Homework 1 - Parallel Computing, Architectures, and Performance Theory

Parallel Computer

1. A system that uses parallel processing features simultaneously to cooperatively solve a computational problem. A computer with a single processing core can still be considered a parallel computer if it uses parallel features like pipelining, vector processing, multithreading.
2. Concurrency, Parallel Hardware and Performance
3. Yes, since Rmax is measure as a rate per second in GFLOP/s computer A can still achieve a larger Rmax by computing its smaller problem size/Nmax at a faster rate than computer B computes its larger problem size.
4. Sustainable performance improvements. Always a need for computers to have higher performance but recently the power wall has ended the notion of simply increasing processor clock frequencies for better performance. To get around the power wall, modern architectures now include hardware to support many parallel features to increase performance. The issue now is moving away from serial programming and getting software to fully utilize parallel computing which is why it is important to study it today.

Parallel Architectures

1. Shared memory parallel systems utilize multiple processors with shared memory communicating via load/stores while a distributed memory parallel system feature processors with their own memory that are interconnected to communicate and cooperate with each other. Advantages for shared memory systems include ease of programming since there is a single view of data, compatibility with SMP hardware.

Advantages for distributed memory systems include simpler and more scalable hardware, explicit and simpler communication.

2. NUMA architectures still share memory but with processors having their own local memory while being able to access the memory local to other processors via network connection. NUMA attempts to solve issues with memory becoming a bottleneck as the number of processors increased in shared memory systems. By utilizing local memory NUMA reduces conflicts between processors over memory access.
3. Multi-core processors achieve higher performance at relatively lower frequencies increasing power efficiency, reducing heat production and in the end reducing fiscal costs. Improving single-core performance by increasing frequency speeds worked until the power wall was met and processor become too power hungry and approached temperatures too hot to manage economically. Multi-core processors solved these problems while increasing performance via parallelism. It is disruptive because moving from single-core to multi-core processors increases parallel hardware but requires changes in systems since each processor needs to utilize concurrency.
4. Yes, with large-scale parallel systems the goal is to maximize use of processing units. With systems boasting insane amounts of processors/cores the interconnection network is very important in being able to evenly distribute workload and provide communication without becoming a bottleneck from all of the latency and overhead.

Parallel Performance Theory

1. "Amdahl's Law" pertained to a fixed problem sized while "Gustafson-Basris' Law" accounted for increasing problem sizes such that the parallelizable fraction of problem would increase while the serial fraction would decrease thus improving speedup.

2. Efficiency goes down because the serial portion of the problem doesn't benefit from an increased number of processing elements, rather these elements would have to wait and idle during the serial portion thus reducing efficiency. Overhead in organizing an increased number of processing elements would also reduce efficiency. Furthermore, computational problems usually have a bound on parallelizable work; at some point dividing the workload further would incur too much overhead and reduced performance and past this point assigning more processing elements that wouldn't be utilized would clearly reduce efficiency. It's possible to get perfect efficiency (or close to it) with a problem set that has little to no serial portion and we limit the scope of increase in processing elements (i.e going from 2 to 4 workers, but eventually adding more workers will reduce efficiency) but perfect efficiency is very hard.
3. Algorithm A B
    a. No, Algorithm A may be good at dividing the problem set into smaller pieces which would make an increased number of workers beneficial at the same problem size but possibly canceled out with a proportional increased problem size. Algorithm B might be excellent at completing larger portions of the problem which would only be better for weak scaling(e.g. with a problem size of 100 algorithm B computes sets of 50 quickly in which adding a worker would do nothing but adding a worker and increasing the problem size by 50 would increase performance.
    b. Yes, possibly cache sizes or overall memory hierarchy for a machine are more optimal for Algorithm B
4. Yes, the comparison of speedups would favor the machine using its fastest sequential execution time. By using a slower sequential algorithm, it would increase 'f' decreasing

the overall speedup for that machine. Because Amdahl's law is so dependant on the fraction of sequential work, a fair comparison would use optimal algorithms for both machines.

5. A* B
    a. The multiplication of of Ai and Bi are independent, so that operation can be split amongst threads and done in any order, so probably a map function. Afterwards a reduce function could be used for the sum of all the products.
    b.
        i. Runtime: 4*N+2*log(n) computational 50*(N-1) communication
        ii. Speedup: Sp=1+(p-1)
        iii. Sp/P=