Vector processors

- The most efficient way to execute a vectorizable application is a vector processor. → Jim Smith
- Vector architectures grab sets of data elements scattered about memory, place them into large, sequential register files, operate on data in those register files, and then disperse the results back into memory. A single instruction operates on vectors of data, which results in dozens of register-register operations on independent data elements.
- These large register files act as compiler-controlled buffers, both to hide memory latency and to leverage memory bandwidth. Since vector loads and stores are deeply pipelined, the program pays the long memory latency only once per vector load or store versus once per element, thus amortizing the latency over, say, 64 elements.

Typical recent systems have the following characteristics: .

- **Vector registers**. These are registers capable of storing a vector of operands and operating simultaneously on their contents. The vector length is fixed by the system, and can range from 4 to 128 64-bit elements.

- **Vectorized and pipelined functional units**. Note that the same operation is applied to each element in the vector, or, in the case of operations like addition, the same operation is applied to each pair of corresponding elements in the two vectors. Thus, vector operations are SIMD.

- **Vector instructions**. These are instructions that operate on vectors rather than scalars. If the vector length is vector length, these instructions have the great virtue that a simple loop such as

  ```
  for (i = 0; i < n; i++)
          x[i] += y[i];
  ```

  requires only a single load, add, and store for each block of vector length elements, while a conventional system requires a load, add, and store for each element.

- **Interleaved memory**. The memory system consists of multiple "banks" of memory, which can be accessed more or less independently. After accessing one bank, there will be a delay before it can be reaccessed, but

a different bank can be accessed much sooner. So if the elements of a vector are distributed across multiple banks, there can be little to no delay in loading/storing successive elements.

- **Strided memory access and hardware scatter/gather**: In strided memory access, the program accesses elements of a vector located at fixed intervals. For example, accessing the first element, the fifth element, the ninth element, and so on, would be strided access with a stride of four. Scatter/gather (in this context) is writing(scatter) or reading (gather) elements of a vector located at irregular intervals—for example, accessing the first element, the second element, the fourth element,the eighth element, and so on. Typical vector systems provide special hardware to accelerate strided access and scatter/gather.

- Efficient way to execute a vectorizable application is by Vector processor- Jim Smith
- ***vector processor*** is a CPU that executes instructions that operate on arrays of data.
- It collects set of data elements put them in the register file
- operates on the data in those register files and stores the results back in memory.
- These reg files acts like buffers and hide the memory latency.
- SIMD classification
- Also be called as **array processor**
- Improves performance on numerical simulations
- Used in Video game console and Graphic accelerators
- Ex: VIS, MMX, SSE, AltiVec and AVX
- Vector processors – Advantages
- Fast.
- Easy to use.
- Vectorizing compilers are good at identifying code to exploit.
- Compilers also can provide information about code that cannot be vectorized.
- Helps the programmer re-evaluate code.
- High memory bandwidth.
- Uses every item in a cache line
- Vector processors – Disadvantages

- They don’t handle irregular data structures as well as other parallel architectures.
- A very finite limit to their ability to handle ever larger problems. (scalability)

An Example

- Let's take a typical vector problem

  Y = a* X+ Y

- X and Y are vectors resident in memory
- a is a scalar
- This problem is the so-called SAXPY or DAXPY
  - SAXPY –single precision a * X plus Y
  - DAXPY -double precision a * X plus Y
- MIPS code for the DAXPY loop

```
        L.D     FO,a        ;load scalar a
        DADDIU  R4,Rx,#512  ;last address to load
Loop:   L.D     F2,0(Rx)    ;load X[i]
        MUL.D   F2,F2,FO    ;a x X[i]
        L.D     F4,0(Ry)    ;load Y[i]
        ADD.D   F4,F4,F2    ;a x X[i]     + Y[i]
        S.D     F4,9(Ry)    ;store into Y[i]
        DADDIU  Rx,Rx,#8    ;increment index to X
        DADDIU  Ry,Ry,#8    ;increment index to Y

        DSUBU   R20,R4,Rx   ;compute bound

        BNEZ    R20,Loop    ;check if done
```

- VMIPS code for DAXPY
  - L.D FO,a ;load scalar a
  - LV V1,Rx ;load vector X
  - MULVS.D V2,V1,F0 ;vector-scalar multiply
  - LV V3,Ry ;load vector Y
  - ADDVV.D V4,V2,V3 ;add
  - SV V4,Ry ;store the result
  - vector processor reduces the instruction bandwidth
  - executes only 6 instructions vs almost 600 for MIPS
  - It occurs because the vector operations work on 64 elements
  - overhead instructions that constitute half the loop on MIPS are not present in the VMIPS code

- compiler produces vector instructions for such a sequence
- resulting code spends its time running in vector mode , the code is said to be vectorized or vectorizable.
- Loops can be vectorized when they do not have dependences between iterations of a loop, are called loop-carried dependences.
- Another important difference between MIPS and VMIPS is the frequency of pipeline interlocks.
- In the MIPS code, every ADD. D must wait for a MUL. D, and every S. D must wait for the ADD. D.
- On the vector processor, each vector instruction will only stall for the first element in each vector, and then subsequent elements will flow smoothly down the pipeline.