Practical Parallelism

Jaimie Murdock

IU Cognitive Science Program
810 Eigenmann Hall
jammurdo@indiana.edu

November 17, 2011

Motivation

Multiprocessing has moved from HPC-only to SMP

Motivation

Multiprocessing has moved from HPC-only to SMP

Cores are cheap:

- **2** – Intel Celeron E3400 – $47
- **4** – AMD Athlon II X4 631 – $90
- **6** – AMD Phenom II X6 1035T – $135
- **8** – AMD FX-8120 – $210

Motivation

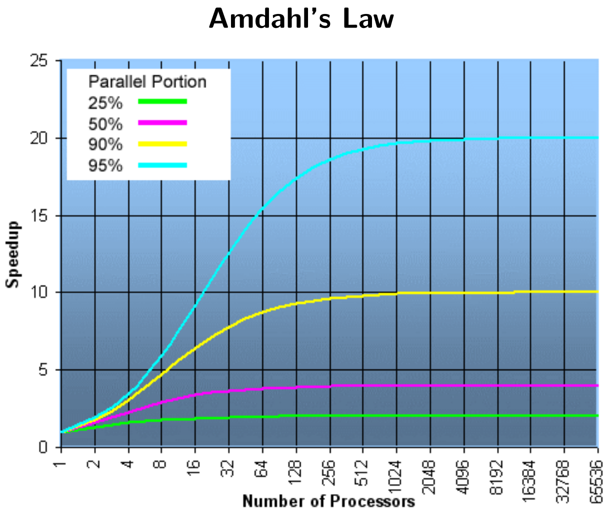
Multiprocessing has moved from HPC-only to SMP

Cores are cheap:

- **2** – Intel Celeron E3400 – $47
- **4** – AMD Athlon II X4 631 – $90
- **6** – AMD Phenom II X6 1035T – $135
- **8** – AMD FX-8120 – $210

The rise of the cloud — clusters for everyone

How fast?


Amdahl's Law
Parallel Portion
25%
50%
90%
95%
Speedup
0
5
10
15
20
25
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Number of Processors

[fragile]

Painfully Parallel Problems

(or: how to use C211 in an interview)

Painfully Parallel Problems

(or: how to use C211 in an interview)
Given a function that is both commutative and associative (e.g., $+$ or $*$)

Painfully Parallel Problems

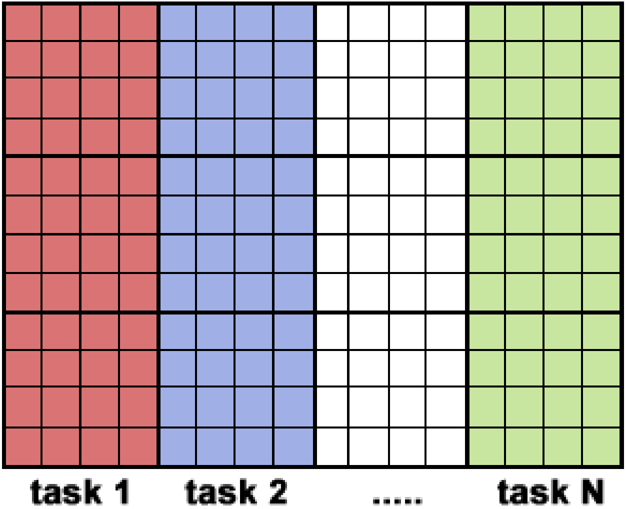
(or: how to use C211 in an interview)

Given a function that is both commutative and associative (e.g., $+$ or $*$)

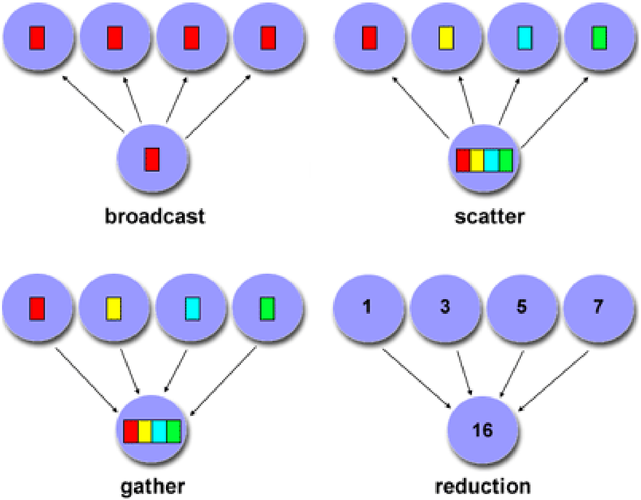
- **Commutative:** $x + y = y + x$
- **Associative:** $(x + y) + z = x + (y + z)$

Partition:	5 6 4 1	4 1 2 5	6 2 7 6	3 4 6 1
Map:	16	12	21	14
Reduce:	63			

Painfully Parallel Problems



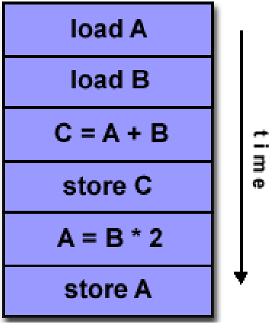
Painfully Parallel Problems



Architecture

Flynn's Taxonomy

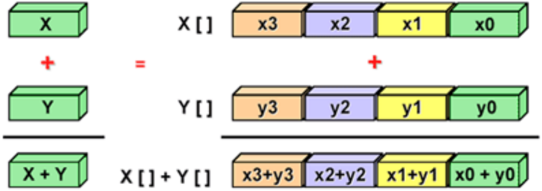
- SISD – Single Instruction, Single Data



Architecture

Flynn's Taxonomy

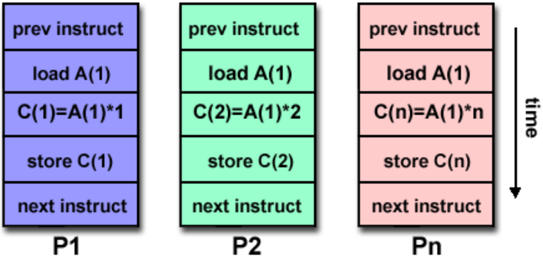
- SISD – Single Instruction, Single Data
- SIMD – Single Instruction, Multiple Data



Architecture

Flynn's Taxonomy

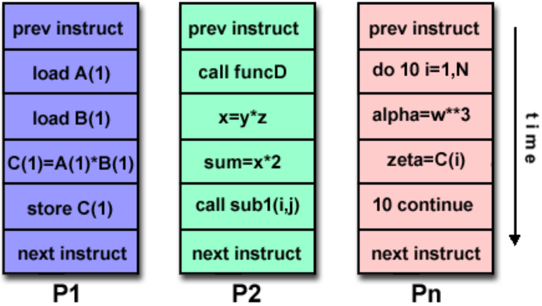
- SISD – Single Instruction, Single Data
- SIMD – Single Instruction, Multiple Data
- MISD – Multiple Instruction, Single Data



Architecture

Flynn's Taxonomy

- SISD – Single Instruction, Single Data
- SIMD – Single Instruction, Multiple Data
- MISD – Multiple Instruction, Single Data
- MIMD – Multiple Instruction, Multiple Data



Synchronization

Process 1	**Process 2**
$a = read(A)$	$a = read(A)$
$a = a + 1$	$b = read(B)$
$A = write(a)$	$A = write(a + b)$

Synchronization

Process 1	**Process 2**
$a = read(A)$	$a = read(A)$
$a = a + 1$	$b = read(B)$
$A = write(a)$	$A = write(a + b)$

Solutions:

- Barriers
- Locking
- Semaphores

Locks

C#

```
class Account {     // this is a monitor of an account
  long val = 0;

  public void Deposit(const long x) {
    lock (this) {   // only 1 thread at a time may execute this statement
      val += x;
    }
  }

  public void Withdraw(const long x) {
    lock (this) {
      val -= x;
    }
  }
}
```

Locks

Java

```java
class Account {      // this is a monitor of an account
  int val = 0;

  public synchronized void Deposit(int x) {
    val += x;
  }

  public synchronized void Withdraw(int x) {
    val -= x;
  }
}
```

Locks

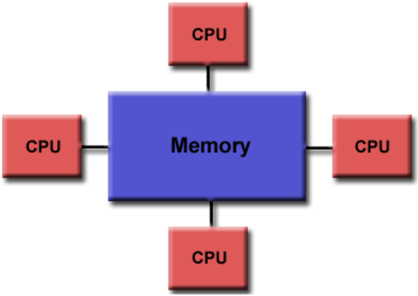
Python

```
from threading import Lock
class Account:
    def __init__(self, value):
        self.value = value
        self.lock = Lock()

    def deposit(x):
        with self.lock:
            self.value += x

    def withdraw(x):
        with self.lock:
            self.value -= x
```

Memory Models

Shared Memory

- Same address space
- Multithreading



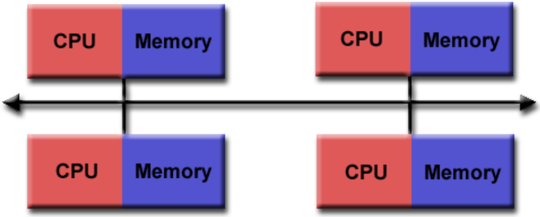
Memory Models

Shared Memory

- Same address space
- Multithreading

Distributed Memory

- Different address space
- Cluster computing/HPC



Memory Models

Shared Memory

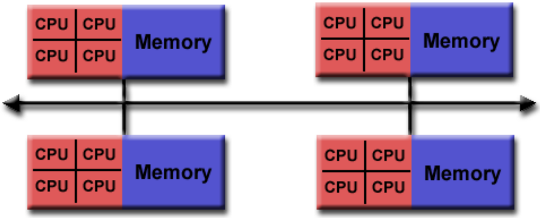
- Same address space
- Multithreading

Distributed Memory

- Different address space
- Cluster computing/HPC

Hybrid Shared-Distributed Memory

- Multiprocessing
- GPU Programming



Threads vs. Processes

Threads

- Shared memory
- Single process (core), multiple execution paths
- Subroutines, GUIs
- Low OS overhead
- Requires synchronization
- POSIX threads, Python `threading`, Java `java.lang.Thread`

Process

- Distributed memory
- Message passing
- High OS overhead
- Only way to utilize multicore or clusters

Pipes and Queues

Queue

- Bianry communication of messages
- I/O across processes

Pipes and Queues

Queue

- Bianry communication of messages
- I/O across processes

Pipe

- One-way communication of messages.
- Useful for handling I/O to multiple processes

MapReduce

2004 Google framework

`http://labs.google.com/papers/mapreduce.html`

APIs: C++, C#, Erlang, Java, OCaml, Perl, Python, PHP, Ruby, F#, R

MapReduce

2004 Google framework
`http://labs.google.com/papers/mapreduce.html`
APIs: C++, C#, Erlang, Java, OCaml, Perl, Python, PHP, Ruby, F#, R

Six-stage pipeline:

- input reader
- map function
- partition function
- comparison function
- reduce function
- output writer

MapReduce

```
void map(String name, String document):
  // name: document name
  // document: document contents
  for word in document:
    EmitIntermediate(word, "1");

void reduce(String word, Iterator partialCounts):
  // word: a word
  // partialCounts: a list of aggregated partial counts
  int sum = 0;
  for pc in partialCounts:
    sum += ParseInt(pc);
  Emit(word, AsString(sum));
```

Hadoop

The Big Concepts

- **JVM** – Implemented in Java, commonly used with Clojure and Scala

Hadoop

The Big Concepts

- **JVM** – Implemented in Java, commonly used with Clojure and Scala
- **HDFS** – distributed file system, operates in 64MB chunks.

Hadoop

The Big Concepts

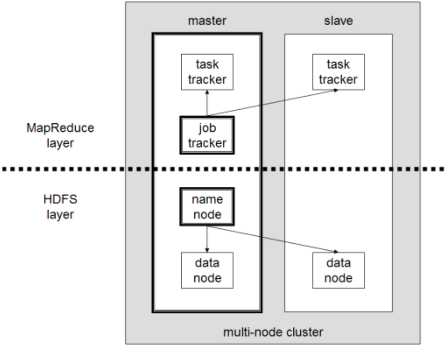
- **JVM** – Implemented in Java, commonly used with Clojure and Scala
- **HDFS** – distributed file system, operates in 64MB chunks.
- **Sharding** – Partitioning. Bring the application to the data to reduce bandwidth

Hadoop

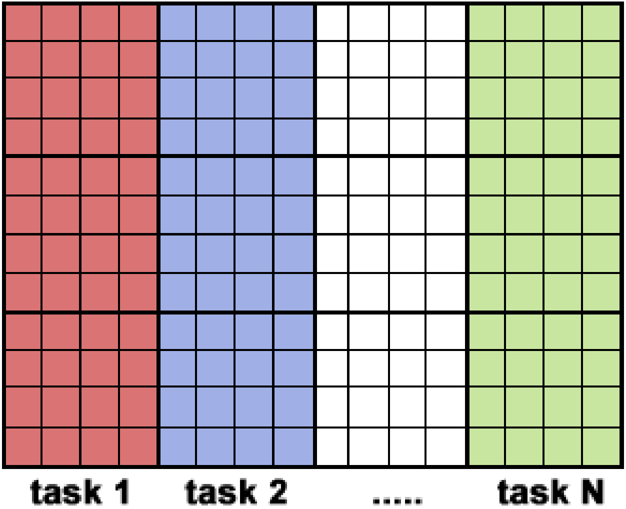
The Big Concepts

- **JVM** – Implemented in Java, commonly used with Clojure and Scala
- **HDFS** – distributed file system, operates in 64MB chunks.
- **Sharding** – Partitioning. Bring the application to the data to reduce bandwidth
- **JobTracker** server takes MapReduce job requests to available **TaskTracker** nodes. (FIFO process pool!)

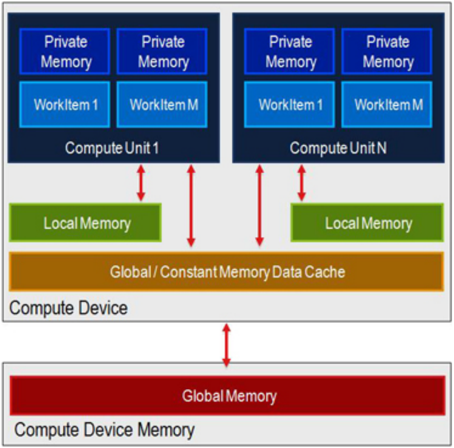
Hadoop


master
slave
task tracker
task tracker
MapReduce layer
job tracker
HDFS layer
name node
data node
data node
multi-node cluster

GPU Programming



GPU Memory Model



OpenMP: Upper Triangular Matrix

```
// Calculating Distances
float d; float* D;
for (i = 0; i < POPSIZE; i++) {
    D = dists[i];

    #pragma omp parallel for shared(D, i, dists) private(d, j)
    for (j = i+1; j < POPSIZE; j++) {
        d = distance(i, j);
        D[j] = d;
    }

    D[i] = 0.0;

    #pragma omp parallel for shared(D, i, dists) private(d, j)
    for (j = 0; j < i; j++) {
        d = dists[j][i];
        D[j] = d;
    }
}
```

Python multiprocessing

```
from multiprocessing import Pool
p = Pool()                    # initialize process pool
results = p.map(f, args)      # spawn processes
p.close()                     # close process pool
```

Resources

General

- https://computing.llnl.gov/tutorials/parallel_comp/

Python

- http://docs.python.org/library/threading.html
- http://docs.python.org/library/multiprocessing.html Python 2.6
- http://docs.python.org/dev/library/concurrent.futures.html Python 3.2

Java

- http://download.oracle.com/javase/7/docs/api/java/lang/Thread.html
- http://download.oracle.com/javase/tutorial/essential/concurrency/

Resources

MapReduce

- http://labs.google.com/papers/mapreduce.html
- http://hadoop.apache.org/

GPU Programming

- http://www.khronos.org/opencl/
- http://developer.amd.com/zones/openclzone/
- http://developer.amd.com/sdks/AMDAPPSDK/samples/
- http://developer.nvidia.com/opencl
- http://developer.nvidia.com/category/zone/cuda-zone

Practical Parallelism

Jaimie Murdock

IU Cognitive Science Program
810 Eigenmann Hall
jammurdo@indiana.edu

November 17, 2011