Contents

- Concurrency vs Parallelism
- Ideal Parallelism
- Example Parallelisation Approaches
- Memory Coalescing (Matrix Multiplication)
- Several Example Parallel Problems/Techniques
- Parallelism at Multiple Levels
- Measuring Scaling Performance

*time permitting:*
- The KubeNow Project
- The H2O Platform
Concurrent vs Parallelism

- **Concurrency/Multithreading**: is when two or more tasks can start, run, and complete in overlapping time periods. It doesn't necessarily mean they'll ever be running at the same instant. Eg. multitasking on a single-core machine.
- **Parallelism**: is when tasks literally run at the same time, eg. on a multicore processor.

Quoting *Oracle's Multithreaded Programming Guide*:

- **Concurrency**: A condition that exists when at least two threads are making progress. A more generalized form of parallelism that can include time-slicing as a form of virtual parallelism.
- **Parallelism**: A condition that arises when at least two threads are executing simultaneously.

Java Concurrency and Multithreading Tutorial:
http://tutorials.jenkov.com/java-concurrency/index.html

https://stackoverflow.com/a/1050257
Ideal Parallelism

- An ideal parallel computation can be immediately divided into completely independent parts
  - “Embarrassingly parallel”
  - “Naturally parallel”
- No special techniques or algorithms required

Parallelisation Approaches

- **Model Parallelism**: different machines in the distributed system are responsible for the computations in different parts of a single network - for example, each layer in the neural network may be assigned to a different machine.

- **Data Parallelism**: different machines have a complete copy of the model; each machine simply gets a different portion of the data, and results from each are somehow combined.

Memory Coalescing 1/2

- 2D matrices are represented as vectors in memory
- Take two **square** Matrices \( \mathbf{A} \) and \( \mathbf{B} \), we want to: \( \mathbf{A} \mathbf{B} \)
- When Multiplying Matrices, we make expensive moves in \( \mathbf{B} \)

\[
\begin{pmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{pmatrix}
\]

\[
\begin{pmatrix}
a1 & a2 & a3 \\
a4 & a5 & a6 \\
a7 & a8 & a9
\end{pmatrix}
\]

\[
\begin{pmatrix}
b1 & b2 & b3 \\
b4 & b5 & b6 \\
b7 & b8 & b9
\end{pmatrix}
\]

This jump is expensive
Memory Coalescing 2/2

- Square Matrix Multiplication in Java
- Transpose B
- Move row by row instead of column by column
HPC Usage Example 1:

- **Task - Permutation test**
  - Data: one dataset -> created **10k** datasets by randomising the variables (not the outcome)
  - Run linear regression with 10 fold cross validation on each dataset to obtain a score (e.g. RMSE)

- **Performance:**
  - When done sequentially, using **1 core**, 10k datasets took > **83 hours**
  - When parallelising the task, using a HPC cluster, 10k datasets took < **83 minutes**
HPC Usage Example (The ChemDistiller Project):

- Task - Compute Fingerprints for ~130 Million Chemical Compounds
- Data: 13k files, each containing 10k compounds
  - Input compound representation: SMILES
- Performance:
  - Using 1 core, in average, 1 file takes ~ 8 hours, up to 24 hours for files with larger molecules
  - When done sequentially, using 1 core, the 130M compounds would finish in > 11 YEARS (1-2 years using 8 cores)
  - When parallelising the task, using a HPC cluster, 130M compounds took ~ 22 days

ChemDistiller: an engine for metabolite annotation in mass spectrometry
https://academic.oup.com/bioinformatics/article/34/12/2096/4852828
Cross Validation

- A model is trained using \( k-1 \) of the folds as training data
- The resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy).
Random Forest

- Random forest algorithm is a supervised classification algorithm. As the name suggests, this algorithm creates the forest with a number of trees
  - In general, the more trees in the forest the more robust the forest looks like.
- Bootstrapping algorithm with Decision tree (CART) model.
- Say, we have \( n \) observations in the complete population with \( m \) variables.
- Random forest tries to build multiple CART models with different samples and different initial variables.
  - For instance, it will take a random sample of \( i \) observations and \( j \) randomly chosen initial variables to build a CART model (\( j << m \)).
  - It will repeat the process (say) \( k \) times and then make a final prediction on each observation.
  - Final prediction can simply be the mean (or mode) of each prediction.
Random Forest
Random Forest
Random Forest
Analysing Very Large Files

- **Map** functions can run in parallel and pass their results to **Reduce** functions.
- Results are output in sorted order by the keys created by the reduce function.

- **Sorting very large files?**

[Image of a diagram showing the MapReduce process]

http://selkie.macalester.edu/csinparallel/modules/IntroWMR/build/html/MapReduceIntro/MapReduceIntro.html
Neural Nets Parameter Averaging

1. Initialize the network parameters randomly based on the model configuration
2. Distribute a copy of the current parameters to each worker
3. Train each worker on a subset of the data
4. Set the global parameters to the average the parameters from each worker
5. While there is more data to process, go to step 2

- Steps 2 through 4 are demonstrated in the diagram
- $W$ represents the parameters (weights, biases) in the neural network
- Subscripts are used to index the version of the parameters over time, and where necessary for each worker machine
Correlation Matrix

\[ r = \frac{1}{n-1} \sum_{i=1}^{n} \left( \frac{x_i - \bar{x}}{s_x} \right) \left( \frac{y_i - \bar{y}}{s_y} \right) \]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
<th>N</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>probe</td>
<td>4308.8</td>
<td>220.1</td>
<td>130.6</td>
<td>75</td>
<td>54.2</td>
<td>195.5</td>
<td>1024.8</td>
<td>1221.5</td>
<td>82.1</td>
<td>57.5</td>
<td>67.1</td>
<td>174</td>
<td>55.8</td>
<td>143.6</td>
</tr>
<tr>
<td>2</td>
<td>1007_s_at</td>
<td>79.5</td>
<td>95.7</td>
<td>178.8</td>
<td>185.8</td>
<td>144.7</td>
<td>113.1</td>
<td>150.2</td>
<td>157.5</td>
<td>158.4</td>
<td>140.7</td>
<td>196.1</td>
<td>123.7</td>
<td>254.8</td>
<td>130.4</td>
</tr>
<tr>
<td>3</td>
<td>1032_s_at</td>
<td>60.9</td>
<td>14.9</td>
<td>1.4</td>
<td>28.7</td>
<td>4.7</td>
<td>25.2</td>
<td>9</td>
<td>3.5</td>
<td>21.9</td>
<td>3.9</td>
<td>17.7</td>
<td>23.7</td>
<td>39.7</td>
<td>4.1</td>
</tr>
<tr>
<td>4</td>
<td>1211_s_at</td>
<td>201.3</td>
<td>28.4</td>
<td>74.1</td>
<td>60.3</td>
<td>78.7</td>
<td>54.5</td>
<td>60.1</td>
<td>11.8</td>
<td>56.8</td>
<td>83.8</td>
<td>80.1</td>
<td>81.9</td>
<td>72.7</td>
<td>91.9</td>
</tr>
<tr>
<td>5</td>
<td>1255_s_at</td>
<td>5</td>
<td>129.8</td>
<td>177.8</td>
<td>291.1</td>
<td>192.4</td>
<td>20</td>
<td>235.4</td>
<td>5.6</td>
<td>54.9</td>
<td>56.6</td>
<td>63.6</td>
<td>61.2</td>
<td>300</td>
<td>41.1</td>
</tr>
<tr>
<td>6</td>
<td>1294_s_at</td>
<td>114.6</td>
<td>57.1</td>
<td>40.1</td>
<td>37.1</td>
<td>44.2</td>
<td>10.9</td>
<td>40.4</td>
<td>28.3</td>
<td>56.3</td>
<td>31.6</td>
<td>38.6</td>
<td>127.8</td>
<td>44</td>
<td>79.4</td>
</tr>
<tr>
<td>7</td>
<td>1316_at</td>
<td>113</td>
<td>52.7</td>
<td>153.1</td>
<td>106.3</td>
<td>63.5</td>
<td>103.6</td>
<td>121.6</td>
<td>57.8</td>
<td>26.1</td>
<td>33.2</td>
<td>27</td>
<td>33.5</td>
<td>24</td>
<td>17.3</td>
</tr>
<tr>
<td>8</td>
<td>1332_at</td>
<td>12.1</td>
<td>17.9</td>
<td>1.7</td>
<td>1</td>
<td>2.2</td>
<td>11.2</td>
<td>17.2</td>
<td>16.1</td>
<td>33.2</td>
<td>58.1</td>
<td>27.4</td>
<td>14.3</td>
<td>46.6</td>
<td>37.9</td>
</tr>
<tr>
<td>9</td>
<td>1409_s_at</td>
<td>111.8</td>
<td>154.4</td>
<td>26.3</td>
<td>18.8</td>
<td>166.4</td>
<td>222.8</td>
<td>81</td>
<td>15.7</td>
<td>64.1</td>
<td>29</td>
<td>18</td>
<td>408.7</td>
<td>28.9</td>
<td>148.7</td>
</tr>
<tr>
<td>10</td>
<td>1431_at</td>
<td>18.8</td>
<td>15.7</td>
<td>16.2</td>
<td>8.6</td>
<td>8.2</td>
<td>66</td>
<td>10.3</td>
<td>15.1</td>
<td>33</td>
<td>14.2</td>
<td>79.8</td>
<td>12.5</td>
<td>15.1</td>
<td>7.4</td>
</tr>
<tr>
<td>11</td>
<td>1438_s_at</td>
<td>2.4</td>
<td>51.9</td>
<td>5.2</td>
<td>9.4</td>
<td>7.2</td>
<td>8.2</td>
<td>8.3</td>
<td>38.5</td>
<td>93.7</td>
<td>193.6</td>
<td>31.5</td>
<td>102.6</td>
<td>59.1</td>
<td>100.8</td>
</tr>
<tr>
<td>12</td>
<td>1487_at</td>
<td>39</td>
<td>86.9</td>
<td>96</td>
<td>52.1</td>
<td>55.3</td>
<td>67.8</td>
<td>92</td>
<td>51.4</td>
<td>63</td>
<td>59.1</td>
<td>160.7</td>
<td>106.4</td>
<td>54.8</td>
<td>54.4</td>
</tr>
<tr>
<td>13</td>
<td>1484_s_at</td>
<td>30</td>
<td>18.8</td>
<td>33</td>
<td>49</td>
<td>54</td>
<td>35.3</td>
<td>44.9</td>
<td>35.3</td>
<td>26.1</td>
<td>22.9</td>
<td>19.8</td>
<td>18.7</td>
<td>19.8</td>
<td>17</td>
</tr>
<tr>
<td>14</td>
<td>1552290_s_at</td>
<td>33.4</td>
<td>55.5</td>
<td>57.8</td>
<td>33.1</td>
<td>52</td>
<td>80.2</td>
<td>61.7</td>
<td>117</td>
<td>182.2</td>
<td>79.1</td>
<td>185.9</td>
<td>187.1</td>
<td>137.1</td>
<td>148.8</td>
</tr>
<tr>
<td>15</td>
<td>1552257_s_at</td>
<td>109.2</td>
<td>78.6</td>
<td>70</td>
<td>83.8</td>
<td>62.7</td>
<td>57.1</td>
<td>73.4</td>
<td>245.7</td>
<td>129.6</td>
<td>113.4</td>
<td>131.9</td>
<td>117.9</td>
<td>112.2</td>
<td>175.1</td>
</tr>
<tr>
<td>16</td>
<td>1552258_at</td>
<td>31.2</td>
<td>8.3</td>
<td>5.3</td>
<td>20.3</td>
<td>7</td>
<td>20.7</td>
<td>35</td>
<td>22.7</td>
<td>22.6</td>
<td>12.2</td>
<td>11.3</td>
<td>24.9</td>
<td>27.3</td>
<td>18.7</td>
</tr>
<tr>
<td>17</td>
<td>15523361_at</td>
<td>123.3</td>
<td>5.8</td>
<td>5.4</td>
<td>11.1</td>
<td>13.7</td>
<td>25.1</td>
<td>22.6</td>
<td>15.8</td>
<td>23.5</td>
<td>9.8</td>
<td>26.8</td>
<td>15.8</td>
<td>15.8</td>
<td>24.6</td>
</tr>
</tbody>
</table>
Parallelism at Multiple Levels

- Sometimes the problem at hand is parallelizable at more than one level
- A typical example is when we want to run a parallel algorithm several times
- An example is to run RandomForest on thousands of datasets
- Which level do we choose?
Measuring Scaling Performance

Strong Scaling:
- Fixed data size (ex: 10000 datasets)
- Change number of parallel processes
- Check performance

Weak Scaling:
- Variable data size (ex: 10, 100, 1000, 10000 datasets)
- Number of parallel processes changes with data size
- Check performance

Metrics:
- Speedup \( S(n) = \frac{T_1}{T_n} \)
- Scaling Efficiency \( E(S) = \frac{T_1}{nT_n} = \frac{S(n)}{n} \)
Summary

1. Embarrassingly Parallel problems are everywhere
2. It is a mindset .. a way of thinking about problem solving
3. Plenty of platforms
4. Sometimes it is a matter of mapping the problem into a format that a parallel platform can process
5. Many real life examples show it is worth the effort!
● A cloud agnostic platform for microservices, based on Docker and Kubernetes
● Fast Kubernetes operations
● Helps you in lifting your final application configuring DNS records and distributed storage

https://github.com/kubenow/KubeNow
Deploy PhenoMeNal with KubeNow

KubeNow

kn-destroy

kn-apply

kn-az,gce

kn-ansible

kn-ssh

kn-helm

https://goo.gl/jZx5sn
```bash
#!/bin/bash
export PYTHONPATH=./

#numberW=(1 5 10 25 50 100 200 250 400 500) #for 2 spectra per file
#numberW=(1 5 10 20 25 40 50) #for 20 spectra per file
#numberW=(100) #for 10 spectra per file

for number in \"${numberW[@]}\"; do
echo "No of Parallel Tasks = $number"
echo $PWD
#uptime=$( $( TIMEFORMAT=\"%lU\"; luigi --module workflow ProcessDatasets --scheduler-host luigi.default --workers $number)"
#echo $uptime >> times.csv
START=$(date +%s)
luigi --module batman DoBatman --scheduler-host luigi.default --workers $number
END=$(date +%s)
DIFF=$(( $END - $START ))
END=$(date +%Y-%m-%d-%H-%M-%s)
echo "With $number of workers, jobs ended at $END. It took $DIFF seconds" >> times.csv
#echo $DIFF >> times.csv

#remove the BATMAN running folders to enable next run
rm -rf $(ls -l \"NMRun\" -I "results*" -O grep '[0-9]-[0-9]' )
#END=$(date +%s)
#mv results results-workers-$END
if [ "$(ls results/*pdf | wc -l)" == 1000 ]; then rm -rf results; else mv results results-workers-$number-workers-$END; fi

cd ..
#number=$((number + 10))
#rm -f data/*.out
#rm -f results.csv
```
Kubernetes Dashboard
H2O.ai

- Founded in 2012, Mountain View, CA Stanford Math & Systems Engineers
- It is produced by the company *H2O.ai* (formerly *0xdata*)
- Open Source Software
- Ease of Use via Web Interface or API
- Cutting Edge Machine Learning Algorithms
- R, Python, Scala, Spark & Hadoop Interfaces Distributed Algorithms Scale to Big Data
- Simple deployment without intermediary transformations
- In-Memory Parallel Processing

- [https://github.com/h2oai](https://github.com/h2oai)
- [http://docs.h2o.ai](http://docs.h2o.ai)
- [https://www.stat.berkeley.edu/~ledell/docs/h2o_hpccon_oct2015.pdf](https://www.stat.berkeley.edu/~ledell/docs/h2o_hpccon_oct2015.pdf)
H2O Community

Companies Using H2O.ai

<table>
<thead>
<tr>
<th>Year</th>
<th>2015</th>
<th>2016</th>
<th>Now</th>
<th>2017 Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>3,810</td>
<td>6,427</td>
<td>10,281</td>
<td>13,000</td>
</tr>
</tbody>
</table>

169 of the 500 Fortune Companies Use H2O

8 of Top 10 Banks

7 of Top 10 Insurance Companies

4 of Top 10 Healthcare Companies

H2O.ai Users

<table>
<thead>
<tr>
<th>Year</th>
<th>2015</th>
<th>2016</th>
<th>Now</th>
<th>2017 Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>38,257</td>
<td>54,163</td>
<td>97,620</td>
<td>110,000</td>
</tr>
</tbody>
</table>
H2O: Current Algorithm Overview

**Statistical Analysis**
- Linear Models (GLM)
- Cox Proportional Hazards
- Naïve Bayes

**Ensembles**
- Random Forest
- Distributed Trees
- Gradient Boosting Machine
- R Package - Super Learner Ensembles

**Deep Neural Networks**
- Multi-layer Feed-Forward Neural Network
- Auto-encoder
- Anomaly Detection
- Deep Features

**Clustering**
- K-Means

**Dimension Reduction**
- Principal Component Analysis
- Generalized Low Rank Models

**Solvers & Optimization**
- Generalized ADMM Solver
- L-BFGS (Quasi Newton Method)
- Ordinary Least-Square Solver
- Stochastic Gradient Descent

**Data Munging**
- Integrated R-Environment
- Slice, Log Transform
H2O Scalability

Parallel Scalability
(for 64 epochs on MNIST, with "0.83%" parameters)

Training Time in minutes

H2O Nodes

100
75
50
25
0
1 2 4 8 16 32 63

2.7 mins

Speedup

4.00
3.00
2.00
1.00
0.00
1 2 4 8 16 32 63

H2O Nodes

(4 cores per node, 1 epoch per node per MapReduce)
Thank you