Concurrency & Parallelism
Test 1 - 2015/16

José Duarte

June 20, 2020

1 Question 1

1.1 a)

Loop iterations should not have dependencies with other loop iterations.

1.2 b)

Loop unrolling is an optimization technique in which a `for` loop is written as if there was no loop. For example, the following loop:

```
for (int i = 0; i < 4; i++) {
    print("Hello, World!");
}
```

Becomes:

```
print("Hello, World!");
print("Hello, World!");
print("Hello, World!");
print("Hello, World!");
```

1.3 c)

```
for (int i = 0; i <n; i+=4) {
    sum += data[i];
    sum += data[i+1];
    sum += data[i+2];
    sum += data[i+3];
}
```

Or:

```
for (int i = 0; i <n; i+=4) {
    out[0] += data[i];
    out[1] += data[i+1];
    out[2] += data[i+2];
    out[3] += data[i+3];
}
sum += out[0] + out[1] +
    out[2] + out[3];
```

1.4 d)

No because the overhead of instantiating a task for one simple sum would not be worth the performance. If using the first approach from subsection 1.3 the variable would also be shared which would not work as well.

2 Question 2

Since each GPU has double the performance of a CPU we assume that instead of 4 GPUs, we have 8 CPUs. We also have a fixed input size so we can use Amdahl's Law.

$$S_p \leq \frac{1}{f + \frac{1-f}{p}}$$
$$S_p \leq \frac{1}{0.25 + \frac{0.75}{8}}$$
$$S_p \leq 2.91$$

3 Question 3

3.1 a)

When multiple processing units are required to execute a task and only one processing unit can execute it at a time.

3.2 b)

Cooperation occurs when processing units are required to wait for others to progress.

3.3 c)

Control point in which every processing unit is required to wait for the others.

3.4 d)

The relative size of a task/approach.

3.5 e)

Measurement of speedup for a fixed problem size, scaling with processing units.

3.6 f)

Identical operations applied to concurrently to different elements.

3.7 g)

Multiple tasks running in a concurrent fashion over multiple data.

3.8 h)

Relationship between two items in which one is required to execute before the other.

4 Question 4

4.1 a)

Read-After-Write since S_2 reads `x` after S_1 writes to it.

4.2 b)

Write-After-Read since S_2 writes to `y` after S_1 reads from it.

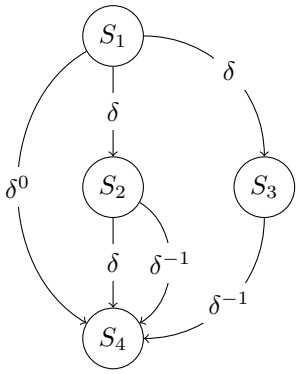
4.3 c)

N/A since it both writes are in different statements and the values are not read.

4.4 d)

Write-After-Write since S_2 writes to `x` after S_1 writes to it.

5 Question 5



6 Question 6

7 Question 7

Cilk+ is easier to program and requires less effort from the programmer. Pthreads give fine-grain control.

8 Question 8

- SISD - Serial Processors
- SIMD - Modern CPU's
- MISD - None
- MIMD - Distributed Cluster

9 Question 9

Both can lead to data consistency problems, crossbar switching has an high incremental cost and accesses to memory are not equidistant, the lack of a shared link improves bandwidth (comparing to bus based which shares the links).

10 Question 10

The map function applies one function over a collection of elements, returning the resulting collection. The reduce function uses one function to reduce a collection of elements to a single element.

11 Question 11

Consider document d as a set of words $w_1, w_2, \ldots, w_m$ of size m, D is then the set of documents $d_1, d_2, \ldots, d_n$. Each document d has an associated unique ID, for simplicity we consider the ID to be the number of the document $1, 2, \ldots, n$. At the document level we map d to tuples of $(id, w_m, 1)$ and then reduce to a collection of unique tuples (in the sense that no two tuples from the same document share the same keyword), this allows for relevance ranking. At a global level we reduce all resulting collections D' to a single collection containing tuples of $(w, d_1, d_2, \ldots, d_n)$ (that is, the word and the documents that contain it).