

PART I - Fundamentals of Parallel Computing

Objectives

- What is scientific computing?
- The need for more computing power
- The need for parallel computing and parallel programs

What is scientific computing?

The concepts of *numerical analysis*, *scientific computing*, and *simulation* are related but distinct.

Numerical analysis is the study of the design and analysis of algorithms that use numerical approximations (as opposed to symbolic manipulations) for solving problems in mathematical analysis (as opposed to discrete mathematics).

Examples of the types of problems popular in numerical analysis are special function evaluation, interpolation, root-finding, numerical linear algebra (linear systems of equations, eigenvalue problems, etc.), differential equations, optimization, and quadrature problems.

Numerical analysis does not assume or require the existence of a computer to perform calculations.

Approximation errors due to rounding, truncation, or discretization would still exist, and the study of their propagation through stability of the problem and the algorithm would remain important.

Scientific computing generally refers to the solution of a mathematical model by means of a computer.

Naturally, numerical analysis is fundamentally important to scientific computing.

Scientific computing is often used synonymously with *computational science*.

Computational science is about using computation, as opposed to theory and experiment, to do science.

I view computational science as a superset of scientific computing and would use it more synonymously with a holistic (as opposed to literal) interpretation of *simulation*, i.e., constructing a mathematical model of a system of interest, simulating it (i.e., solving the model using a computer), and interpreting or otherwise quantitatively analyzing its output.

Regardless of the nuances, all three concepts are distinct from *computer science*, which is more about the study of computers, how computations are performed, and information processing.

The need for more computing

Historically, science has been based on the paradigms of theory and experiment.

However, with the extraordinary and relatively recent increase in computing power, a third scientific paradigm has emerged: simulation.

Simulation allows scientists to test theories without the need for experiments.

This is especially important when experiments are expensive, dangerous, or simply impossible.

Simulation can also reduce the number of experiments needed to validate a theory.

Similar to experiment, simulation can generate data that can inform theory.

Industry uses simulation to make informed decisions in the absence of complete theory or observational data but also to reduce the time required to go from design to prototype to product.

The need for more computing

The natural progression of scientific discovery and industrial research is to continually seek out harder and harder problems to study.

We have already reached a stage where computing is all around us: robots, cars, weather prediction, GPS, google, . . . , all require significant amounts of high-level computation to function.

As of November 2013, the fastest¹ supercomputer (the Tianhe-2 (MilkyWay-2) computer, based on Intel Xeon E5-2692 12C 2.2000 GHz / Intel Xeon Phi 31S1P with 3,120,000 cores developed by China's National University of Defense Technology) was clocked at about 34 PFlops ("sustained"); its theoretical peak is about 55 PFlops.

This may sound like a lot of flops per second, but it may not be as much as you think.

¹according to the maximal LINPACK performance achieved

The need for more computing

Example: A virtual human heart.

The human heart has approximately 10^{10} muscle cells.

A model of the human heart will require approximately 100 variables per cell.

Suppose it takes 1000 flops to advance the state of each variable by 1 microsecond of simulated time.

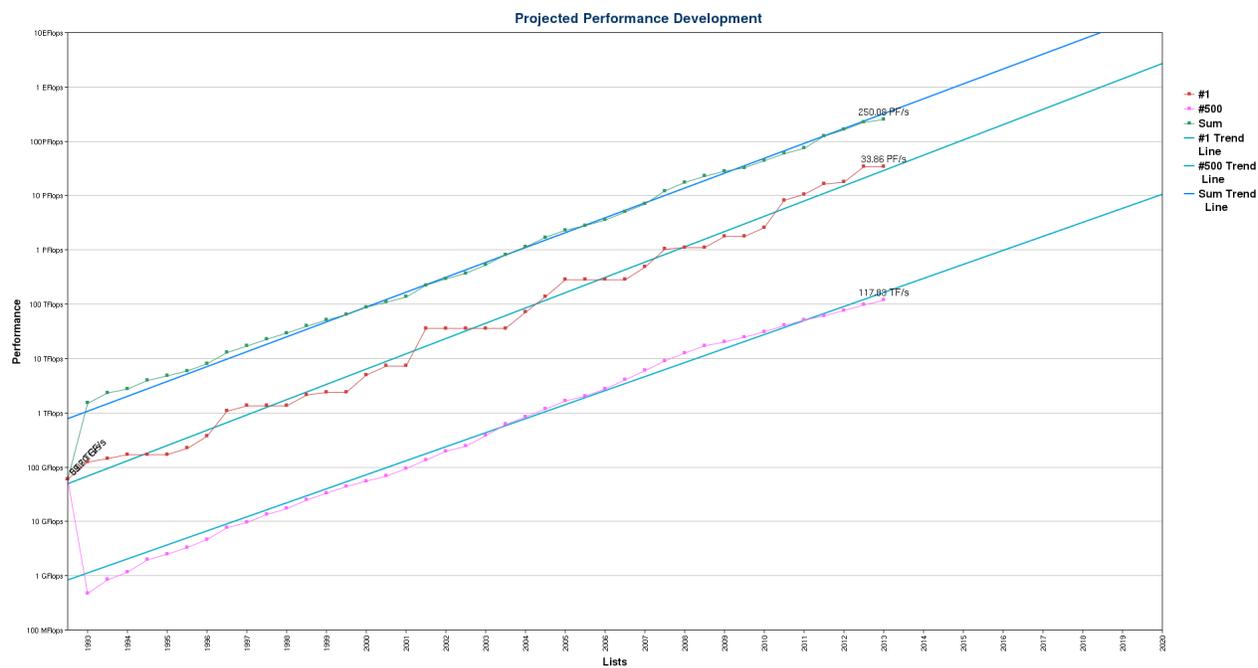
Then a microsecond of simulated time needs 10^{15} flops.

The bad news: The fastest supercomputer can only do about 10^{10} flops per microsecond of real time.

The good news: This is a bit of a worst-case analysis (total flop count for a microsecond of a useful virtual heart might be closer to 10^{11}), and computers are still getting “faster” (exaflop computing).

We might see a useful virtual heart this decade.

Projected Performance Development



Source: www.top500.org

The need for more computing

Other examples:

- Climate modelling
- Natural disaster prediction / mitigation
- Protein folding; drug discovery
- Energy research
- Data analysis (genomics, particle physics, astronomy, ocean research, search engines)

The economics of CPU technology

Technology is governed by economics.

The rate at which a computer processor can execute instructions (e.g., floating-point operations) is proportional to the frequency of its clock.

However,

$$\begin{aligned} \text{power} &\sim (\text{voltage})^2 \times (\text{frequency}), \\ \text{voltage} &\sim \text{frequency}, \\ \implies \text{power} &\sim (\text{frequency})^3 \end{aligned}$$

e.g., a 50% increase in performance by increasing frequency requires **3.4 times more power**.

By going to 2 cores, this performance increase can be achieved with **16% less power**².

²This assumes you can make 100% use of each core.

Physical limits of CPU technology

Suppose we wanted to build a single CPU capable of 1 TFlop (= 10^{12} flops) per second; e.g., add vectors x , y with 10^{12} elements and store in vector z .

This requires 3×10^{12} copies between memory and registers per second.

If data can travel at the speed of light (3×10^8 m/s), then the average distance of a word of memory to the CPU must be 10^{-4} m.

Suppose we have a square grid with the CPU at the centre. The size of the square can be 2×10^{-4} m.

A row of memory contains $\sqrt{3 \times 10^{12}} = \sqrt{3} \times 10^6$ words. So each word must fit into

$$\frac{2 \times 10^{-4}}{\sqrt{3} \times 10^6} \approx 10^{-10} \text{ m.}$$

This is the size of a relatively small atom!

Physical limits of CPU technology

In other words, unless we can figure out how to represent a 32- (or 64-) bit word with a single atom, it will be impossible to build such a computer.

Transistors are already getting so small that (undesirable) quantum effects (such as electron tunnelling) are becoming non-negligible.

No matter which way you look at it,

increasing frequency is no longer an option!

When increasing frequency was possible, software got faster because hardware got faster.

Hardware is in some sense still getting better, but increases in performance are only possible nowadays by exploiting parallelism.

The only software that will survive will be that which takes advantage of parallel processing.

When to parallelize

Not every problem needs to be parallelized in order to find its solution.

There are only two scenarios in which parallelization makes sense as a way to help solve a problem:

1. The problem is too big to fit into the memory of one computer.
2. The problem takes too long to run.

The goal in both of these scenarios can be described as reducing the amount of (real) time it takes to get the solution³.

Note that this is **not** the same as reducing the overall amount of computation.

³In the first scenario, the original time required can be viewed as infinite (no such single computer exists) or indefinite (until you acquire a new computer).

The challenge of parallel computing

Even if we had an unlimited processors and memory, there are still many critical issues to deal with.

- Designing and implementing suitable interconnection networks for processors and memory modules.
- Designing and implementing system software.
- Devising algorithms and data structures.
- Usefully dividing algorithms and data structures into sub-problems.
- Identifying how the sub-problems will communicate with each other.
- Assigning sub-problems to processors and memory modules.

In this course, we will be mostly concerned with the last 4 items.

The need for parallel computing

Parallel programming and parallel computers allow us to take better advantage of computing resources to solve *larger and more memory-intensive* problems *in less time* than is possible on a single serial computer.

Trends in computer architecture towards multi-core and accelerated systems make parallel programming and HPC a necessary practice going forward.

It is already difficult to do leading-edge computational science without parallel computing, and this situation can only get worse in the future.

Computational scientists who ignore parallel computing do so at their own risk!

Summary

- Parallel computing is here to stay.
- Parallel programming will become the prevalent mode of programming in the very near future.
- Leading-edge computational science and technology must take advantage of parallel computers and parallel programs.