CS61C Summer 2014 Discussion 9 – Parallelism (Solutions)

Exercise 1 (Power): We're going to roll out a new processor. Compared to the previous version, we've reduced our capacitive load by 25% and are running it on 2V instead of 3V. How much faster can we clock the processor without exceeding the power consumption of the previous version?

$$C_1V_1^2f_1 \geq C_2V_2^2f_2 \Rightarrow \frac{f_2}{f_1} \leq \left(\frac{C_1}{C_2}\right)\left(\frac{V_1}{V_2}\right)^2 = \frac{4}{3} * \frac{9}{4} = 3$$

We can make the clock 3 times as fast (200% increase).

Exercise 2 (Amdahl's Law): In a given program, 95% of the execution time can be parallelized. How many processors are needed to achieve a speed up at least 10 times?

Check: $\frac{1}{1-F} = \frac{1}{0.05} = 20 \Rightarrow 20 > 10$ so a speedup of 10 times is possible.

$$10(1-0.95) + 10\left(\frac{0.95}{N}\right) < 1 \Rightarrow 0.5 + \frac{9.5}{N} < 1 \Rightarrow N > 19$$

Exercise 3 (Amdahl's Law): A given program runs 55 floating point operations, 5 of which can't be parallelized. What is the speed up for 5 processors? Now if that same program runs 105 floating point operations with the same 5 that can't be parallelized, what is the speed up for 5 processors? Assume a floating point operation takes t_{fp} time.

$$\frac{50t_{fp}}{5} + 5t_{fp} = 15t_{fp} \Rightarrow \frac{55}{15} = \frac{11}{3} \text{ speedup (Or } \frac{1}{\left(1-\frac{10}{11}\right)+\frac{\left(\frac{10}{11}\right)}{5}} = \frac{11}{3}\text{)}$$

$$\frac{100t_{fp}}{5} + 5t_{fp} = 25t_{fp} \Rightarrow \frac{105}{25} \Rightarrow \frac{21}{5} \text{ speedup (Or } \frac{1}{\left(1-\frac{20}{21}\right)+\frac{\left(\frac{20}{21}\right)}{5}} = \frac{21}{5}\text{)}$$

Exercise 4 (Amdahl's Law): The limits of Amdahl's law are shown below. Under what conditions/situations can we achieve the following limits?

a. $\lim_{F \to 1} S(F, N) = N$

A perfectly parallelizable task (where 100% of the program can be split up into equal and identical tasks)

b. $\lim_{F \to \infty} S(F, N) = \frac{1}{1-F}$

A machine with ∞ processors.

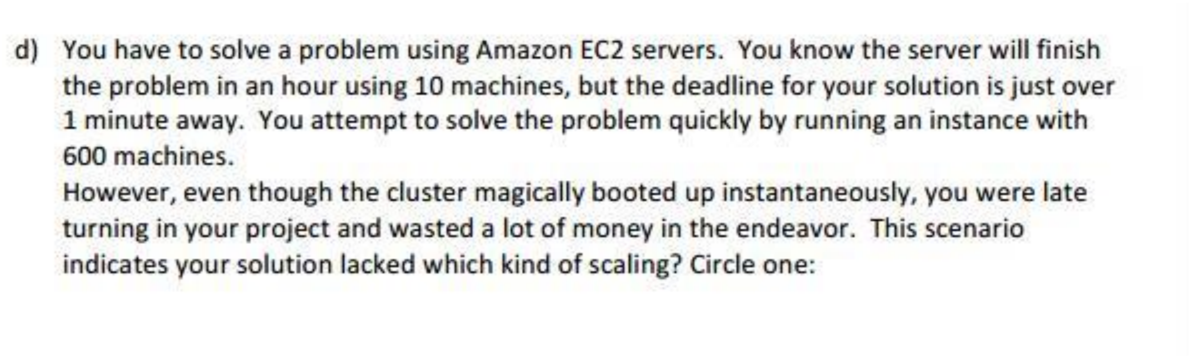
Exercise 5 (OpenMP): Use OpenMP to speed up the following code:

#pragma openmp parallel for shared(…) may be helpful

```
//Adds 2 n x n matrices
float* addMatrix (float* a, float* b, size_t n)
{
    float* result = malloc(sizeof(float) * n * n);
    for (int i = 0; i < n * n; i++)
    {
        result[i] = a[i] + b[i];
    }
    return result;
}
```

```
float* addMatrix (float* a, float* b, size_t n)
{
    float* result = malloc(sizeof(float) * n * n);
    #pragma openmp parallel for shared(a, b, n, result)
    for (int i = 0; i < n * n; i++)
    {
        result[i] = a[i] + b[i];
    }
    return result;
}
```

Fall 12 Midterm Q2



d) You have to solve a problem using Amazon EC2 servers. You know the server will finish the problem in an hour using 10 machines, but the deadline for your solution is just over 1 minute away. You attempt to solve the problem quickly by running an instance with 600 machines.
However, even though the cluster magically booted up instantaneously, you were late turning in your project and wasted a lot of money in the endeavor. This scenario indicates your solution lacked which kind of scaling? Circle one:

Weak Scaling **Strong Scaling**