Jack Ziesing
HW 1
951176818

Parallel Computing

1. Parallel computing is the process of using multiple processing resources at the same time to work together to solve a problem. A computer with one core can actually practice parallel computing due to the idea of multi-threading and achieving parallelism at the instruction level.

2. Three components of parallelism = concurrency, parallel hardware, and performance.

3. Rmax deals with the maximum performance achieved, or achievable, while Nmax deals with the size of the problem that goes with each rmax maximum performance. To answer this question parallel computer A must reach its rmax before computer B. Looking at existing machines we can see that the answer is yes, and that there have been machines that prove this already. But looking deeper you can see that up to a certain point rmax improves as problem size increases, but at some point it tops off, and the rmax doesn't grow much more.

4. Parallel computing has many reasons: larger problems, better performance, etc, but some may be still wondering, specifically what? Large weather simulators and monitoring systems are some of the most common programs to live on a large parallel computer but there are many other uses as well. Expanding our thought we can view parallel computers in smaller scales like our phones as examples of how parallel computing helps dailey activity. But a common theme is that parallel computing improves performance, which becomes a problem with lots of data, or hardware requirements. It is important to study because right now the industry is changing a lot and it will continue to change because for some reason no one is ever satisfied with performance, faster is always better.

Parallel Architectures

1. Shared memory parallel computing systems share memory, which enables any processor to reference any memory location in the system. All of the processors must be connected to the shared memory through some kind of network or bus that helps with concurrency. The advantages of this system is that processors can share data/memory efficiently and that the shared memory reduces average latency. Distributed memory parallel computing systems have memory for each of the processors in the system, so there is a bunch of memory that is grouped/divided/distributed up for each processor to have its own piece of memory. In distributed memory systems, some sort of network is needed to communicate data between each processor. Some advantages to distributed memory systems are scalable architecture (keeps costs down) and less concurrency issues.

2. NUMA (non-uniform memory access) parallel architecture is a type of system that contains both local and remote memory. It is based on a cross between shared memory and distributed systems. NUMA tries to tackle the issue shared memory systems have with scaling while also allowing fast retrieval of shared system by combining the two.

3. Multicore processors exist to give a processor more computing power and performance. They are needed to process larger programs, better graphics, and overall better performance. They are disruptive due to the fact that serial programming, which at first seemed really great, is now not good enough. In order to be effective the computation style must be different, that is, we must now use parallel patterns and algorithms to be affective.

4. In large parallel systems the interconnection network is key because it can easily improve or hurt the performance of the system. Some situations might occur where your parallel algorithm might lose efficiency with a particular network. There also is the issue of networks getting too large that the parallel computation costs less than data travel.

Parallel Performance Theory

1. The main differences between the two have to do with how speedup is defined. Gustafson-Baris law defines the speed up to include the number of processors which makes more processors increase or decrease the speed up based on problem size too. On the other hand Amdahl's law does not consider the number of processors in its speedup.

2. This is true in the long run because of a very large number of processors and communication. When you increase the number of processors for a while performance gets better for a specific problem, but at some point on every program there is a point where adding processors starts to hurt the performance. This is mainly because of the communication of data throughout the system. A system that large requires communication that creates more overhead than the actual computation. In greater detail you can think about a program that uses all the

3.

a. Not really. Saying that algorithm A has better strong scaling than B on a parallel machine just means that it is faster at computing a single use of the problem rather than weak scaling which concerns computing a larger scale of things. So just because algorithm A has better strong scaling means that algorithm B could most definitely have better weak scaling. An easy example is if the problem size that algorithm A has better strong scaling on grows to a really large size, algorithm B (if wrote in parallel correctly) could do better as this problem size gets bigger.

b. It might be possible. I don't know much about algorithm A or B, the question is vague, but I suppose that it might be possible to achieve better strong scaling for B if the machine was tailored in the right way. For example if the first machine maybe had more processors than the second machine that is used, algorithm B will then have a smaller limit as to size of program to run, which may entail algorithm B to have better strong scaling on that different machine.

4. This question is really rather vague. The question doesn't specify if each algorithm is solving the same problem. Speedup tells how much a system get better when its fastest sequential program is ran in parallel. To compare Speedups of any two systems is hard because they are two different systems. Comparing different speedups on different algorithms on the same machine might be more realistic. So yes I believe a problem is caused because there really isn't much to learn from two different speedups of different algorithms on different machines.

5.

a. To parallelize this problem you can you use a map pattern. The map pattern can be used for the multiplication part with Ai *Bi and then a sum reduction can be used to add everything up

b. T1 = 4n + 2n = 6n
Tp = log(n/p) * (4 + 52 + log p)
Sp = 6n/(log(n/p) * (4 + 52 + log p))
Efficiency = (6n/(log(n/p) * (4 + 52 + log p)))/p

c)

p	speedup	efficiency
1	.45	.45
4	.81	.2
16	1.81	.11
64	7.11	.11
256	640	250

d) through trial and error n=132 with the parallel runtime I provided.