Assignment 4 : Analytical Modeling of Parallel Programs
CSE560: GPU Programming

Anshuman Suri
2014021

1. All three parts have been calculated individually for the four task-dependency graphs:

    a. For a full binary tree of height n, the number of leaf nodes is 2n (which will determine the maximum degree of concurrency, as all those nodes can be run in parallel).
        i. The maximum degree of concurrency will be the leaf nodes, which is 2^{n-1}.
        ii. Maximum possible speedup for an infinite number of processing elements is (sum of nodes)/(critical path). The critical path will have length the same as the height of the tree. Thus, the maximum possible speedup will be $\frac{2^n-1}{n}$
        iii. (i) Since the number of processing elements is the same as the maximum degree of concurrency, Tp = critical path length.Thus:

        Speedup = $\frac{Ts}{Tp} = \frac{2^n-1}{n}$

        Efficiency = Speedup / p = $\frac{2^n-1}{n * 2^{n-1}} = \frac{2}{n}(1 - \frac{1}{2^n})$

        Overhead = pTp - Ts = $(n * 2^{n-1}) - (2^n - 1)$

        (ii) Since the number of processing elements is half the maximum degree of concurrency, the processes will be allocated to tasks on one-to-one till the second last level, after which they will be allocated such that every process gets 2 tasks. Thus, Tp = (n-1) + 2 (since till the second last level, we have critical path length, and in the last level sharing) = n + 1

        Speedup = $\frac{Ts}{Tp} = \frac{2^n-1}{n+1}$

        Efficiency = Speedup / p = $\frac{2^n-1}{(n+1) * 2^{n-1}} = \frac{2}{n+1}(1 - \frac{1}{2^n})$

        Overhead = pTp - Ts = $((n+1) * 2^{n-1}) - (2^n - 1)$

    b. The structure of this task dependency graph is the same as in the previous part, and will not affect any of the answers (as the maximum degree of concurrency, critical path length, all remain the same). Thus, the answers for this part will be

same as (a).

c. For a given square graph, elements along the diagonal can be run in parallel (which is the maximum); the square can be seen as two trees connected with their leaf nodes common.
   i. The maximum degree of concurrency will be the number of elements along the main diagonal, which is n.
   ii. The critical path length will be the longest path between two opposite ends of square of side n, which is 2n-1. Thus, the maximum possible speedup will $\frac{n^2}{2n-1}$
   iii. (i) Since the number of processing elements is the same as the maximum degree of concurrency, Tp = critical path length.Thus:

Speedup = $\frac{Ts}{Tp} = \frac{n^2}{2n-1}$

Efficiency = Speedup / p = $\frac{n^2}{n*(2n-1)} = \frac{n}{2n-1}$

Overhead = pTp - Ts = $n * (2n-1) - n^2 = n^2 - n$

(ii) Since the number of processing elements is half the maximum degree of concurrency, the processes will be allocated to tasks this way:

We keep allocating tasks to independent processors with levels along diagonals until the point where we can, which is, a square of side n/2. After that, we start from the bottom left (because of the structure of the task dependency graph) and allocate processors in sizes of n/2, wrapping around to the left diagonal when we have extra processors left after that diagonal. After that, we process the corner right square nodes normally. Thus, Tp will be:

$$ceil(n/2) + (n^2 - 2(* (ceil(n/2) * (ceil(n/2)+1)/2))/ceil(n/2) + ceil(n/2)$$
$$= 2 * ceil(n/2) + n^2/ceil(n/2) - ceil(n/2) - 1$$
$$= ceil(n/2) + n^2/ceil(n/2) - 1$$

(the first and last ceil(n/2) terms are for up to the diagonal parts which can be done with each task having an independent processor, and the term in between is because of the nodes in the area in between, being divided among processors with wrap around to the next diagonal.

Speedup = $\frac{Ts}{Tp} = \frac{n^2}{ceil(n/2) + n^2/ceil(n/2) - 1}$

Efficiency = Speedup / p = $\frac{n^2}{ceil(n/2)(ceil(n/2) + n^2/ceil(n/2) - 1)}$

Overhead = pTp - Ts = $ceil(n/2) * (ceil(n/2) + n^2/ceil(n/2) - 1) - n^2$

d. For the given graph, the number of nodes at every level is increasing as the level increases. Also, the number of levels (thug the height) of this graph is n.
   i. The maximum degree of concurrency will be the number of nodes in the last level, that is, n.
   ii. The critical path length of this graph is n, as that is the height of this graph. Thus, the maximum possible speedup will be $\frac{n*(n+1)}{n*2} = \frac{(n+1)}{2}$
   iii. (i) Since the number of processing elements is the same as the maximum degree of concurrency, Tp = critical path length.Thus:

Speedup = $\frac{n*(n+1)}{n*2} = \frac{(n+1)}{2}$

Efficiency = Speedup / p = $\frac{n+1}{n*2}$

Overhead = pTp - Ts = $(n^2) - (n(n+1))/2 = n(n-1)/2$

(ii) In this case, we will use the processors up to the part of the graph with ceil(n/2) depth. After that, we proceed in a zigzag pattern like we did in the previous part. The critical path length from the part with tasks mapped to independent will be n/2. For the remaining graph, we will keep wrapping around. Thus, Tp will be:

$$ceil(n/2) + ((n*(n+1)/2) - (ceil(n/2)*(ceil(n/2)+1)/2))/ceil(n/2)$$
$$= ceil(n/2) + (n*(n+1))/(2*ceil(n/2)) - (ceil(n/2)+1)/2$$

Speedup = $\frac{Ts}{Tp} = \frac{n}{ceil(n/2) + (n*(n+1))/(2*ceil(n/2)) - (ceil(n/2)+1)/2}$

Efficiency = Speedup / p =

$$\frac{n}{ceil(n/2)*(ceil(n/2) + (n*(n+1))/(2*ceil(n/2)) - (ceil(n/2)+1)/2)}$$

Overhead = pTp - Ts =

$$ceil(n/2)*(ceil(n/2) + (n*(n+1))/(2*ceil(n/2)) - (ceil(n/2)+1)/2) - (n(n+1)/2)$$

2.

a. Tp = (n/p -1) + 11log2(p)
   Total execution time <= 512
   i. p=1:
      (n-1) + 0 < =512
      n <= 513
      Thus, the largest problem size that can be solved is 513
   ii. p=4:
      (n/4-1) + 11*2 <= 512
      (n/4 - 1) + 22 <= 512
      n/4 - 1 <= 490
      n/4 <= 491

n <= 1964

Thus, the largest problem size that can be solved is 1964

iii. p=16:

(n/16-1) + 11*4 <= 512

n/16-1 + 44 <= 512

n/16 <= 469

n <= 7504

Thus, the largest problem size that can be solved is 7504

iv. p=64:

(n/64-1) + 11*6 <= 512

n/64-1 <= 446

n/64 <= 447

n <=28608

Thus, the largest problem size that can be solved is 28608

v. p=256:

(n/256-1) + 11*8 <= 512

n/256 -1 <= 424

n/256 <= 425

n <= 108800

Thus, the largest problem size that can be solved is 108800

vi. p=1024:

(n/1024-1) + 11*10 <= 512

(n/1024-1) <= 402

n/1024 <= 403

n <= 412672

Thus, the largest problem size that can be solved is 412672

vii. p=4096:

(n/4096-1) + 11*12 <= 512

n/4096 - 1 <= 380

n/4096 <= 382

n <=1560576

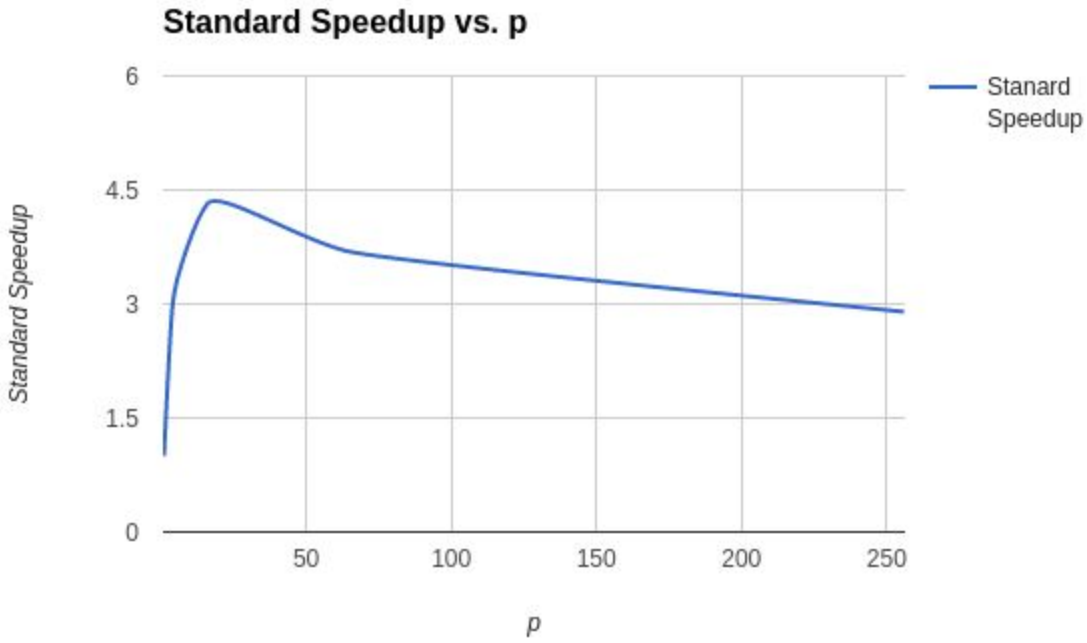
Thus, the largest problem size that can be solved is 1560576

b. No. In general, an arbitrarily large problem cannot be solved in constant time, no matter how many processing elements we have. Consider the problem of adding n numbers, where n is arbitrarily large. We have access to increasing P such that it is sufficient for N. However, even the most optimal algorithm involves communication between processes, writing to memory. Since this operation has a non-zero overhead, increasing N increases the sum of these overheads. Thus, the running time is no longer constant, as overheads add up as the problem size shoots up.

Since we have a counterexample, the statements cannot possibly hold in general.

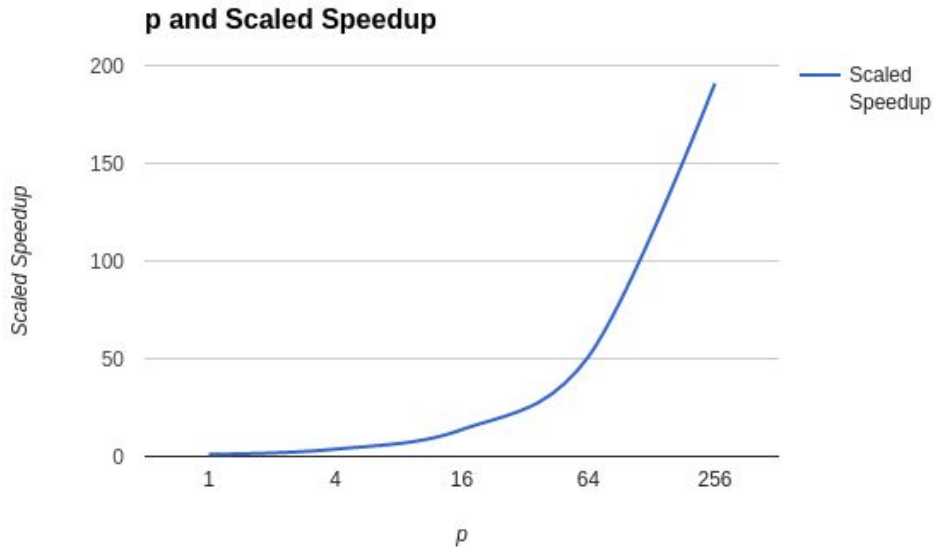
3. *Normal speedup:*

The normal speedup is defined as Ts/Tp = $\frac{n-1}{(n/p-1)+11logp}$. Plotting this for the value of n as 256, and different values of p, we get the following curve:



Scaled speedup:

The scaled speedup is defined as Ts/Tp = $\frac{p(n-1)}{(p*(n-1)/p-1)+11logp} = \frac{p(n-1)}{n-2+11logp}$. Plotting this for the value of n as 256, and different values of p, we get the following curve:



Looking at these graphs, we can conclude that the normal speed up seems to drop increasing as we increase, however the scaled speedup keeps increasing. The normal speedup starts decreasing after a while because of low efficiency. However, the low overhead is factored into the equation of scaled speedup, because of which its graph is unlike normal speedup and is increasing.