UCF

School of Computer Science

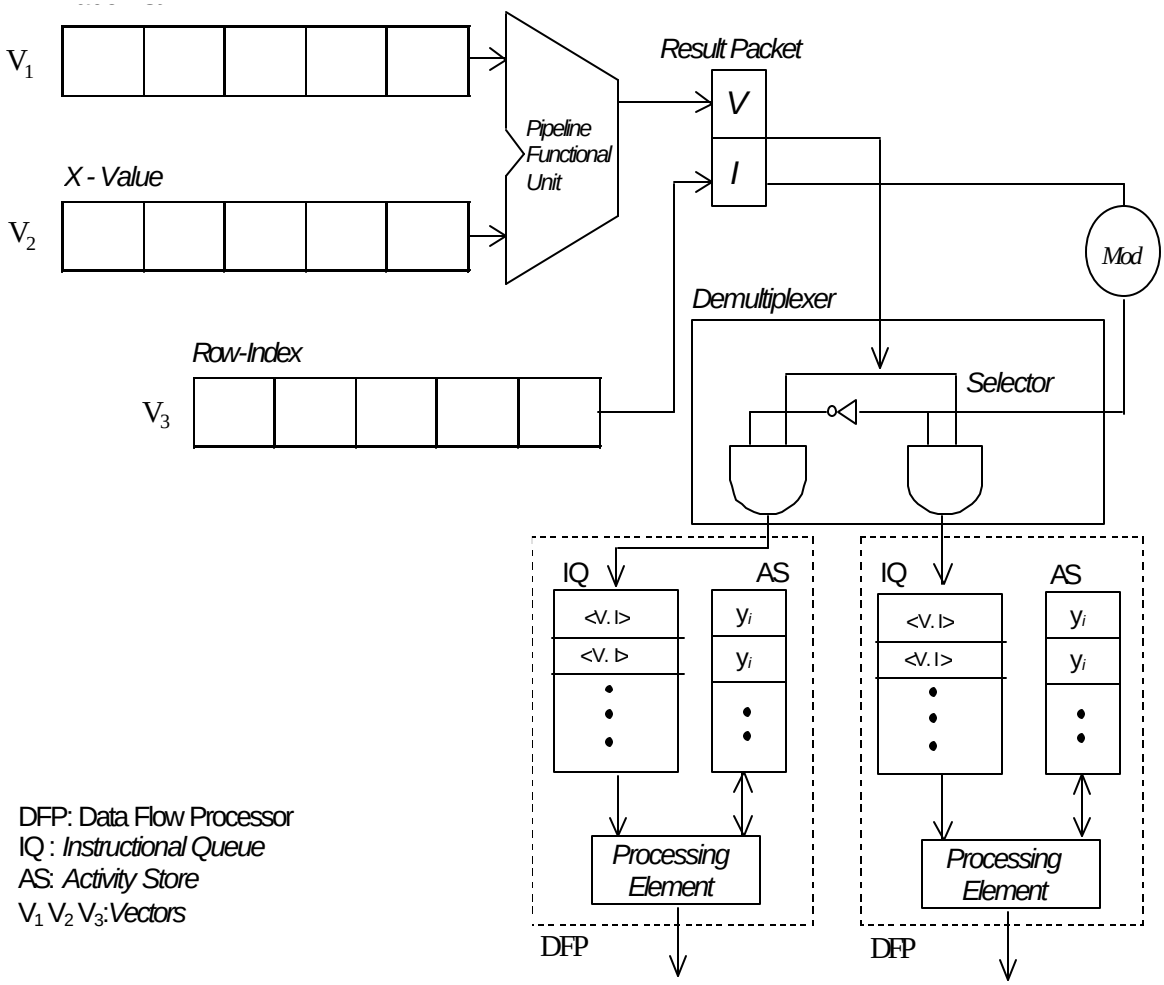
CDA 4150 Computer Architecture

Spring 2005

Project (Due 4/22/05)

The concept of the VecFlow architecture originates with the viewpoint of benefiting from the inherent advantages posted by the vector architecture and the data flow architecture. Several scientific applications are highly data parallel in nature. They involve performing the same sequence of operations on different data sets. The VecFlow architecture enhances the efficient pipelined processing capabilities of vector processing by introducing a scatter-add mechanism, which uses the data flow processing elements as additional pipelined functional units to exploit multichaining. The key feature is to ensure that there is a continual stream of data available for both the vector processor functional units and the data flow processing units to exploit data locality.

The proposed VecFlow architecture is shown in Figure1. VecFlow uses the classical vector-register processor architecture where all vector operations, except load and store, are executed among the vector registers. As in any other vector processor, in VecFlow, a set of vector registers are used to feed data to the pipelined functional units. In order to achieve a higher degree of parallelism, execution of simultaneous vector operations can be supported by having several functional units working concurrently. The results coming out from the vector functional units (multiplications) are streamed to a "Distributor" unit which scatters these intermediate results to the dataflow processing elements for the reduction (addition) step of the computation. The "Distributor" also needs to have additional information in order to scatter the values among the available dataflow processors. Thus using another vector register, we also provide the "Distributor" with the indices array, based on which the "Distributor' scatters data coming from the vector unit towards the data flow units. Therefore we see that the input to the "Distributor" is a packet of the form <Value,Index>. The "Distributor" performs a modulus operation on the index value using the number of DFUs units as a dividend to determine to which dataflow unit the packet has to be sent.



The scatter-reduction step employs a set of data flow processors (DFP) and a modulus operator (Demultiplexer in Figure 1), which scatters the incoming data stream from the vector processor unit among the dataflow units (DFU). These DFUs are capable of executing concurrently. As shown in Figure 1, the key components of the dataflow units are a Processing Element (PE), an Instruction Queue (IQ) and an Activity Store (AS). These dataflow units are based on Dennis' static model. It is obvious to note that the speed of the pipelined vector unit in producing results is much larger than the speed of the DFUs to apply the reduction operation. For this purpose, we have a Instruction Queue (IQ) which stores the resultant packets sent by the Distributor. The size of this Instruction Queue, which is nothing but a register file, is crucial to our design. In our simulation, we will study the size of this Instruction Queue by varying the number of DFUs so that they collectively can match the throughput of the vector processing unit, resulting in a small register file per DFU. The <value,index> packets arrived from distributor are store in the IQ. The PE load <value,index> packets from the Instruction Queue in a first come first serve basis and then access the activity store using the index to retrieve the appropriate Y[I] value. Then the following reduction step, Y[I] = Y[I] + value, is carried out to compute a new Y[I]. The new Y[i] value is stored back in the activity store while it is a

partial result of the computation of the Y_[I] value. But when the final Y_[I] values is computed, it is sent back to Memory.

As a final note we will discuss a few other issues. The execution of any application is first preceded by an initialization phase involving a set of actions performed to ensure the smooth execution of the application. Firstly, we need to load the vector registers with the appropriate data. This loading is performed by special pipelined Load-Store units, which are designed to move one word of data per clock cycle from the memory to the appropriate vector registers. Secondly we need to consider the latency of the pipelined functional units. It is not mandatory that all applications make use of both the vector and dataflow processors. But in some applications, the dataflow processors may also need to be initialized with appropriate data. In this scenario, the values that need to be loaded into the data flow units(for instance, counters with initial values) is first fed into the vector registers and then along with an index register, they are used to directly produce a <counter-value,index> packet which the distributor will send to the appropriate data flow unit to initialize some memory locations in the data flow unit.

You have to simulate the execution of the sparse matrix product, y =Ax, on Vecflow. The matrix has to be compressed using the transpose jagged diagonal storage format (TJDS), as you did in HW2, and then mapped onto Vecflow as explained in class. You have to calculate de arrival rate, λ, the number of data values sent to each DFU, and the departure rate of each DFU.

You must deliver in an envelope: a floppy with the programs and input data used, the resulting output from your simulation program, and the report.

Note from TA:

You must print out the final result of Y and verify the correctness of your result.
I recommend you use ibm32 matrix (HW2) as reference input.

You must send your zipped source code file to jfkong@cs.ucf.edu.
The file should be named **YourLastName_YourFirstName_Project.zip**)

Don't forget to turn in your paper report in class.

Thanks

Another Note from TA:

If your programming language is c/c++, be sure to make your program be compiled well either with gcc on eola or with VC 6.0 under windows .

Add a readme file describing how to compile and run your program into your source code.

Be sure to meet all the above requirements, otherwise you will be responsible for the subsequent consequence.

Any question? jfkong@cs.ucf.edu

Thanks