Comp 428
Maximilien Malderle
26562906

NR1:

c)

The tasks that are mapped to a process is the generation of random xy coordinates, the determination of whether the generated coordinate is inside/outside the circle and the appropriate increment of the counter variable.

The granularity of the task is determined by the number of processes over the amount of iterations of the point generating loop.

G = Loop Iterations / number of active processes

Data-Parallel:

In this case the problem is data parallel, because there are no data dependencies, each individual point creates a result. This result is summed and manipulated at the end of the program. Therefore there is no order in which the points are created and tested, meaning that they can all be done in parallel.

Hybrid:

This is not a hybrid program, it focuses primarily on data parallelization.

SPMD:

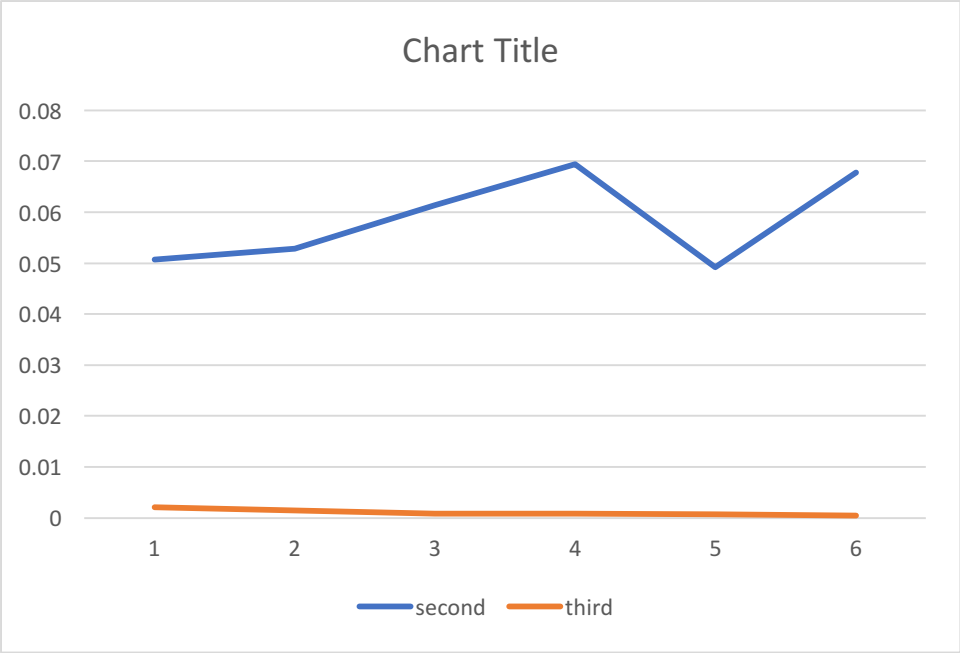
This is not the case the parallelizable portion of the program is localized at one portion (not independent points). Therefor there is no point to execute multiple parts of the program at the same time. No control unit is needed to execute different portions of the task.

MPMD:

This is not the case in this program. To be MPMD there needs to be at least two programs that are run, which is not the case here.

e)

		1	2	3
time	regular	0.008176	0.008176	0.008176
	second	0.161350012	0.077296019	0.04440784
	third	3.872212887	2.805860043	3.559684992
speedup	second	0.050672447	0.105775176	0.184111634
	third	0.002111454	0.002913902	0.002296832
		1	2	3
speddup/cpu	second	0.050672447	0.052887588	0.061370545
	third	0.002111454	0.001456951	0.000765611
		4	5	6
time	regular	0.008176	0.008176	0.008176
	second	0.02947402	0.033234119	0.020116091
	third	2.523200989	2.512112141	3.434199095
speedup	second	0.27739684	0.246012235	0.406440799
	third	0.003240328	0.003254632	0.002380759
		4	5	6
speddup/cpu	second	0.06934921	0.049202447	0.067740133
	third	0.000810082	0.000650926	0.000396793



The third answer has a very low speedup, at the beginning it is 470 times slower than the regular c program. The reason for this is the additional communication overhead. Each time the program runs the main process has to create sub processes, communicate the portion of the points to calculate and finally the sub process needs to communicate its results back to the master process. All this communication takes time and slows down performance. The speed up is also not linear, because with the more sub processes the higher the over head but also the faster the execution. As execution reaches 0 as processors grow, the gain in efficiency flattens out, but the overhead keeps on growing, resulting in overhead growing faster than processing efficency.

Nr 2:

Npoints = 10000

```
Circle count = 0

P = #tasks

If(MASTER)
        Do i=0,p
                Send(to: Process i what: npoint chunk)
                Npoints -chunk
End if

Do true
        If(MASTER)
                If(Npoints == 0)
                        False
                End if

                Receive(Count and process rank) //process is done and waiting
                Send(to: Process recieved what: npoint chunk)
                Npoints –chunk
        Else if(Slave)
                Int count = 0
                X = rand
                Y = rand
                If(inside(X,Y))
                        Count ++
                End if
                Send(Count/rank)
        End if
End do
```

Nr 3:

a)

The lowest amount of processors to achieve minimum time is 10;

Ts = 1 unit + 10*2 units + 5*5 units + 1unit = 47 units
Tp10 = 1 unit + 2 units + 5 units + 1 unit = 9 units

SpeedUp = 47/9 = 5.22
Efficency = 5.22/10 = 0.522

b)

i)

Increasing the amount of processors does not necessarily increase efficiency. As p > 10 the efficiency will fall. Do to the dependence of E on p (E = S/p) as p increases efficiency goes down. This mean that if the number of processors increases beyond ten, the speedup wont increase anymore, meaning that the efficiency will go down.

ii)

Decreasing the amount of processors does not increase efficiency, because it will most likely decrease the speed up. Speedup 10 processors > Speedup 5 processors etc. The lower amount of processors makes increases efficiency but a lower Speedup lowers it. If Speedup lowers efficiency more than the processors increase it, then overall efficiency will fall.

iii)

Having faster processors will always increase efficiency until the unit time cannot be reduced anymore. We don't change the number of processors, yet there is a speed up. Meaning that efficiency will always rise as long as a better processor actually decreases the processing time.

c)

The maximum Speedup is S = 47 units / 15 units = 3.1333333

Tp = 1unit + 2units (4 regular 2unit processes) + 5 units (2 5unit processes + 4 2 unit processes + 2 half of 2unit processes) + 6 units (2 half of 2 unit processes + 3 5unit processes (last process started 1 unit after the previous 5 unit process are done and must complete one more)) + 1 unit

d)

The maximum Speedup is S = 47 units / 11 units = 4.272727

Tp = 1 unit + 2 units (fast: 4 2unit, slow: 2 2unit) + 5 units (fast: 4 2 units (in 2units) 2 5 units (in 2.5 units) 1/5 5 units, slow: 2 5 units) + 2 units (fast: finishes 4/5 of last 5 unit) + 1 unit = 11 units

Nr4:

a)
Traversing the height of a tree requires O(log n) time. In a sequential environment we have to take the width into consideration as well, the width cannot exceed input size n. So sequential time for quicksort is O(n*log(n)). In a parallel environment with infinite processors the width processes can always be handled in parallel, meaning the time is O(log(n)).

Speedup = Ts/ Tp

Ts = O(n*log(n))
Tp = O(log(n))

Speedup = n

b)
Having N processors and N input size does not change the speed up from a. At any level there will never be more processes than processors. The bottom of the tree will have max N nodes which can be handled by the processor.

Speedup = n

Efficiency = S/p

N = n
N = p

Efficiency = N/N = 1

c)

This means that there is no real gain from a parallel implementation. The extra cost of the processors offsets the gained speed of processing.

Nr5:

Lets assume that the critical path length is equal to 1.

This means that the number of particles is equal to 1, because the critical path is the longest path in the task dependency graph, so if the longest path is 1, that means that there is only 1 task.

The maximum degree of concurrency in a program is equivalent to the critical path, it cannot be sub divided.

With these assumptions the formula holds

$t/l <= d <= t-l + 1$

$1/1 <= 1 <= 1-1+1$

$1 <= 1 <= 1$