Parallel computing is a type of computation in which many calculations or the executions of processes are carried out simultaneously. **Parallel computing** is the use of two or more processors (cores, computers) in combination to solve a single problem. The different processors access the data elements through shared memory. The largest computers using parallel processing architecture, known as Super-Computers, have hundreds of thousands of processors. **A vector processor** is a processor that can operate on an entire vector in one instruction. **Vector processors** reduce the fetch and decode bandwidth, as the number of instructions fetched are less.

Vector and Parallel Computers

Samir A. El-Seoud

Vector Processors

1. Introduction

A vector processor is a processor that can operate on an entire vector in one instruction. The operand to the instructions are complete vectors instead of one element. Vector processors reduce the fetch and decode bandwidth, as the number of instructions fetched are less. They also exploit data parallelism in large scientific and multimedia applications. Based on how the operands are fetched, vector processors can be divided into two categories:

- Memory-memory architecture operands are directly streamed to the functional units from the memory and results are written back to memory as the vector operation proceeds.
- Vector-register architecture, operands are read into vector registers from which they are fed to the functional units and results of operations are written to vector registers.

Instruction set has been designed with 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 construction of a highly parallel vector unit, which can be structured as multiple parallel lanes. As with a traffic highway, we can increase the peak throughput of a vector unit by adding more lanes. Adding multiple lanes is a popular technique to improve vector performance as it requires little increase in control complexity and does not require changes to existing machine code.

<u>Characteristics of vector processors:</u>

- A processor can operate on an entire vector in one instruction
- Work done automatically in parallel (simultaneously)
- The operand to the instructions are complete vectors instead of one element
- Reduce the fetch and decode bandwidth
- Data parallelism
- Tasks usually consist of:
  - Large active data sets
  - Poor locality
  - Long run times
- Each result independent of previous result
  - Long pipeline
  - Compiler ensures no dependencies
  - High clock rate
- Vector instructions access memory with known pattern
- Reduces branches and branch problems in pipelines
- Single vector instruction implies lots of work
  - Example:
    ```
    for(i=0; i<n; i++)
                        c(i) = a(i) + b(i);
    ```
- Data is read into vector registers which are FIFO queues
  - Can hold 50-100 floating point values
- The instruction set:
  - Loads a vector register from a location in memory
  - Performs operations on elements in vector registers

    - Stores data back into memory from the vector registers
- A vector processor is easy to program parallel SIMD computer
- Memory references and computations are overlapped to bring about a tenfold speed increase

A DVANTAGES

- Each result is independent of previous results- allowing high clock rates.
- A single vector instruction performs a great deal of work - meaning less fetches and fewer branches (and in turn fewer mispredictions).
- Vector instructions access memory a block at a time, which results in very low memory latency.
- Less memory access = faster processing time.
- Lower cost due to low number of operations compared to scalar counterparts.
- Programs size is small as it requires less number of instructions. Vector instructions also hide many branches by executing a loop in one instruction.
- Vector memory access has no wastage like cache access. Every data item requested by the processor is actually used.
- Once a vector instruction starts operating, only the functional unit (FU) and the register buses feeding it need to be powered. Fetch unit, decode unit, ROB etc can be powered off. This reduces the power usage.

DISADVANTAGES

- Works well only with data that can be executed in highly or completely parallel manner.
- Needs large blocks of data to operate on to be efficient because of the recent advances increasing speed of accessing memory.
- Severely lacking in performance compared to normal processors on scalar data.
- High price of individual chips due to limitations of on chip memory.
- Increased code complexity needed to vectorize the data.
- High cost in design and low returns compared to superscalar microprocessors.

2 Why are they expensive

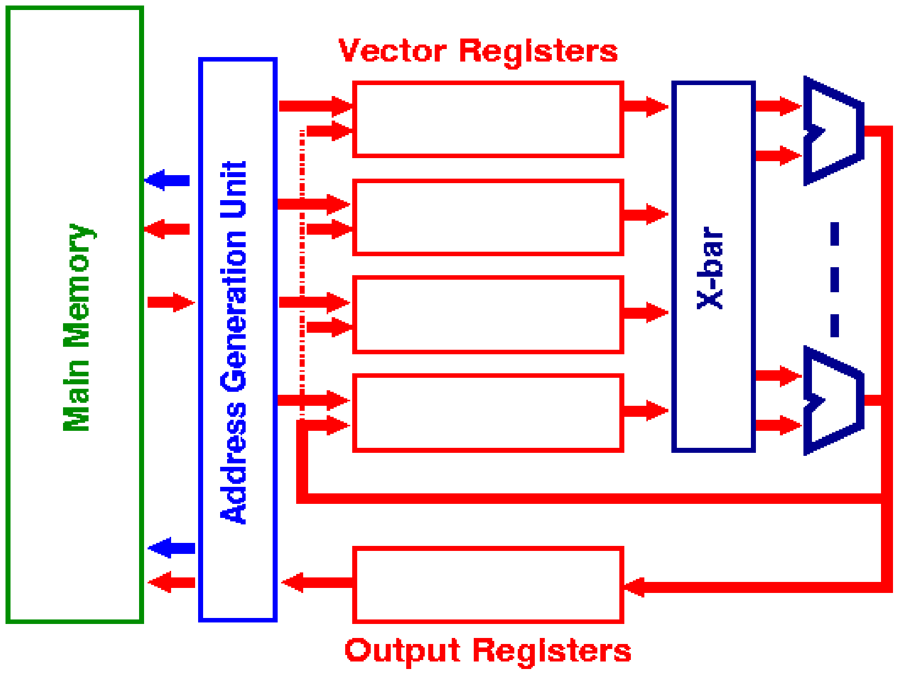
The reason behind the declining popularity of vector processors are their cost as compared to multiprocessors and superscalar processors. The reasons behind high cost of vector processors are

- Vector processors do not use commodity parts. Since they sell very few copies, design cost dominates overall cost.
- Vector processors need high speed on-chip memory which are expensive.
- It is difficult to package the processors with such high speed. In the past, vector manufactures have employed expensive designs for this.
- There have been few architectural innovations compared to superscalar processors to improve performance keeping the cost low.

3. What is scalar and vector processor?

- **Scalar processor**: A **CPU** that performs computations on one number or set of data at a time. Most computers have **scalar** CPUs. A **scalar processor** is known as a "single instruction stream single data stream" (SISD) **CPU**. Contrast with vector **processor**.

- **Vector Processors:** Commonly called **supercomputers**, the vector processors are machines built primarily to handle large scientific and engineering calculations. Their performance derives from a heavily pipelined architecture which operations on vectors and matrices can efficiently exploit.



Anatomy of a typical vector processor showing the vector registers and multiple floating point ALUs.

Data is read into the **vector registers** which are FIFO queues capable of holding 50-100 floating point values. A machine will be provided with several vector registers, V_a, V_b, *etc*. The instruction set will contain instruction which:

- load a vector register from a location in memory,
- perform operations on elements in the vector registers and
- store data back into memory from the vector registers.

Thus a program to calculate the dot-product of two vectors might look like this:

```
V_load     $V_a$, $add_A$
V_load     $V_b$, $add_B$
V_multiply $V_c$, $V_a$, $V_b$
V_sum      $R_1$,$V_c$
```

where the last operation sums the elements in vector register C and stores the result in a scalar register, R_1.

What is SIMD in computer architecture?

Single instruction, multiple data (**SIMD**), is a class of parallel computers. It describes computers with multiple processing elements that perform the same operation on multiple data points simultaneously.

4. Conclusion

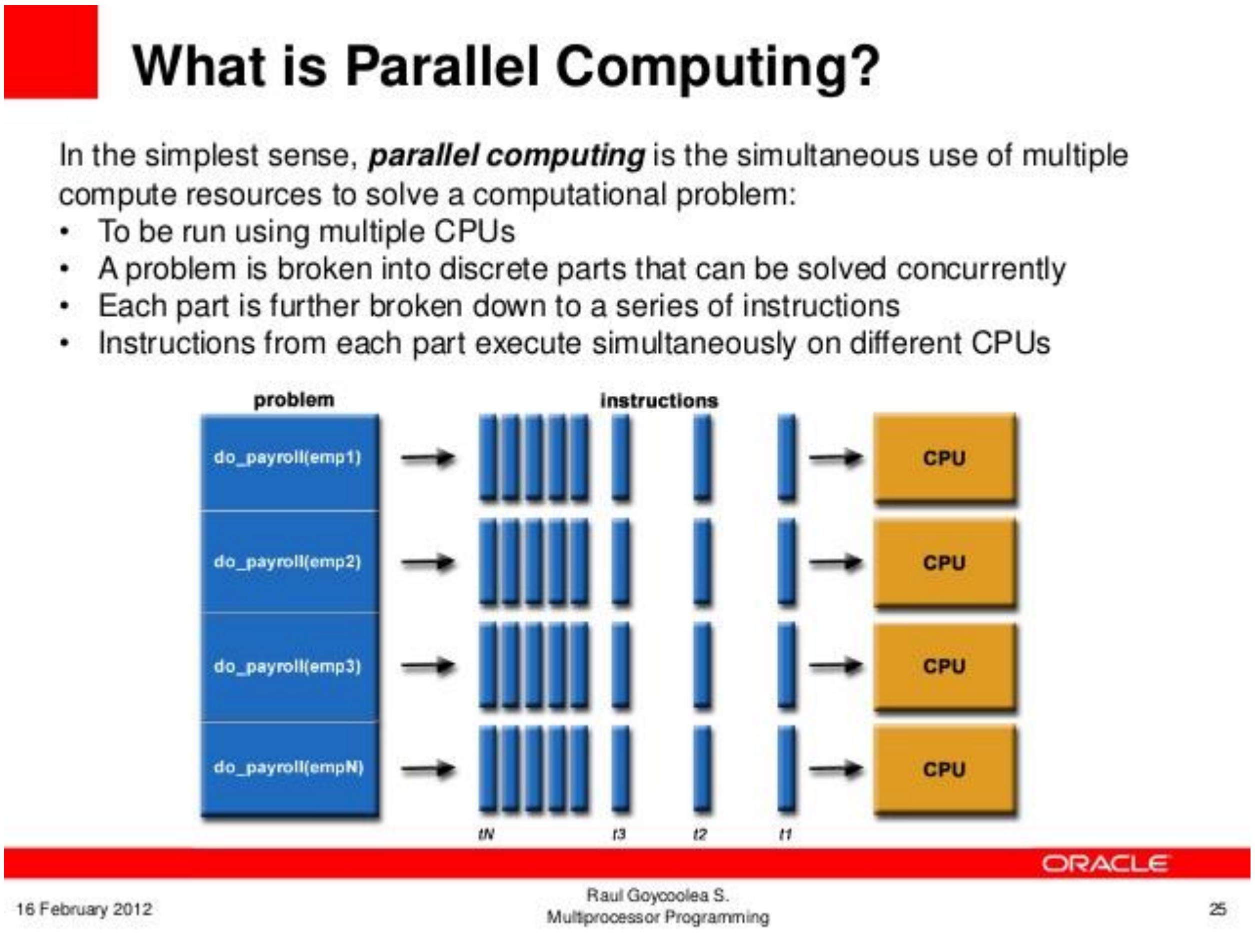
- Vector supercomputers are not viable due to cost reason, but vector instruction set architecture is still useful.
- Vector supercomputers are adapting commodity technology like SMT to improve their price-performance. Superscalar microprocessor designs have begun to absorb some of the techniques made popular in earlier vector computer systems (Ex - Intel MMX extension).
- Vector processors are useful for embedded and multimedia applications which require low power, small code size and high performance.
- The Vector machine is faster at performing mathematical operations on larger vectors.
- The Vector processing computer's vector register architecture makes it better able to compute vast amounts of data quickly.
- While Vector Processing is not widely popular today, it still represents a milestone in supercomputing achievement.
- It is still in use today in home PC's as SIMD units which augment the scalar CPU when necessary (usually GPUs).
- Since scalar processors designed can also be used for general applications their cost per unit is reduced drastically. Such is not the case for vector processors/supercomputers.
- Vector processors will continue to have a future in Large Scale computing and certain applications but can never reach the popularity of Scalar microprocessors

Parallel Computers

Parallel computing is a type of computation in which many calculations or the execution of processes are carried out simultaneously. **Parallel computing** is the use of two or more processors (cores, computers) in combination to solve a single problem.

Large problems can often be divided into smaller ones, which can then be solved at the same time. The programmer has to figure out how to break the problem into pieces, and has to figure out how the pieces relate to each other. Suppose instead that this work you have is only a single job, but it takes a very long time. Now you have to do something to reorganize the job, somehow breaking it into pieces that can be done concurrently. For example, if the job is to build a house, it can be broken up into plumbing, electrical, etc. However, while many jobs can be done at the same time, some have specific orderings, such as putting in the foundation before the walls can go up. If all of the workers are there all of the time, then there will be periods when most of them are just waiting around for some task (such as the foundation) to be finished. Not very cost-effective, and you are not getting the job done 100 times faster. Such is the life of a parallel programmer.

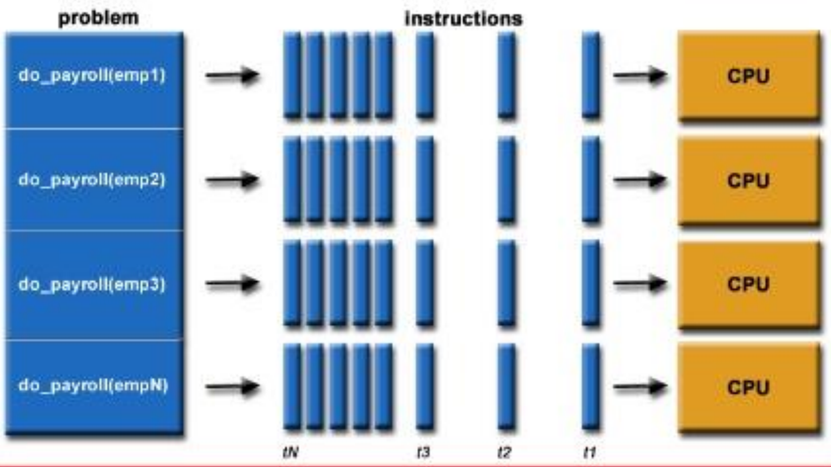
Parallelism has been employed for many years, mainly in high-performance computing, but interest in it has grown lately due to the physical constraints preventing frequency scaling. As power consumption (and consequently heat generation) by computers has become a concern in recent years, parallel computing has become the dominant paradigm in computer architecture, mainly in the form of multi-core processors.



What is Parallel Computing?

In the simplest sense, ***parallel computing*** is the simultaneous use of multiple compute resources to solve a computational problem:

- To be run using multiple CPUs
- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different CPUs



ORACLE

16 February 2012 Raul Goycoolea S. Multiprocessor Programming 25

Is Parallel Computing Easier or Harder than Serial Computing?

Easier:
Most of the universe is inherently parallel, with numerous things happening simultaneously. You may put your pants on one leg at a time, but millions of people are putting their pants on simultaneously. Many problems have an abundance of natural parallelism that a programmer can exploit. Sometimes people are forced to rethink the problem when they try to develop a parallel program, and they change their entire approach in order to directly utilize the inherent parallelism.

Harder:

- Blatant poor performance: it is easier to see that you're not doing well. On a serial computer, the processor may wait as it moves data from RAM to cache, but it basically appears busy all the time and it is only after you do a more detailed analysis that you determine that it is actually performing poorly. On a parallel computer of 100 processors most people quickly notice that a program is not running 100 times faster. The serial program may not have been very good but it is not generally as obvious as the fact that many of the parallel processors are often idle.

- Little experience: most programmers have little or no experience with parallel computing, and there are few parallel programs to use off-the shelf or even good examples to copy from. Hence, people often have to reinvent the parallel wheel. People developing parallel systems software are similarly behind on their learning curves. Often they reuse material developed for serial systems, even when it causes performance problems.
- Poor portability: a program may work on one machine, but when the program is ported to a new machine it may be so different that drastic changes need to be made just to permit the program to be run. Most serial computers have the same basic organization, but this is not so for parallel computers. Some parallel computers, such as the popular cluster systems, are essentially just a collection of computers linked together with Ethernet. They use simple commands, similar to read and write, to communicate among the processors. Such parallel computers are known as *message-passing* systems, or *distributed memory* computers. Clusters are usually quite inexpensive because they are built from commodity parts, and they have become quite common in businesses and universities. Distributed memory computers with better performance have faster interconnections among the processors, and software better tuned to support parallelism, but this increases the cost because many of the parts are more non-standard.

  Other parallel computers, such as some of the systems from SGI and Cray, are *shared-memory* systems, where every processor can directly read and write data from every memory location. Think of this as being like everyone using a blackboard to do their calculations, where everyone has chalk and an eraser and can decide to write anywhere on the board. For these systems, there is a standard known as OpenMP, which is a collection of language extensions to Fortran and C/C++. Shared memory computers must use many non-commodity parts to support the sharing, and hence tend to be more expensive than distributed memory systems. However, it is often easier to write a program for shared memory systems than for distributed memory ones.

  Yet other computers exploit graphics processors, GPUs, to achieve parallelism. However, while a GPU may have 100 cores, there are serious restrictions on their programability, and serious performance problems if there is much data movement. Further, the languages to use for GPUs are rapidly evolving, so it is unclear if you should use CUDA, OpenCL, OpenACC, accelerator extensions being incorporated in OpenMP, etc.
- Amdahl's law: if the foundation takes 5% of the time, and only one worker can do it while everything else has to wait until the foundation is done, then you can never get the job done in less than 5% of the original time, no matter how many workers you have. Incidentally, this helps explain why it is much easier to be a successful parallel programmer on a small system. In such a situation, if everything else parallelizes perfectly, then 100 workers would take 5% (serial) + 95%/100 (perfect parallel) = 5.95% of the original time, compared to 100%/100= 1% of the original time if everything was perfectly parallel. 1/(100*0.0595) ≈ 0.17, so the efficiency is only about 17%, where *efficiency* is (time for 1 worker)/[(number of workers)*(the time they take)]. You can think of this as the wages if one worker does it, divided by the total wages when the group does it. 20 workers would work at about 51% efficiency, and 10 workers at about 69%. The fewer the workers, the higher the efficiency.
- Weakest links: if a group of workers all depend on each other, then the group can go no faster than the slowest worker. If one worker is talking on the cellphone trying to arrange a date, everyone slows down, especially if they are trying to listen in. Other weak links can be the compiler, operating system, communication system, etc.

Why can't you hide?
In the past, when someone needed to solve bigger problems they could often wait a bit, and a faster computer would come out. This was mainly due to *Moore's law*, which people interpreted to mean that the speed of computers would double about every two years. However, this isn't true any longer: if you look at the GHz speed of the processors in a desktop computer 2 years ago, versus the speed now, it has barely, if any, increased. Basically this happened because they can't make reliable low-cost processors that run significantly faster.

What has doubled, however, is the number of processors on a single chip. Some smartphones now have 8 cores (processors), you can buy CPU chips with 12 cores, and this number will soon increase. These are called "multi-core" or "many-core" chips. There are also graphics processing units (GPUs) with over 100 highly specialized processors. This is closer to what Moore was talking about, because what he really said was that the number of transistors would keep doubling. The rate of improvement will slow down, but significant increases in the number of transistors should continue for at least a decade.

If there are only a few cores in your computer then it is pretty easy to find tasks that can be done simultaneously, such as waiting for keystrokes and running a browser, i.e., embarrassingly parallel. However, as the number of processors keeps increasing, the parallelization problems mentioned above become increasingly serious. There are now parallel computers that have > 1,000,000 cores, and people are planning for ones that will have > 100,000,000.

Overall, this means that there is a massive need to make use of the parallelism in multi-core chips for almost any problem, and to use many of these combined together for large problems. Job security for me! (and you, if you learn parallel computing).

Summary

Vector processing

A computer with built-in instructions that perform multiple calculations on vectors (one-dimensional arrays) simultaneously. It is used to solve the same or similar problems as an array processor; however, a vector processor passes a vector to a functional unit, whereas an array processor passes each element of a vector to a different arithmetic unit.

Vector processing is a procedure for speeding the processing of information by a computer, in which pipelined units perform arithmetic operations on uniform, linear arrays of data values, and a single instruction involves the execution of the same operation on every element of the array

A **vector processor**, or **array processor**, is a central processing unit (CPU) that implements an instruction set containing instructions that operate on one-dimensional arrays of data called vectors. This is in contrast to a scalar processor, whose instructions operate on single data items.

Parallel processing

Parallel processing involves a technique by which complex data sets are broken into individual threads and processed simultaneously across one or more cores. Both AMD and Intel processors have incorporated this technique (known as HTT) to greatly increase the speed at which they operate. Until recently, this did not always provide a significant increase in speed because the technology to properly split up data sets and then bring them back together was in its infancy.

Parallel computing is the processing of data many bits at a time as opposed to serial computing which is the processing of data one bit at a time.

In other words, parallel computer runs more tasks in **parallel**, over more CPUs, for faster execution.