Adding Parallelism to AP CS Principles

CS4EDU Workshop

David Bunde

Knox College

Rationale for parallelism

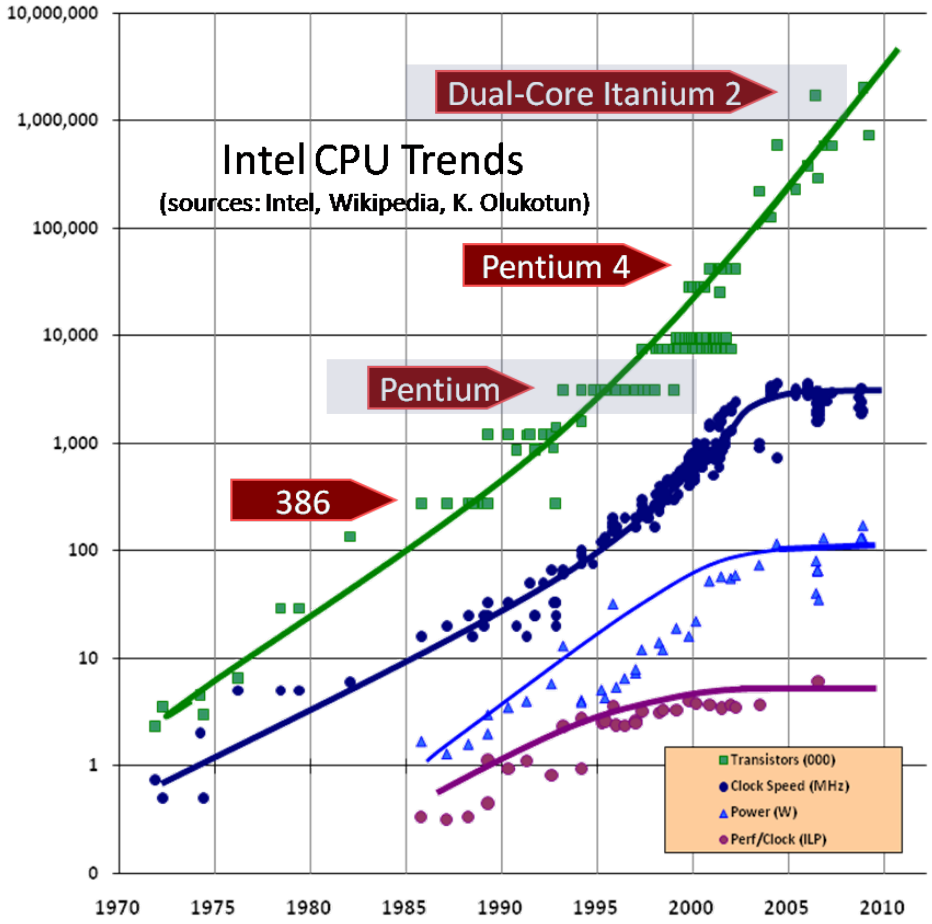

Figure: Herb Sutter "The free lunch is over: A fundamental turn toward concurrency in software" Dr. Dobb's Journal, 30(3), March 2005. http://www.gotw.ca/publications/concurrency-ddj.htm

An important distinction

- Parallelism
  - Solving problem faster with multiple cores or processors
- Concurrency
  - Coordinating access to shared resources

Puzzle demo

H. Neeman, L. Lee, J. Mullen, and G. Newman.
"Analogies for teaching parallel computing to inexperienced programmers",
inroads --- The SIGCSE Bulletin, 38(4), 2006.

Module-based approach

- Short (≤3 days) self-contained units
- Include supporting readings, labs, and/or assignments
- Ideally, suitable for variety of courses and language independent

My first module

- Counting prime numbers using Java/C threads
- Lab asking students to fix simple program followed by discussion and homework

Thread “workload”

```
class PrimeFinder implements Runnable {
   ...
   public void run() {
      for(int i=from; i<to; i+=2)
      if(isPrime(i))
        pCount++;
   }
}
```

Creating and starting the threads

```
pCount = 1;   //(already know 2 is prime)

PrimeFinder p1 = new PrimeFinder(3, 1000000);
Thread t1 = new Thread(p1);
PrimeFinder p2 = new PrimeFinder(1000001, 2000000);
Thread t2 = new Thread(p2);

t1.start();
t2.start();
```

Use of the module

- Fix critical bugs in given code
  - No join after threads are started
  - Race condition on shared count of primes
- Privatizing shared variable and load balancing to improve performance
- Homework: Parallelize version that computes small primes serially and then uses them to test larger values

Higher-level languages

Java threads implementation

```
class PrimeFinder implements Runnable {
  ...
  public void run() {
    int localCount = 0;
    for(int i=from; i<to; i+=2)
      if(isPrime(i))
        localCount++;
    synchronized(lock) { pCount += localCount; }
  }
}
```

```
  ...
  pCount = 1;   //(already know 2 is prime)

  PrimeFinder p1 = new PrimeFinder(3,1000000);
  Thread t1 = new Thread(p1);
  PrimeFinder p2 = new PrimeFinder(1000001, 2000000);
  Thread t2 = new Thread(p2);

  t1.start();
  t2.start();
  t1.join();
  t2.join();
  ...
```

Chapel implementation

```
const num = + reduce [i in [3..2000000] by 2]
    (if(isPrime(i)) then 1 else 0);
```

OpenMP implementation

```
#pragma omp parallel for reduction(+: num)
 for(i=3; i<=2000000; i+=2)
   if(isPrime(i))
     num += 1;
```

A different high-level language

Java version

```
PrimeFinder p1 =
      new PrimeFinder(3, 1000000);
Thread t1 = new Thread(p1);
PrimeFinder p2 =
      new PrimeFinder(1000001,
                      2000000);
Thread t2 = new Thread(p2);

t1.start();
t2.start();

t1.join();
t2.join();
```

Habanero Java (HJ) version

```
finish {
   async countPrimes(3,
                     1000000);
   async countPrimes(1000001,
                     2000000);
}
```

MapReduce / Hadoop

- Parallel data processing as map and reduce stages:

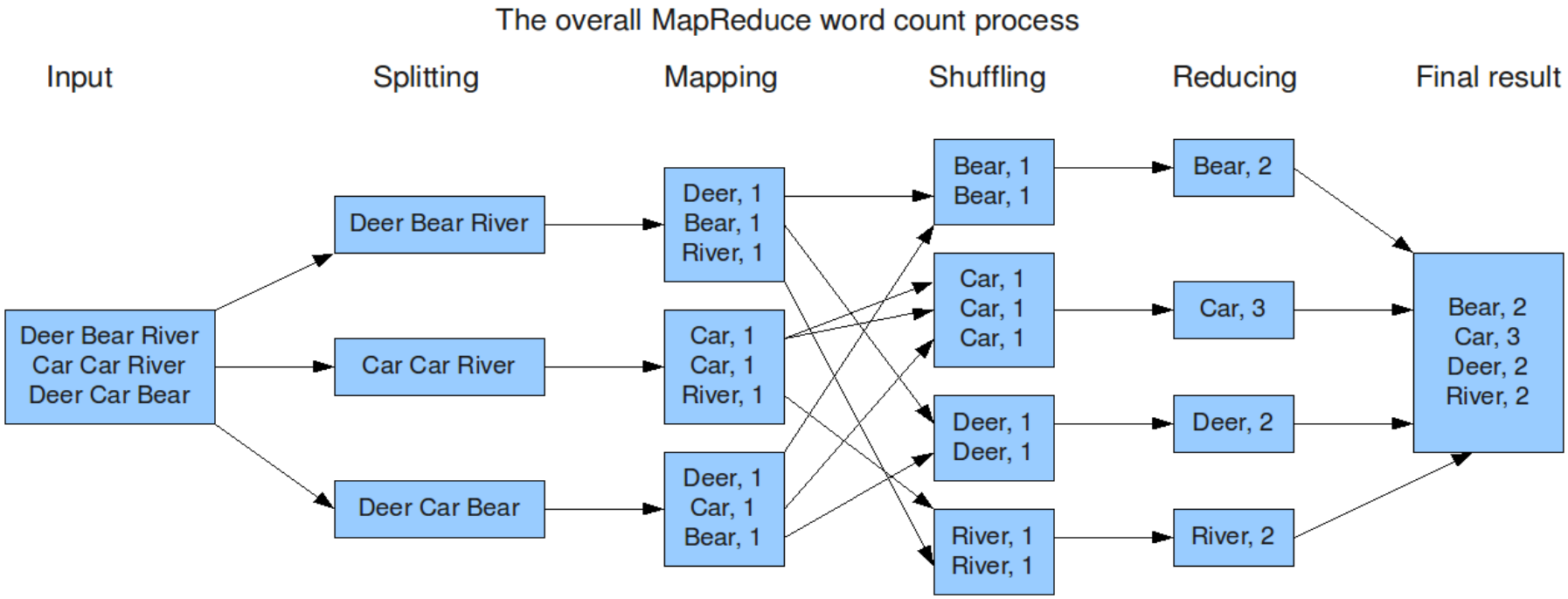

Image: http://blog.jteam.nl/2009/08/04/introduction-to-hadoop/

“Simple” Hadoop program

(Source: http://hadoop.apache.org/common/docs/r1.0.3/mapred_tutorial.html)

```
public static class Map extends MapReduceBase
      implements Mapper<LongWritable, Text, Text, IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(LongWritable key, Text value,
                  OutputCollector<Text, IntWritable> output, Reporter reporter)
                  throws IOException {
    String line = value.toString();
    StringTokenizer tokenizer = new StringTokenizer(line);
    while (tokenizer.hasMoreTokens()) {
      word.set(tokenizer.nextToken());
      output.collect(word, one);
    }
  }
}
```

```
public static class Reduce extends MapReduceBase
       implements Reducer<Text, IntWritable, Text, IntWritable> {
  public void reduce(Text key, Iterator<IntWritable> values
                     OutputCollector<Text, IntWritable> output,
                     Reporter reporter) throws IOException {
    int sum = 0;
    while (values.hasNext()) {
      sum += values.next().get();
    }
    output.collect(key, new IntWritable(sum));
  }
}
```

```
public static void main(String[] args) throws Exception {
   JobConf conf = new JobConf(WordCount.class);
   conf.setJobName("wordcount");

   conf.setOutputKeyClass(Text.class);
   conf.setOutputValueClass(IntWritable.class);

   conf.setMapperClass(Map.class);
   conf.setReducerClass(Reduce.class);

   conf.setInputFormat(TextInputFormat.class);
   conf.setOutputFormat(TextOutputFormat.class);

   FileInputFormat.setInputPaths(conf, new Path(args[0]));
   FileOutputFormat.setOutputPath(conf, new Path(args[1]));

   JobClient.runJob(conf);
 }
```

WebMapReduce (WMR)

[P. Garrity, T. Yates, R. Brown and E. Shoop. SIGCSE `11]

- Simplified web interface for Hadoop cluster

```
def mapper(key, val):
  words = key.split()
  for word in words:
    Wmr.emit(word, '1')

def reducer(word, counts):
  total = 0
  for count in counts:
    total += int(count)
  Wmr.emit(word, count)
```

Overall recommendations

- Definitely teach:
  - Idea of multicore and rationale for parallelism
  - Speedup (w/ load balance, overhead, and impact of serial sections)
- Maybe teach:
  - Parallelism vs. concurrency
  - Concurrency topics (race conditions, deadlock)
  - Actual coding of this...

Useful resources

- Thread lab:

  http://faculty.knox.edu/dbunde/pubs/threadIntro.html

- High-level languages:

  http://faculty.knox.edu/dbunde/parallel.html

- Variety of modules: csinparallel.org
- WMR (including cluster access):

  Libby Shoop (shoop@macalester.edu)

Thanks

- Contact me: dbunde@knox.edu

- Work partially supported by NSF DUE-1044299