Raspberry Pi Lab II – Distributed Computing

Distributed Computing involves the breaking down a computational problem into several parallel tasks to be completed by two or more computers in a network which form a distributed system. This combines the computational power of several computers to solve large problems which involve the processing of large data or require a huge number of iterations.

In this lab, we will experiment with a classical problem, sorting. Sorting can be easily parallelized using Mergesort[1].

Speedup in Distributed Computing

Amdahls law[2] provides a way to estimate the speedup of running a program on several machines. It requires a single parameter p, which describes the fraction of the code that can be parallelized. Given n machines, the speedup S compared with running the program on a single machine is calculated as:

$$S = \frac{1}{1-p+\frac{p}{n}}$$

Example: Consider a program of which 90% can be parallelized (p=0.9). Then the speedup of running the program on 3 machines is $\frac{1}{1-0.9+\frac{0.9}{3}} = 2.5$.

Distributed Mergesort

You can download Python code for distributed Mergesort from https://goo.gl/fKnN1L. The provided code breaks the data to be sorted into n partitions (with n being the number of machines involved), which each locally sort their data, then send their results back, where they are finally merged.

[1] https://en.wikipedia.org/wiki/Merge_sort
[2] https://en.wikipedia.org/wiki/Amdahl's_law

MergeSort.py implements the classical Mergesort algorithm, while Merge1.py creates an array of 100000 elements an lets Mergesort run on it. You can run Merge1.py from the command line by executing

```
        sudo python Merge1.py
```

To distribute the task to two RPis, first take note of the IP address of the server (you may want to set a static address, if there is no DHCP server). Then, run MergeServer.py on the server.

Insert the IP of the server into MergeClient.py. When the server prints "Waiting for client..." you can, run MergeClient.py on the second Pi.

Tasks

1. Estimate the speedup of the distributed Mergesort code when distributing the sorting of 10000, 50000, 100000, 150000 and 200000 numbers using 2 machines. ***Hint**: To determine the fraction of the Mergesort that can be parallelized, remember that sorting has complexity O(n*log n), while in the end, there is a centralized merge step for the individual results, which runs in O(n) time.*

2. Measure the runtime of the local Mergesort and the distributed Mergesort using the problem sizes described above.

3. Plot the runtimes using e.g. Excel.

4. Try to explain any differences to your theoretical calculation from (1).

5. Team up with another group in order to run Mergesort distributed over 3 and 4 machines. Again, take note of the runtimes and compare them with the theoretically possible speedup.

Note: Please leave the desktop machines in their original state when finishing the lab, i.e., plug back any monitor/Ethernet/other cables.

Further Reading

- A Comparison of Parallel Sorting Algorithms on Different Architectures (http://dl.acm.org/citation.cfm?id=892860)