EE 382V: Parallel Algorithms **Summer 2017**

Lecture 1: June 9

Lecturer: Vijay Garg *Scribe: Hector Jesus Acosta Garcia*

1.1 Memory Hierarchy

Memory Hierarchy can be used to discuss performance issues in computer programs. Throughout this course, an emphasis is made on the time and work complexity of algorithms, however there are other factors that can affect real world performance of processes executing different algorithms. After an algorithm is implemented, it is important to verify cache behavior and other lower level.

From faster to slower the memory hierarchy follows:

- Processor Registers
- Processor caches (L1, L2, L3, . . .)
- Main memory
- Disk

Accessing data on a lower level in the above hierarchy orders of magnitude of slowdown.

1.2 PRAM

PRAM is the parallel analogous to RAM for parallel algorithm. It is assumed to be shared by more than one processor, and at any point in time a processor can read or write to any memory location.

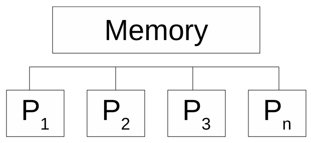

Figure 1.1: A PRAM representation

All processors work in lock-step. Occasionally, a ”BARRIER” instruction is needed, this is similar tu syncThreads() in CUDA.

In PRAM, the following are valid Read/write conflict resolution strategies:

EREW	Exclusive Reads Exclusive Writes	Most flexible. Every memory location can be read or written to by only one processor at the time. Algorithms designed for this conflict strategy, work for all conflict strategies.
CREW	Concurrent Reads Exclusive Writes	Closest to reality.
ERCW	Exclusive Reads Concurrent Writes	Not interesting.
CRCW	Concurrent Reads Concurrent Writes	Fastest model.

For concurrent writes, PRAM can be further defined as:

- Arbitrary: Choose arbitrarily when two concurrent writes are performed to the same memory location
- Common: All processors must write the same value.
- Priority: Processor id is used to resolve conflicts.

1.3 Network

In a parallel computing system, there are several types of Network topologies that can be used.

- Mesh: data is transmited at 1 unit per hop
- Torus: Mesh with wraparound link at the edges
- Completely connected: Not efficient or scalable.

1.3.1 Hypercube

In a hypercube with n^d nodes, the maximum distance between two nodes is d.

If each node can be labeled by a d-bit string, two nodes are connected if they differ in exactly 1 bit.

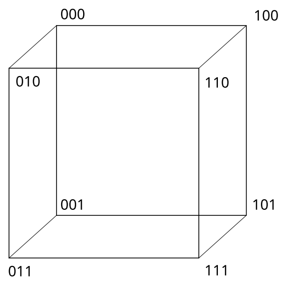

Figure 1.2: An 8 node hypercube with $d = 3$

1.4 Scalability

1.4.1 Strong Scalability

As the number of cores increases, the time taken to solve the same problem decreases.

1.4.2 Weak Scalability

For the same problem $problemSize/numberOfCores = Constant$

1.4.3 Amadahl's Law

Amadahl's law is used to calculate the maximum speedup possible by improving the parallel portion of an algorithm. Assume a program can be divided into sequential part s and non-sequential part $1 - s$, and n is the number of processors. Then, the speedup can be given by:

$$\begin{aligned}
speedup &= \frac{T_{seq}}{T_{parallel}} \\
T_{parallel} &\geq s(T_{seq}) + \frac{(1-s)T_{seq}}{n} \\
T_{parallel} &\geq (s + \frac{1-s}{n})T_{seq} \\
\frac{T_{seq}}{T_{parallel}} &\leq \frac{1}{s + \frac{1-s}{n}}
\end{aligned}$$

1.4.4 Gustafson's Law

Gustafson argues that the problem size is not constant and if the parallel portion of a program is improved, the problem size may also be increased, while solving the problem in the same time.

1.5 Programming Languages

In class, we briefly went over the following example programs (found in the course's github page).

- FooBar.java
- Fibonacci1.java
- Fibonacci2.java
- Fibonacci3.java