Parallel Programming

Scientific Computing
Fall, 2019
Paul Gribble

1 What is parallel computing?

Simply put, parallel computing refers to performing multiple computations in *parallel*, i.e. simultaneously. By default most operations that take place on your computer happen in *serial*, that is, one at a time. These days CPU chips (even those on laptops) have multiple cores, which allow for some degree of parallel operations.

In principle, every time you double the number of CPU cores (or CPUs themselves), you can achieve something close to a halving of time to complete the operations. In practice however, there is always some overhead cost in carryout out the parallel computations. If the operations are at all lengthy however, the overhead cost is always worth it.

There are several types of parallel computing, which we'll talk briefly about (and which are listed below). What we'll get hands-on experience with is symmetric multiprocessing. This is the style of parallel computing where multiple CPUs, or multiple cores on a single CPU, share access to the same memory (RAM) store, and can carry out operations in parallel.

Today (Fall, 2015), it's still the case that not many programs take advantage of multiple cores. Operating systems, however, can take advantage of multiple cores through multithreading (see below), by assigning different threads to their own processing nodes.

Some programs like MATLAB (and some Apple applications) come with the ability to take advantage of multiple cores built-in. Due to the relative complexity of parallelizing serial code, however, most applications still operate in a serial fashion.

Parallel computing (Wikipedia)

2 Multi-threading

Modern operating systems like Mac OS X, Linux, and other Unix variants, provide the ability for programs to spawn multiple threads that execute independently of each other. The advantage of multithreading is that one process can do its work and other processes don't have to wait until the working process is done. This is used extensively for graphical user interfaces.

When you copy a file, you can still move your mouse around, you can still start other programs, you can still browse the web, while other things are happening simultaneously. Multithreading can occur on a single CPU with a single core. This isn't parallel computing per se, as multiple threads still have to share a single CPU processing unit to do their work—but the operating system manages the multiple threads so that the user has the impression that multiple things are happening at once.

See the wikipedia article for more details:

Multithreading (wikipedia)

3 Symmetric Multiprocessing (SMP)

These days modern computers ship with CPUs that have multiple cores, or even multiple CPUs each with multiple cores. At the time of writing these notes (Fall 2015) you can for example buy a Mac Pro desktop computer with two 6-core CPUs, for a total of 12 independent processing cores. With hyperthreading (see below) you get 24 processing cores, all for around $5,000—which seems like a lot, but just 10 years ago a computer cluster with 24 nodes would have cost around $75,000–$100,000.

When multiple CPUs and/or multiple CPU cores live in a single machine, they typically all share access to the same physical RAM (memory). These days all Apple desktops and laptops have CPUs with multiple cores. Generic PCs also ship with multiple CPUs and cores. Even smartphones (e.g. the iPhone, and Google's Nexus phone) come with multiple CPUs and multiple cores.

The great advantage of having multiple computing nodes in a single machine, is that unlike multithreading on a single CPU, where the operating system has to switch back and forth between each thread, with multiple CPUs/cores, each core can execute a different task in parallel with the others (i.e. at the same time).

A good analogy is the following. Imagine someone gives me 10 decks of playing cards, and each deck has been shuffled, and my task is to re-order each deck of cards. A computer with a single CPU/core is like a single person who is tasked with sorting all 10 decks of cards. I would have to sort them all, one at a time, one after the other, i.e. in serial. I could implement "multithreading" by sorting one deck for a few seconds, setting it aside, sorting the next deck for a few seconds, setting it aside, and so on, sorting bits of each one, one by one. It's still happening in serial though.

If I had access to other processing nodes, I could parallelize the task. So imagine instead of me sorting all 10 decks, I found 9 other people to help me. I gave them one of the 10 decks of cards, and I took one. Now we can all sort them, at the same time, in parallel. In theory it should take $\frac{1}{10}^{\text{th}}$ the time compared to me sorting them all myself, in serial. In fact though, there would be some overhead cost, for example at the beginning, when I would have to hand out each deck and give everyone their instructions, and then at the end, when I would have to collect all the sorted decks from each person. If the actual computational task being parallelized is time intensive, however, then these overhead costs would be minimal compared to the gain in speed I would achieve by parallelizing the task.

Symmetric multiprocessing, i.e. having multiple independent processing units share the same memory store, is advantageous compared to cluster or grid computing, where each processing node has its own memory. In the latter cases, there is a (sometimes relatively major) overhead cost involved in transferring data to the memory store for each processing unit, and back again to a head node. When this transfer happens over a network, as you can imagine, this would be way slower than if it happens on a common logic board on which all processing cores sit (as is with the case of symmetric multiprocessing).

Here is a wikipedia article on symmetric multiprocessing:

Symmetric multiprocessing (wikipedia)

4 Hyperthreading

Hyperthreading is a proprietary implementation by Intel for allowing modern CPUs to behave as if they have twice as many logical cores as physical cores. That is, if your CPU has two cores, hyperthreading implements a series of tricks at the operating system level, that interface with a series of tricks at the hardware layer (i.e. in the CPU itself) that results in the ability to address four "logical" cores.

Unlike multithreading, which is simply a software implementation at the operating system level, hyperthreading involves special implementations both at the operating system level and at the hardware level. Current Apple laptops and desktops all implement hyperthreading. Several generic PCs also implement hyperthreading.

For large, time consuming computations, hyperthreading won't actually double the computation speed, since at the end of the day, there are still x physical cores, even though hyperthreading pretends there are $2x$ logical cores. If however each computations is small, and doesn't last a long time, hyperthreading can end up giving you performance gains above and beyond regular multithreading, since it implements a number of efficiencies and tricks at the software and hardware layers.

For our purposes, hyperthreading is either there, or it isn't, and it's not something we will be fiddling with. Here is a wikipedia article on hyperthreading:

Hyper-threading (wikipedia)

5 Clusters

So far we have been talking about a single machine with multiple CPUs and/or multiple cores. Another way of implementing parallel computing is to connect together multiple machines, over a specialized local network. In principle one can connect as many machines as one likes, to achieve just about any level of parallelism one wants. Today's fastest supercomputers are in fact clusters of machines hooked together. The world's fastest supercomputer, as of today, October 2015, is the Tianhe-2, located in Guangzhou, China. It has 16,000 computer nodes, each one comprising two Intel Ivy Bridge Xeon CPUs and three Xeon Phi chips for a total of 3,120,000 cores (3.12 million cores).

Sharcnet is a Canadian cluster computing facility with several individual clusters, the largest of which has 8,320 cores. Western has access to the Sharcnet clusters, you just have to sign up for an account.

Many individual researchers also operate smaller clusters, for example with 8, or 12, or 24 machines hooked together.

A relatively recent development is the advent of gigantic server farms operated by private companies like Amazon and Google. Amazon's Elastic Compute Cloud allows individuals to spawn multiple "virtual" machines, and hook them together in networks and clusters, and run jobs on them. Cost is per machine and per unit time, and so one can essentially (1) define your own cluster and (2) pay for only those minutes that you actually use. It's a very flexible system that many researchers are beginning to utilize. Rhodri Cusack's lab, for example, uses cloud-based machines for brain imaging data analysis.

The obvious advantage of a cluster over a single SMP machine, is that one can add as many nodes onto the cluster (growing it as you go) to whatever size you want (provided you can pay for it). The disadvantage is that data transfer over a network can be slower than a SMP machine where CPU cores share the same RAM store. There is also added complexity in managing a cluster of machines, for example in configuring each one, and configuring a head node to manage all of the slave nodes. There is software out there that can organize this for you, for example Oracle Grid Engine, and others, but it's still not trivial and takes some investment of time to fully implement.

Computer cluster (wikipedia)

6 Grids

A grid is like a cluster, but the individual machines are not on a local network, but they can be anywhere on the internet. Sometimes multiple clusters are hooked together over the internet to form a grid. Sometimes a grid is composed of multiple individual machines, spread out over multiple labs, multiple Departments, Univerisities, or even countries. Sometimes grids are set up so that individual machines can be "taken over" as dedicated computational nodes. In other configurations, individual machines only process grid jobs during their downtime, when for example the user is not using the machine for something else. One way of setting this up is via a specialized screensaver. Wheneven the screensaver activates (which is an indication that the machine is not being used), the grid process starts up and processes grid jobs.

Two classic examples of grids are the SETI@home grid (searching for extra-terrestrial life in the universe) and the Folding@home grid (simulations of protein folding for disease research). In each case, anyone around the world can sign up their machine to join the grid and donate computer time, install some local software, and then anytime their computer is not busy, it is recruited by the grid to process data. As of now (Oct 2015) the Folding@home website shows that there are 8,067,858 CPUs active right now on the Folding@home grid.

There are also nefarious uses for grids, which are sometimes called Botnets. In this case, a virus infects a user's machine, installs a nefarious program, which lies dormant until a central machine somewhere on the internet activates it, for some nasty purpose (like a DDoS attack, or for sending spam). Your machine essentially becomes a sleeper cell.

Grid computing (wikipedia)

7 GPU Computing

In recent years computer engineers and software developers have teamed up, and have delivered software libraries that allow developers to utilize graphics cards for more general purpose computing (GPGPU Computing).

Graphics cards, unlike CPUs, have hundreds if not thousands of cores, each of which are typically used to process graphics for things like 3D games, video animation and scientific visualization. Each processing unit on a graphics card is a much simpler beast than the cores on CPU chips—but for some computational tasks, one doesn't need much complexity, and massive parallelism can be achieved by farming out general purpose computational tasks to the thousands of cores on a graphics card.

For example, today (Oct 2015) for around $5,000 one can purchase an NVidia Tesla GPU, which is a single graphics card, that has 12GB of GPU memory, 2880 cores, and has a processing power of 1.43 Tflops. As you can imagine, if your computational task is well suited to GPU processing, running it on 2880 cores will be quite a bit faster than running on 4, 8 or 12 cores (e.g. that you get with a modern dual 6-core CPU Mac Pro).

There are two major C/C++ software libraries that provide relatively high-level interfaces to performing general purpose computation on graphics cards

- CUDA (Nvidia proprietary)
- OpenCL (open)

MATLAB's Parallel Computing Toolbox has the ability to farm out some computations to NVidia CUDA-enabled GPUs, see this page for more info:

MATLAB GPU Computing Support for NVIDIA CUDA-Enabled GPUs

See this wikipedia page for more general information on GPGPU Computing:

GPGPU Computing

8 Types of Parallel problems

Multithreading is an example of *fine-grained parallelism* (many shared operations per second), in which the operating system manages (e.g. switches between) threads at a very fast rate, e.g. with each CPU clock cycle. This can thus happen many times per second. This is what your operating system does in the background, as you are interacting with your graphical user interface, surfing the web, playing music, processing video in the background, all the while copying files from one disk to another.

In another kind of fine-grained parallelism, multiple processes communicate with each other many, many times per second.

In *coarse-grained parallelism*, there are many, many independent threads/tasks, that rarely or never communicate with each other.

Finally, so-called *embarassingly parallel* problems are 100% independent operations, and never communicate with each other. Each process doesn't depend in any way on the result of another operation. This is the kind of parallelism that we will be talking about in this class.

9 MATLAB

MATLAB provides parallel computing via its Parallel Computing Toolbox (see below).

- MATLAB Parallel Computing Toolbox
- MATLAB Execute loop iterations in parallel using parfor
- MATLAB Getting Started with parfor
- MATLAB Parallel Computing Toolbox Examples

10 Shell scripts

Finally, one can parallelize tasks at the level of the shell, even if the programs you write/run aren't parallelized, using a tool like GNU Parallel (see below). Briefly, with GNU Parallel you can split a list of (ambarassingly parallel) tasks across multiple cores even if the program itself is serial in nature. See the GNU Parallel page below and the tutorial page for some examples. In our lab we use GNU Parallel to distribute subject-level brain imaging processing across multiple cores.

- GNU Parallel
- GNU Parallel tutorial