CS 406 Homework 1
Rebah Özkoç
29207

Q1.

If we look at the serial program the runtime is r + 1 – r = 1

We can use Amdahl's law here. It can be expressed like this:

$$Speedup = \frac{Old\ Execution\ Time}{New\ Execution\ Time}$$

$$= \frac{1}{[(1-\mathrm{r})+\left(\frac{r}{p}\right)]}$$

r is the fraction of the runtime of the program that can be perfectly parallelized.

1 - r is the fraction of the runtime that is not parallelized.

p is the number of processors.

We want to find the upper bound on the speedup, thus we need to increase the number of processors as much as possible.

If p → ∞, $\left(\frac{r}{p}\right)$ approaches to 0.

So, the upper bound of the speedup = $\frac{1}{(1-\mathrm{r})}$

Q2.

a) Latency is the time that passes from the memory request until the data arrives to the processor whereas bandwidth is the rate of data transfer from the memory to the processor. Latency has seconds and bandwidth has bytes/seconds units.
b) Spatial locality is the tendency of recent memory accesses will be to nearby locations on memory whereas temporal locality is the tendency of memory accesses to be reused within a short period of time.
c) In shared address space the multiple processor access to a common memory whereas in distributed address space each processor has its own private memory and processors communicate via message passing.

Q3.

a) The critical path of the graphs are the longest paths from the starting node to the ending node(s). In the first graph where $N = n^2$ the critical path length is $= 2 * n - 1$
In terms of N, it is $2\sqrt{N} - 2$

In the second graph the critical path is any of the paths from the starting node to the ending node. The height of the tree gives the result. The length of the critical path is:

$$= n = log_2(N + 1)$$

b) The maximum degree of the concurrency is equal to the largest number concurrent tasks at any point in the execution. In the first graph it is diagonal part.

$$= n = \sqrt{N}$$

In the second graph, it is the number of leaf nodes in the tree which is equal to:

$$= \frac{N+1}{2} = \frac{2^n}{2}$$

c) $Speedup = \frac{Old\ Execution\ Time}{New\ Execution\ Time} = \frac{T_1}{T_p}$

For the first figure:

The serial program runs one task per time. Thus:

$$T_1 = n^2$$

If we parallelize the tasks we need only 1 CPU time per each diagonal. Thus, we need to count the number of nodes that lie at the diagonal to calculate Tp:

$$T_p = \sqrt{(n*n)+(n*n)} = n*\sqrt{2}$$

$$Speedup = \frac{T_1}{T_p} = \frac{n^2}{n*\sqrt{2}} = \frac{n*\sqrt{2}}{2} = \frac{\sqrt{2*N}}{2}$$

Efficiency is calculated as $= \frac{T_1}{T_p * p}$

$$Efficiency = \frac{n^2}{n*\sqrt{2}*\ n*\sqrt{2}} = \frac{1}{2}$$

For the second figure:

The serial program runs one task per time. Thus:

$$T_1 = N = 2^n - 1$$

If the number of processor is equal to the maximum degree of concurrency at each step we can run all the task that lay at the same tree level. Thus, we need to count the number of nodes from height zero to height n for Tp, which is equal to the height of the tree.

$$T_p = n$$

$$Speedup = \frac{T_1}{T_p} = \frac{2^n - 1}{n} = \frac{N}{log_2(N+1)}$$

$$Efficiency = \frac{2^n - 1}{n * \frac{2^n}{2}} = \frac{N}{log_2(N+1) * \frac{N+1}{2}}$$

Q4.

The parallelized reversing function can be written like this.

```
#include <iostream>
#include <memory>
#include <omp.h>

using namespace std;

void reverse(int* arr, int length) {
    #pragma omp parallel for
    for (int i = 0; i < length / 2; ++i) {
        swap(arr[i], arr[length - i - 1]);
    }
}


int main(){
    int N = 1000000;
    int* arr = new int[N];
    for (int i = 0; i < N; i++) {
        arr[i] = rand();
    }

    reverse(arr, N);

}
```

Q5.

a) Given that the processor has two multiply-add units and each multiply-add operation involves two floating-point operations (one multiplication and one addition), it can execute up to 2 * 2 = 4 floating-point operations per cycle.

The processor operates at a frequency of 4 GHz, which means it can perform 4 billion cycles per second. Since we are looking for the peak performance, we don't need to consider latency of DRAM access, all the data can be at the registers. To find the peak performance in FLOPS, multiply the floating-point operations per cycle by the number of cycles per second:

Peak performance = Floating-point operations per cycle * Cycles per second

Peak performance = 4 * 4 * 10^9

Peak performance = 16 * 10^9 FLOPS (or 16 GFLOPS)

b) A dot-product computation performs one multiply-add (2 flops) on a single pair of vector elements, i.e., each floating-point operation requires one data fetch. It follows that the peak speed of this computation is limited to one floating point operation every 25 ns, or a speed of 2.5 MFLOPS.

c) If the cache is one word long it can store 32 bits and we can make a prefetch to increase the performance. And get the peak performance that we have at a.

d)

Q6.

a) If we don't put -fopenmp we won't have parallelism thus the code will run in a sequential manner. In this case, we will add 1 to a in each step and the function party will return only 5 every time.

b) If we add **-fopenmp** the program will run with parallel manner. We will have 3 threads. Since the addition is not wrapped with an "omp critical" or an "omp atomic" there will be race condition to access variable a. The maximum value of the function party will be 5 if every thread avoids writing while other threads are reading it or writing it. It can't be greater than 5 because the "pragma omp parallel for" manages the variable "i".
If a collision happens after first thread reads the value while a = 0 the other two threads can finish the loop by going 4 times total and at the end the first thread makes an increment and variable a becomes 1. Thus, maximum return value is 5 and minimum return value is 1.