Flexible parallel processing using the R future and Python Dask packages

Chris Paciorek, Department of Statistics, UC Berkeley

1) This Tutorial

This tutorial covers the use of R's future and Python's Dask packages, relatively recent tools for parallelizing computions on a single machine or across multiple machines.

You should be able to replicate much of what is covered here provided you have Rand Python on your computer, but some of the parallelization approaches may not work on Windows.

This tutorial assumes you have a working knowledge of either R or Python, but not necessarily knowledge of parallelization in R or Python.

Materials for this tutorial, including the markdown files and associated code files that were used to create this document are available on Github at [https://github.com/berkeley-scf/tutorial-dask-future]. You can download the files by doing a git clone from a terminal window on a UNIX-like machine, as follows:

```
git clone https://github.com/berkeley-scf/tutorial-dask-future
```

See the `Makefile` for how to generate the html files in this tutorial.

2) Types of parallel processing

There are two basic flavors of parallel processing (leaving aside GPUs): distributed memory and shared memory. With shared memory, multiple processors (which I'll call cores) share the same memory. With distributed memory, you have multiple nodes, each with their own memory. You can think of each node as a separate computer connected by a fast network.

2.1) Some useful terminology:

- *cores*: We'll use this term to mean the different processing units available on a single node.

- *nodes*: We'll use this term to mean the different computers, each with their own distinct memory, that make up a cluster or supercomputer.
- *processes*: computational tasks executing on a machine; multiple processes may be executing at once. A given program may start up multiple processes at once. Ideally we have no more processes than cores on a node.
- *threads*: multiple paths of execution within a single process; the OS sees the threads as a single process, but one can think of them as 'lightweight' processes. Ideally when considering the processes and their threads, we would have the number of total threads across all processes not exceed the number of cores on a node.
- *forking*: child processes are spawned that are identical to the parent, but with different process IDs and their own memory.
- *sockets*: some of R's parallel functionality involves creating new R processes (e.g., starting processes via *Rscript*) and communicating with them via a communication technology called sockets.

2.2) Shared memory

For shared memory parallelism, each core is accessing the same memory so there is no need to pass information (in the form of messages) between different machines. But in some programming contexts one needs to be careful that activity on different cores doesn't mistakenly overwrite places in memory that are used by other cores.

Threading

Threads are multiple paths of execution within a single process. If you are monitoring CPU usage (such as with *top* in Linux or Mac) and watching a job that is executing threaded code, you'll see the process using more than 100% of CPU. When this occurs, the process is using multiple cores, although it appears as a single process rather than as multiple processes.

Note that this is a different notion than a processor that is hyperthreaded. With hyperthreading a single core appears as two cores to the operating system.

2.3) Distributed memory

Parallel programming for distributed memory parallelism requires passing messages between the different nodes. The standard protocol for doing this is MPI, of which there are various versions, including *openMPI*.

2.4) Other type of parallel processing

We won't cover either of these in this material.

GPUs

GPUs (Graphics Processing Units) are processing units originally designed for rendering graphics on a computer quickly. This is done by having a large number of simple processing units for massively parallel calculation. The idea of general purpose GPU (GPGPU) computing is to exploit this capability for general computation.

Most researchers don't program for a GPU directly but rather use software (often machine learning software such as Tensorflow or PyTorch) that has been programmed to take advantage of a GPU if one is available.

Spark and Hadoop

Spark and Hadoop are systems for implementing computations in a distributed memory environment, using the MapReduce approach.

Note that Dask provides a lot of the same functionality as Spark, allowing one to create distributed datasets where pieces of the dataset live on different machines but can be treated as a single dataset from the perspective of the user.

3) Python's Dask package

Please see the Python dask materials.

4) R future package

Please see R future materials.