Vector Architecture

Questions we may have?

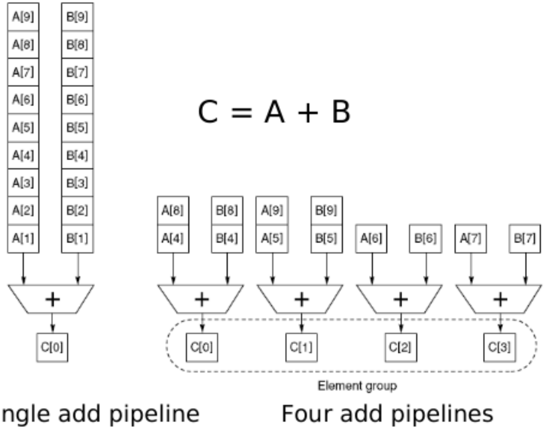
- How can a vector processor execute a single vector faster than one element per clock cycle?
- How does a vector processor handle programs where the vector lengths are not the same as the maximum vector length (mvl)?
- What happens when there is an IF statement inside the code to be vectorized?
- What does a vector processor need from the memory system?
- How does a vector processor handle multiple dimensional matrices?
- How does a vector processor handle sparse matrices?
- How do you program a vector computer?

Multiple Lanes: Beyond One Element per Clock Cycle

- Lanes are multiple parallel pipelines
- The RV64V instruction set has the property that all vector arithmetic instructions only allow element N of one vector register to take part in operations with element N from other vector registers. This dramatically simplifies the design of a highly parallel vector unit, which can be structured as multiple parallel lanes.
- For multiple lanes to be advantageous, both the applications and the architecture must support long vectors; otherwise, they will execute so quickly that you'll run out of instruction bandwidth
- Each lane contains one portion of the vector register file and one execution pipeline from each vector functional unit.

Multiple Lanes: Beyond One Element per Clock Cycle (continued)

The vector processor (A) on the left has a single add pipeline and can complete one addition per clock cycle. The vector processor (B) on the right has four add pipelines and can complete four additions per clock cycle.



Vector-Length Registers: Handling Loops Not Equal to max vector length(mvl)

- In a real program, the length of a particular vector operation is often unknown at compile time The solution to these problems is to add a vector-length register (vl). The vl controls the length of any vector operation, including a vector load or store. The value in the vl, however, cannot be greater than the maximum vector length (mvl). This parameter means the length of vector registers can grow in later computer generations without changing the instruction set.
- What if the value of n is not known at compile time and thus may be greater than the mvl? To tackle the second problem where the vector is longer than the maximum length, a technique called strip mining is traditionally used.
- Strip mining is the generation of code such that each vector operation is done for a size less than or equal to the mvl. One loop handles any number of iterations that is a multiple of the mvl and another loop that handles any remaining iterations and must be less than the mvl

Predicate Registers: Handling IF Statements in Vector Loops

- The presence of conditionals (IF statements) inside loops and the use of sparse matrices are two main reasons for lower levels of vectorization.
- Programs that contain IF statements in loops cannot be run in vector mode using the techniques we have discussed up to now
- Consider the following loop written in C:

```c
for (i = 0; i < 64; i=i+1)
    if (X[i] != 0)
    X[i] = X[i] - Y[i];
```

This loop cannot normally be vectorized because of the conditional execution of the body; however, if the inner loop could be run for the iterations for which X[i] != 0, then the subtraction could be vectorized.

Predicate Registers: Handling IF Statements in Vector Loops(continued)

- The common extension for this capability is vector-mask control.
- In RV64V, predicate registers hold the mask and essentially provide conditional execution of each element operation in a vector instruction. These registers use a Boolean vector to control the execution of a vector instruction, just as conditionally executed instructions use a Boolean condition to determine whether to execute a scalar instruction
- The entries in the destination vector register that correspond to a 0 in the mask register are unaffected by the vector operation.
- Using a vector-mask register does have overhead. Vector instructions executed with a vector mask still take the same execution time, even for the elements where the mask is zero. But the unnecessary computation's overhead will be be less compared to the gain from vectorization in most of the cases.

Memory Banks: Supplying Bandwidth for Vector Load/Store Units

- The behavior of the load/store vector unit is significantly more complicated than that of the arithmetic functional units.
- Unlike simpler functional units, the initiation rate may not necessarily be 1 clock cycle because memory bank stalls can reduce effective throughput. Lets say the data to be load in a vector are in two different memory blocks(can't be fetched at once), then a delay is bound to occur. This scenario is not possible in simple functional units.
- To maintain an initiation rate of one word fetched or stored per clock cycle, the memory system must be capable of producing or accepting this much data.
- Spreading accesses across multiple independent memory banks usually delivers the desired rate.

Memory Banks: Supplying Bandwidth for Vector Load/Store Units(continued)

- Most vector processors use memory banks, which allow several independent accesses rather than simple memory interleaving for three reasons:
  - Many vector computers support many loads or stores per clock cycle, and the memory bank cycle time is usually several times larger than the processor cycle time. To support simultaneous accesses from multiple loads or stores, the memory system needs multiple banks and needs to be able to control the addresses to the banks independently.
  - Most vector processors support the ability to load or store data words that are not sequential. In such cases, independent bank addressing, rather than interleaving, is required.
  - Most vector computers support multiple processors sharing the same memory system, so each processor will be generating its own separate stream of addresses.

Stride: Handling Multidimensional Arrays in Vector Architectures

- The position in memory of adjacent elements in a vector may not be sequential.
- Consider the following code for matrix multiplication:

```
for (i = 0; i < 100; i=i+1)
    for (j = 0; j < 100; j=j+1) {
        A[i][j] = 0.0;
        for (k = 0; k < 100; k=k+1)
            A[i][j] = A[i][j] + B[i][k] * D[k][j];
        }
```

- We could vectorize the multiplication of each row of B with each column of D. To do so, we must consider how to address adjacent elements in B and adjacent elements in D.
- But when an array is allocated memory, it is linearized and must be laid out in either row-major order (as in C) or column-major order (as in Fortran). This linearization means that either the elements in the row or the elements in the column are not adjacent in memory.

Stride: Handling Multidimensional Arrays in Vector Architectures(continued)

- We saw that blocking could improve locality in cache based systems. For vector processors without caches, we need another technique to fetch elements of a vector that are not adjacent in memory.
- This distance separating elements to be gathered into a single vector register is called the stride. Matrix D has a stride of 100 double words (800 bytes), and Matrix B would have a stride of 1 double word (8 bytes)
- Vector processor can handle strides greater than one, called nonunit strides, using only vector load and vector store operations with stride capability. And after the operation, it acts as if it had logically adja- cent elements.
- This ability to access nonsequential memory locations and to reshape them into a dense structure is one of the major advantages of a vector architecture
- In RV64V, Instruction VLDS (load vector with stride) fetches the vector into a vector register and VSTS (store vector with stride) stores a nonunit stride vector.

Stride: Handling Multidimensional Arrays in Vector Architectures(continued)

Supporting strides greater than one complicates the memory system. Once we introduce nonunit strides, it becomes possible to request accesses from the same bank frequently. When multiple accesses contend for a bank, a memory bank conflict occurs, thereby stalling one access. A bank conflict and thus a stall will occur if

$$\frac{\text{Number of banks}}{\text{Least common multiple (Stride, Number of banks)}} < \text{Bank busy time}$$

Gather-Scatter: Handling Sparse Matrices in Vector Architectures

- In a sparse matrix, the elements of a vector are usually stored in some compacted form and then accessed indirectly
- Sparse matrices are common, so it is important to have techniques to allow programs with sparse matrices to execute in vector mode.
- Consider the following C code:

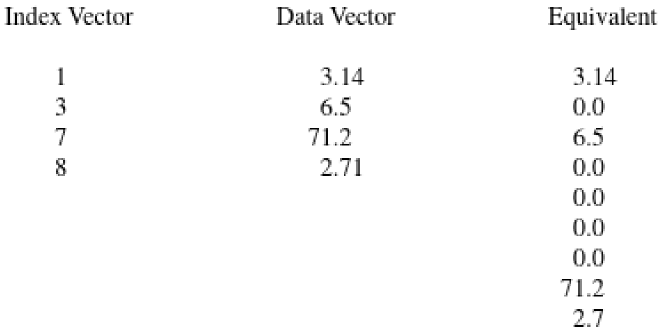
```
for (i = 0; i < n; i=i+1)
    A[K[i]] = A[K[i]] + C[M[i]];
```

This code implements a sparse vector sum on the arrays A and C, using index vec- tors K and M to designate the nonzero elements of A and C(A and C must have the same number of nonzero elements—n).

Gather-Scatter: Handling Sparse Matrices in Vector Architectures(continued)

- The primary mechanism for supporting sparse matrices is gather-scatter operations using index vectors.
- A gather operation takes an index vector and fetches the vector whose elements are at the addresses given by adding a base address to the offsets given in the index vector. The result is a dense vector in a vector register.
- After these elements are operated on in a dense form, the sparse vector can be stored in an expanded form by a scatter store, using the same index vector.
- Hardware support for such operations is called gather-scat- ter, and it appears on nearly all modern vector processors.
- RV64V has instructions vldi (load vector indexed or gather) and vsti (store vector indexed or scatter).

Gather-Scatter: Handling Sparse Matrices in Vector Architectures(continued)



Index Vector	Data Vector	Equivalent
1	3.14	3.14
3	6.5	0.0
7	71.2	6.5
8	2.71	0.0
		0.0
		0.0
		0.0
		71.2
		2.7

Programming Vector Architectures

- An advantage of vector architectures is that compilers can tell programmers at compile time whether a section of code will vectorize or not, often giving hints as to why it did not vectorize the code.
- This straightforward execution model allows experts in other domains to learn how to improve performance by revising their code or by giving hints to the compiler when it's okay to assume indepen- dence between operations, such as for gather-scatter data transfers.
- The main factor that affects the success with which a program runs in vector mode is the structure of the program itself:
  - Do the loops have true data dependences?
  - Can they be restructured so as not to have such dependences?

This factor is influenced by the algorithms chosen and, to some extent, by how they are coded.