Outline

- Multi-core computing, distributed computing
- Multi-core computing tools
- Distributed computing tools
Parallel Programming

Parallel algorithms can be different in the following two cases:

- **Shared Memory Model (Multiple cores)**
  - Independent L1 cache
  - Shared/independent L2 cache
  - Shared memory

- **Distributed Memory Model**
  - Multiple processes in single machine
  - Multiple computers
Shared Memory Model (Multiple cores)

- Shared memory model: each CPU can access the same memory space
- Programming tools:
  - C/C++: openMP, C++ thread, pthread, intel TBB, ...
Parallel for loop in OpenMP

```c
#pragma omp parallel for private(i)
for(i=0; i<w_size; i++)
g[i] = w[i] + g[i];
```
Two types of shared memory model:
1. Uniform Memory Access (UMA)
2. Non-Uniform Memory Access (NUMA)
Distributed Memory Model

- Programming tools: MPI, Hadoop, Spark, ...

(Figure from http://web.sfc.keio.ac.jp/rdv/keio/sfc/teaching/architecture/computer-architecture-2013/lec09-smp.html)
Programming for distributed systems

- Low-level programming:
  - Socket programming
- Need to write code to send/receive sockets (messages) through network
  - Initialize socket in each processor
  - Sender send message “sendto()”
  - Receiver get message “recvfrom()”
- Use this only when you need very low level control

(Figure from https://people.eecs.berkeley.edu/~culler/WEI/labs/...
Message Passing Interface (MPI)

- A easier (and standard) way to pass messages in distributed systems
- C, python, ...
- Several types of “Collective Communication Routines”
  - Broadcast
  - Reduce
  - Gather, Allgather
  - ...

```python
import numpy
from mpi4py import MPI
comm = MPI.COMM_WORLD

rank = comm.Get_rank()
rankF = numpy.array(float(rank))
total = numpy.zeros(1)
comm.Reduce(rankF, total, op=MPI.MAX)
```
**Message Passing Interface (MPI)**

- **MPI_Broadcast**: Broadcasts a message to all other processes of that group

![Diagram showing task 0 broadcasting a message to task 1, task 2, and task 3](image)
 MPIReduce: Reduces values on all processes to a single value
   (Can specify an operator, e.g., +, −, ×, /)
Message Passing Interface (MPI)

- Many other operations.
MapReduce Paradigm

- Framework for parallel computing
- Handles parallelization, data distribution, load balancing & fault tolerance
- Allows once to process huge amounts of data (terabytes & petabytes) on thousands of processors
MapReduce Paradigm

- Framework for parallel computing
- Handles parallelization, data distribution, load balancing & fault tolerance
- Allows once to process huge amounts of data (terabytes & petabytes) on thousands of processors
- Google
  - Original implementation
- Apache Hadoop MapReduce
  - Most common (open-source) implementation
  - Built to specs defined by Google
- Amazon MapReduce
  - On Amazon EC2
MapReduce concept

- **Map**
  - Grab the relevant data from the source
  - User function gets called for each chunk of input (key, value) pairs
- **Reduce**
  - Aggregate the results
  - Gets called for each unique key
MapReduce concept

- **Map**: (input shard) \(\rightarrow\) (intermediate key, intermediate value)
  - Automatically partition input data to each computer
  - Group together all intermediate values with the same intermediate key
  - Pass to the reduce function

- **Reduce**: (intermediate key, intermediate value) \(\rightarrow\) result files
  - Input: key, and a list of values
  - Merge these values together to form a smaller set of values
  - Automatically partition the reducers distributedly
MapReduce: the complete picture

(Figure from https://www.cs.rutgers.edu/~pxk/417/notes/content/16-mapreduce-slides.pdf)
Example

- Count number of each word in a collection of documents
- Map: parse data, output each word with a count (1)
- Reduce: sum together counts for each key (word)
- Mapper:
  
  ```
  map(key, value):
  // key: document name, value: document contents
  for each w in value:
    output (w, '1')
  ```

- Reducer:
  
  ```
  reduce(key, values):
  // key: a word; values: a list of counts
  for each v in values:
    result += Int(v)
  output(result)
  ```
Example

**Mapper:**

```python
for line in sys.stdin:
    line = line.strip().split()
    for word in words:
        print '%s\t%s'%(word,'1')
```

**Reducer:**

```python
word2count = {}
for line in sys.stdin:
    line = line.strip()
    word, count = line.split(\'\t\', 1)
    word2count[word] = word2count[word]+count
for word in word2count.keys():
    print '%s\t%s'%(word,word2count[word])
```
Example

It will be seen that this mere painstaking burrower and grub-worm of a poor devil of a Sub-Sub appears to have gone through the long Vaticans and street-stalls of the earth, picking up whatever random allusions to whales he could anyways find in any book whatsoever, sacred or profane. Therefore you must not, in every case at least, take the higgledy-piggledy whale statements, however authentic, in these extracts, for veritable gospel cetology. Far from it. As touching the ancient authors generally, as well as the poets here appearing, these extracts are solely valuable or entertaining, as affording a glancing bird's eye view of what has been promiscuously said, thought, fancied, and sung of Leviathan, by many nations and generations, including our own.

(Figure from https://www.cs.rutgers.edu/~pxk/417/notes/content/16-mapreduce-slides.pdf)
Hadoop Distributed File System (HDFS)

- The Hadoop Distributed File System (HDFS) is designed to store very large data sets on multiple servers.
- To use hadoop MapReduce, input/output files are in HDFS
Hadoop Ecosystem

Apache Hadoop Ecosystem

Ambari
Provisioning, Managing and Monitoring Hadoop Clusters

Sandbox
Data Exchange

Hive
SQL Query

Flume
Log Collector

Mahout
Machine Learning

Oozie
Workflow

Pig
Scripting

R Connectors
Statistics

HBase
Columnar Store

YARN Map Reduce v2
Distributed Processing Framework

HDFS
Hadoop Distributed File System
Spark

- Hadoop: Need to read/write HDFS for all the mapper/reducer
  Main bottleneck is disk reading time
  Not suitable for machine learning (iterative computing)
- Spark: Also a MapReduce framework with
  In memory data flow
  Optimize for multi-stage jobs
Spark

- Machine Learning using Spark: MLLib
Parameter Server

- A concept mainly for parallelizing machine learning algorithms (deep learning)
- Maintain a set of “shared parameters”
- Local machine communicate with parameter server to get the latest parameters

\[ w' = w - \eta \Delta w \]
Coming up

- Final Project Report

Questions?