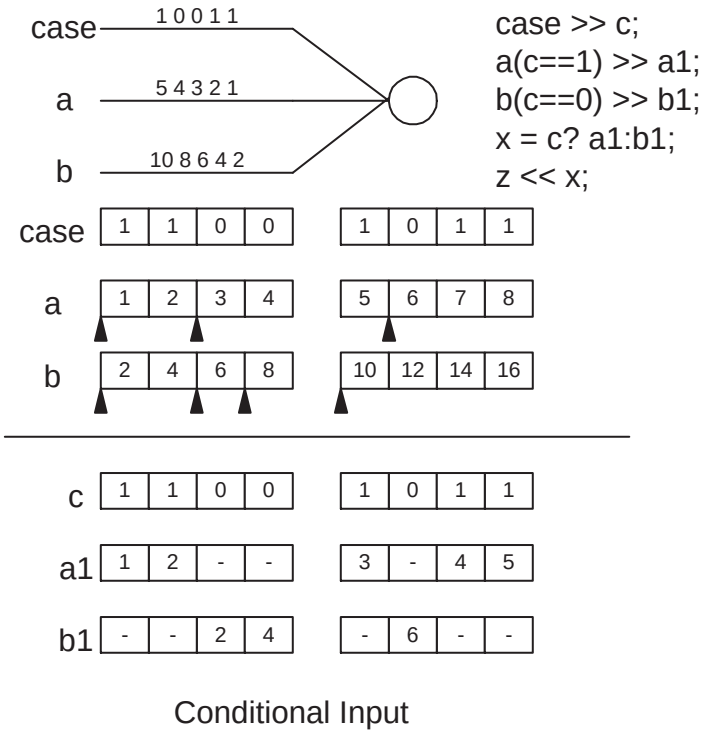

Figure 3: Combining using conditional streams

Streams can be combined using conditional streams as in the following code:

```
case >> c;
a(c==1) >> a1;
b(c==0) >> b1;
x = c ? a1:b1;
z << x;
```

Figure 3. shows how the streams are combined together.

3.4 Load Balancing

Some operations have different workloads among clusters. If it is solely predicated, processing elements are wasted while they wait for the cluster with the largest load to finish.

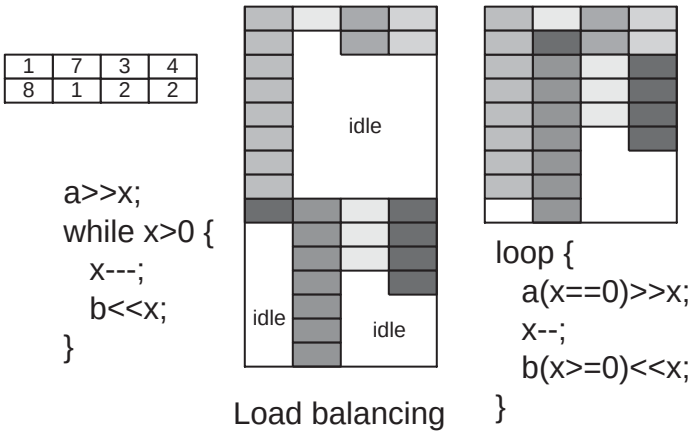

Figure 4: Load balancing using Conditional streams

By using conditional streams, clusters that finish early may retrieve new data and start on a new cycle, as illustrated in figure 4.

```
loop() {
   a(x==0) >> x;
   x --;
   b(x >=0) << x;
}
```

This will result in better utilization of processing elements. However, there are two disadvantages to this approach. First, outputs may be reordered, and this could be a critical factor for some applications. An additional stream of indices can be generated and used to sort the output stream later on, but this will be costly. Second, since initialization must be moved into the loop for SIMD operation, if this initialization cost is large, the loop body will become significantly longer. In this case, the loop can be unrolled to amortize the initialization cost, but in doing so it will also increase the loop size and result in higher load imbalance. Therefore a suitable tradeoff point should be found.

3.5 Hardware requirements for conditional streams

There are two requirements to be met by the hardware in order to implement conditional streams. These are switching and buffering. Stored values must be switched to arbitrary clusters, as illustrated by figure 5 below. Also, since data from two consecutive stream 'lines' may need to be accessed in the same cycle, a buffer must hold these values. In Imagine, switching is provided by intercluster communication and buffering is provided by the scratchpad.

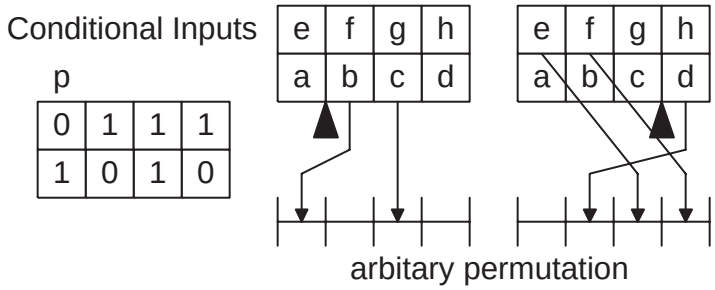

Figure 5: Switching from buffers to clusters

3.6 Tradeoff point between predication and conditional streams

```
{ Do A
    if (b) {
       Do B
       if (c) {
          ...
       }
    }
}
```

To use conditional streams, an application must be split to separate kernels.
This results in overhead, and thus if the conditional block is small, predication is better.

Following is a figure that shows the overhead for predication and conditional streams. For conditional streams, the overhead is fixed regardless of the number of operations that are conditional. The overhead for predication is proportional to the number of operations that are predicated.

It is clear which option to choose in the left and right extreme points, but it is more difficult to decide for the border region. Would MIMD be a viable alternative in this

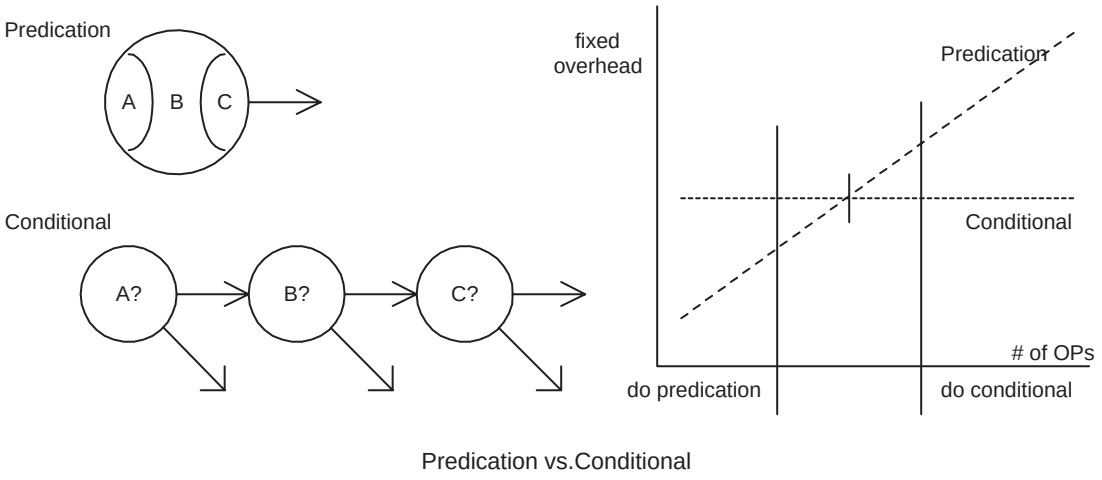

Figure 6: Tradeoff of predication and conditional streams

case? Some problems with MIMD include high instruction bandwidth and synchronization. Synchronization can no longer be implicit as in SIMD, and must be explicitly provided.