Partnership for Advanced Computing in Europe

Introduction to HPC Programming

1. HPC: Introduction, Motivation and Architectures

Valentin Pavlov <vpavlov@rila.bg>

About these lectures

- This is the first of series of six introductory lectures discussing the field of High-Performance Computing;
- The intended audience of the lectures are high-school students with some programming experience (preferrably using the C programming language) having interests in scientific studies, e.g. physics, chemistry, biology, etc.
- This lecture provides background, motivation and some insights about different parallel architectures.

What is HPC?

- As the name suggests, High-performance Computing (HPC) denotes a **certain way of using computational devices so as to maximize their *performance***: performing as much calculations per unit time as possible.
- If we are to maximize something, we need a way to measure it. The basic measurement unit of performance in HPC is called FLOPS (FLoating point Operations Per Second).
- Modern computers are capable of performing millions of billions FLOPS, so derived units are usually used instead.

What is HPC?

- These derived units are:
  - 1 megaFLOPS (MFLOPS) = 10^6 FLOPS;
  - 1 gigaFLOPS (GFLOPS) = 10^9 FLOPS;
  - 1 teraFLOPS (TFLOPS) = 10^{12} FLOPS;
  - 1 petaFLOPS (PFLOPS) = 10^{15} FLOPS;
  - 1 exaFLOPS (EFLOPS) = 10^{18} FLOPS;
- Most microprocessors today can do up to 4 FLOPS per clock cycle, at least theoretically. Thus, a very simplistic way to calculate the maximum performance of a computing device is to multiply the number of cores by the clock frequency by 4.

Example performance

- Typical theoretical performance of various computing devices:
  - Smartphones and similar gadgets: around 1—2 GFLOPS;
  - Laptops and desktops: around 10 GFLOPS;
  - High-end servers: around 25—100 GFLOPS;
  - Graphical accelerators: around 20 GFLOPS — 2 TFLOPS;
  - 1 rack of IBM Blue Gene/P: around 14 TFLOPS;
  - Tianhe-2A[1]: around 54 PFLOPS = 54,000,000 GFLOPS (!);

[1] #1 in the `http://top500.org` as of June 2013

HPC Uses

- HPC is used by both science and industry;
- In scientific studies, HPC is used to create digital experiments – models of physical systems whose behavior is being examined through governing equations calculations. Examples include weather prediction, discovery of new medicines, nanotechnology and discovery of new materials, disaster prevention, etc.
- Major industrial users of HPC are oil and gas companies, pharmaceutical companies, defense and aerospace organizations, automobile manifacturers, financial institutions.

Simplistic example

- Let's say we want to perform numeric experiments in the area of new medicine discovery;
- For this we decide to simulate the behaviour of the microbial ribosome—a large mollecular machine that drives protein synthesis;
- If we can invent a chemical compound that can interact with the microbial ribosome and prevent it to synthesize proteins, we'll have a new antibiotic medicine.

Simplistic example

- Let's assume that the ribosome contains roughly 1,000,000 atoms[2] and we want to calculate the trajectory of these 1,000,000 atoms;
- The trajectory of a particle is given as the sequence of the particle's position over the course of several—let's say 1,000—steps.
- Given the current position, how do we find the position at the next instant of time?

[2] This example is just for illustration purposes. In a real experiment we need to put this in a solvent (water) and account for its atoms also, and they can easily be twice as much.

Simplistic example

- By using Newton's Second Law $\vec{a} = \vec{F}/m$ we can find out the accelaration of the particle if we know the value of the force that acts on it;
- The value of the fource in this case is given by the Coulomb's Law $\vec{F} = \frac{1}{4\pi\varepsilon}\frac{q_1 q_2 \vec{r}}{|r|^3}$ for the force between a pair of charges;
- If we know its new acceleration and its current velocity, we can calculate the particle's new velocity (integration);
- Similarly, if we know its new velocity and its current position, we can find the particle's new position;

Simplistic example

- The key here is to note that we have to work out Coulomb's Law formula *for each pair of particles at every time step*.
- This makes $1000000 \times 999999 \times 1000 \approx 10^{15}$ calculations of the formula;
- Roughly counting the operations needed to calculate the formula once, we can say there are 15 operations (5 operations for each of the components of the force in each of the 3 direction);
- This totals the number of floating point operations to around 15×10^{15}

How long it will take to calculate this?

- Smartphone: 90 days;
- Desktop: 18 days;
- High-end server: 2 days;
- Graphical accelerator: 1 hour and a half;
- 1 rack of IBM Blue Gene/P: 20 minutes;
- Tianhe-2A: 300 miliseconds;
- And what if we want to perform 1,000 such experiments? Which one would *you* choose?

Why use HPC instead of real experiments?

- **Digital experiments are a lot cheaper than real ones**: for example, a pharmaceutical company is trying to find a new drug. It has let's say 100 potential formulas that *might* work. If by using digital experiments the company can narrow down these 100 candidates to let's say 10, it will save enourmous amount of time and money, that would otherwise be spent on clinical trials for these 90 formulas that were rejected by the digital experiments.
- This is true for almost all digital vs real-life experiments.

Why use HPC instead of real experiments?

- **Ethical issues**: sometimes the subject of an experiment has to be a human being or another living creature. By carrying out digital experiments, scientists can relieve their subjects from eventual pain. For example, figuring out the exact chemical composition of a prosthesis can reduce the amount of pain the patient is suffering. And using a 'trial and errors' approach is definitely not acceptable in such situations.

Why use HPC instead of real experiments?

- **Safety**: Sometimes real-life experiments can be dangerous for the researchers, while digital experiments are safe. Examples include finding weak points in joints in a nuclear reactor environment or even more obviously – modelling the consequences of a disaster.

Why use HPC instead of real experiments?

- **No other choice**: Some experiments can only be digital due to their predictive and speculative nature: weather and climate prediction, seismic engineering, cosmology, financial predictions, etc.
- In seismic engineering for example, scientists simulate the propagation of seismic waves caused by a hypothetic earthquake and study the impact it would have on the foundations of future buildings.

Why use HPC instead of real experiments?

- Sometimes HPC is used to process the results of real-life experiments.
- For example, in oil discovery series of explosions are used to generate seismic waves, which get reflected at the contact surface between different media (oil and rock). The reflections are processed by HPC and they can reveal the location of underground (or underwater) reservoirs of oil and gas.
- This same technique is used in Computer Tomography (CT) scans performed at hostpitals in order to experiments reveal internal conditions in patients.

Moore's law

- Obviously HPC is quite useful and has a lot of applications. But it needs high-performance computing devices in order for it to work.
- It is obvious that computing devices get more powerful each day. But what is the rate of this increase and is there a limit?
- Moore's law is **the observation that over the history of computing hardware, the number of transistors on integrated circuits doubles approximately every two years.**

Moore's law

- Combined with other factors that help to increase performance, this means that the computing power is doubling each 18 months.
- Moore's law is the reason you always regret that you bought your laptop the minute you walk out of the store – they always begin selling better models on the next day!
- Gordon E. Moore, one of Intel co-founders, described this tendency in 1965 and predicted it will continue *"for at least ten more years"*. It has persevered with frightening accuracy for almost 50 years now!

Moore's law

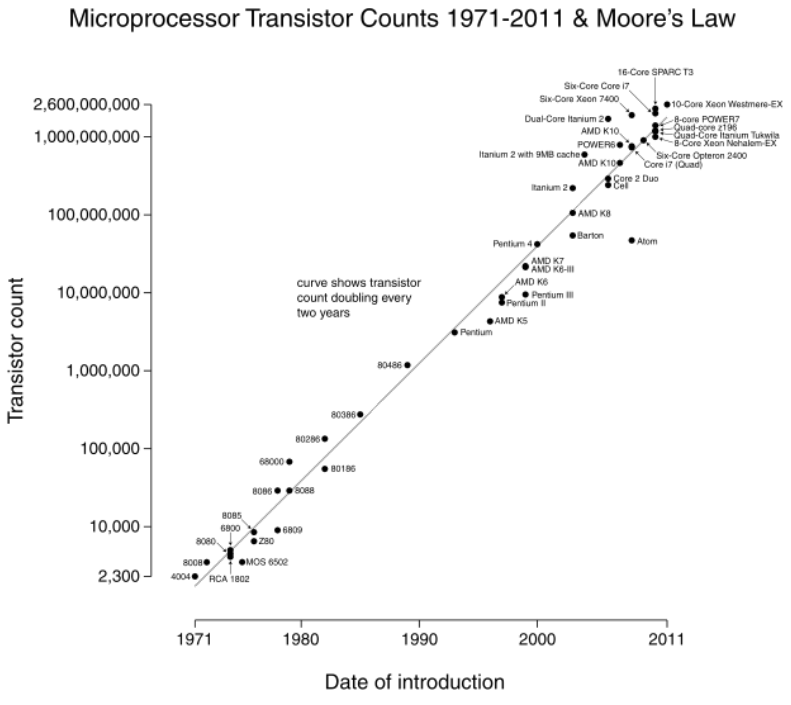

Figure: Moore's Law (source: Wikipedia)

Moore's law – until when?

- Simple calculations show that according to Moore's law by the end of year 2480 the number of transistors in a single integrated circuit will become larger than the number of atoms in the known Universe (at which point it should be renamed *Moore's Paradox*);
- Most likely this will not happen :-) The industry is hitting its limits and lagging behind Moore's law, now doubling the number of transistors every 3, instead of 2 years.

Moore's law – until when?

- Modern widespread CPUs are manifactured using 32 nm process.
- Having in mind that atoms in a Silicon crystal are 0.543 nm apart, this fixes the size of one transistor to around 60 atoms.
- For comparison, the size of the ribosome, one of the cellular organelles, is about 20 nm. Thus, the industry can now make transistors smaller than a biological cell.
- Certainly, this cannot go on much further.

HPC after Moore's Law

- Humankind will need HPC regardless of whether Moore's law holds or not.
- How are we then supposed to increase the number of calculations per unit time, given that we can no longer increase the performance of a single device?
- The answer is simple – by using more devices that will cooperatively perform our calculations *in parallel*.
- This is why parallel hardware and software exists.

Example of a parallel hardware—TITAN

- 18,688 nodes;
- Each node: AMD Opteron 16 cores @ 2.2GHz, NVIDIA Tesla K20, 38 GB RAM;
- This totals to 299,008 cores, 18,688 graphical accelerators and 710,144 GB RAM;
- Its area is about 500 m^2 and its power consumption is about 8 MWh;
- Its energy efficiency is 2150 MFLOPS/W
- How are we supposed to program this monster? That's what these lectures are about!

Flit's Taxonomy

- Computing architectures are classified in what is called the Flint's Taxonomy:
  - SISD (Single Instruction, Single Data) – one CPU is executing single sequence of isntructions over a single data set. This is the classical von Neumann architecture;
  - SIMD (Single Instruction, Multiple Data) – multiple CPUs are executing the same sequence of instructions, but each of them does so on different data set. This is called 'data parallel' architecture and is useful among other things for vector arithmetics, digital signal and image processing, etc.

Flit's Taxonomy

- MIMD (Multiple Instructions, Multiple Data) – multiple CPUs, each of them executing different sequence of instructions over different data sets. This corresponds to a totally parallel machine.
- MISD (Multiple Instructions, Single Data) – multiple CPUs, each of them executing different sequence of instructions over the same data set. The use of such architectures is limited mostly to high-availability systems.

The place of the memory

- Depending on how the multiple CPUs access their data, the parallel machines are classified as either *shared memory* or *distributed memory* machines or a combination of both;
- In a shared memory machine, all the CPUs have equal access to the same block of memory.
- Without care, this can lead to race conditions and non-determinism—different runs might produce different results, depending on some subtle timing conditions—and this is not something we want in general;

Shared memory machines

- In order to achieve deterministic behavior, a shared memory machine needs *synchronozation*—simplisticly put, this is a way for each of the CPUs to teporarily claim ownership of the memory so nobody else touches it, or some other way to ensure no race conditions exists.
- Pros:
  - Easier to program (somewhat);
  - Some algorithms are naturally parallel on data level;
- Cons: their size is limited to only several cores per machine;

Distributed memory machines

- Machines in which each CPU has access to its own memory block are called *distributed memory machines*.
- If order for the different CPUs are to achieve something in cooperative effort, they usually need *communication* between them;
- Pros:
  - Unlimited in size;
  - Some algorithms require little or no communication;
- Cons: communication is slow and when there are many nodes they might end up predominantly waiting on communication instead of actually computing.

Graphical accelerators

- A subclass of the shared memory machines are the *stream processors* found in graphical accelerators;
- They cannot work standalone—they need a central processor (CPU) to control them—send them data, send them instructions and reading the results. This data exchange is typical for CPU-GPU systems;
- Pros: being special-purpose hardware they offer high performance at low price;
- Cons: harder to program; the data exchange with the CPU takes time and can become problematic;

Hybrid systems

- As seen above, every architecture has its pros and cons;
- Hybrid systems combine some (or all) of the above architectures;
- The good news is that by combining them, we get a combination of their pros;
- The bad news is that we also get a combination of their cons;
- And the moral is that **when a hybrid system is to be programmed, you have to take into account all the cons, in order to achieve all pros**;
- Most of the high-performance machines today are hybrid;

Examples of different HPC systems

- Shared memory:
  - Multi-core/many-core CPUs;
- Accelerators and co-processors:
  - Stream processors (graphical accelerators);
  - Intel Xeon Phi;
- Distributed memory:
  - Grid computing networks (CERN, SETI@HOME);
  - Clusters (collections of independent nodes);
  - Supercomputers (tightly coupled, energy efficient and small footprint clusters);