Leon Wilkosz	193101

AMCS 312

Homework 1

Exercise 1

(a)

We compute the speedup $\frac{T_1}{T_p}$ for different combinations of non-parallelizable fractions of code and the number of processors in Table 1. The graph of the cross section along the middle row is found in Figure 1 and the graph of the cross section along the middle column is found in Figure 2. We choose to evaluate the Speedup in the number of processors in an exponential scale, since for a large number of processors, relatively small changes become less important. On the other hand, we choose equally spaced a, since the whole range seems interesting.

The results of this exercise give us a realistic expectation of what to expect from the speedups possible with parallelization. We see in Figure 1 that it highly depends on the fraction of parallelizable code and that already a little fraction of non-parallelizable code is enough to strongly dampen the speedup. Also, in Figure 2 we see that given a fixed fraction of non-parallelizable code, it does not make sense to increase the number of processors at some point as the speedup converges asyptotically to a maximum given through Amdahl's law. The overall conclusion is that it is crucial to develop algorithms with an as large as possible parallelizable fraction to see the benefits of parallel computing and that the latter is not a magic tool that makes every computation efficient.

p \ a	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1	1	1	1	1	1	1	1	1	1	1	1
2	2	1.818	1.666	1.538	1.428	1.333	1.25	1.176	1.111	1.052	1
4	4	3.076	2.5	2.105	1.818	1.6	1.428	1.29	1.176	1.081	1
8	8	4.705	3.333	2.58	2.105	1.777	1.538	1.355	1.212	1.095	1
16	16	6.4	4	2.909	2.285	1.882	1.6	1.391	1.23	1.103	1
32	32	7.804	4.444	3.106	2.388	1.939	1.632	1.409	1.24	1.107	1
64	64	8.767	4.705	3.216	2.442	1.969	1.649	1.419	1.245	1.109	1
128	128	9.343	4.848	3.273	2.471	1.984	1.658	1.423	1.247	1.11	1
256	256	9.66	4.923	3.303	2.485	1.992	1.662	1.426	1.248	1.110	1
512	512	9.827	4.961	3.318	2.492	1.996	1.664	1.427	1.249	1.11	1
1024	1024	9.912	4.98	3.325	2.496	1.998	1.665	1.427	1.249	1.11	1

Table 1 Speedups for different combinations of non-parallelizable fractions and processor numbers

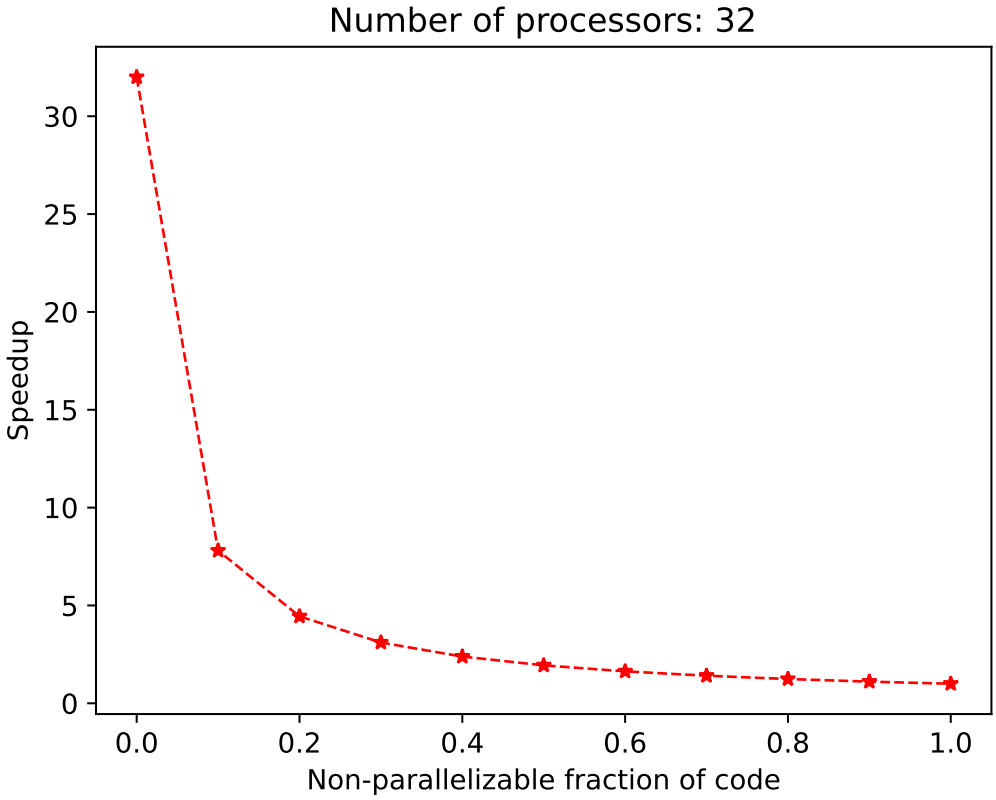

Figure 1 Speedup for a fixed number of processors over the fraction of non-parallelizable code

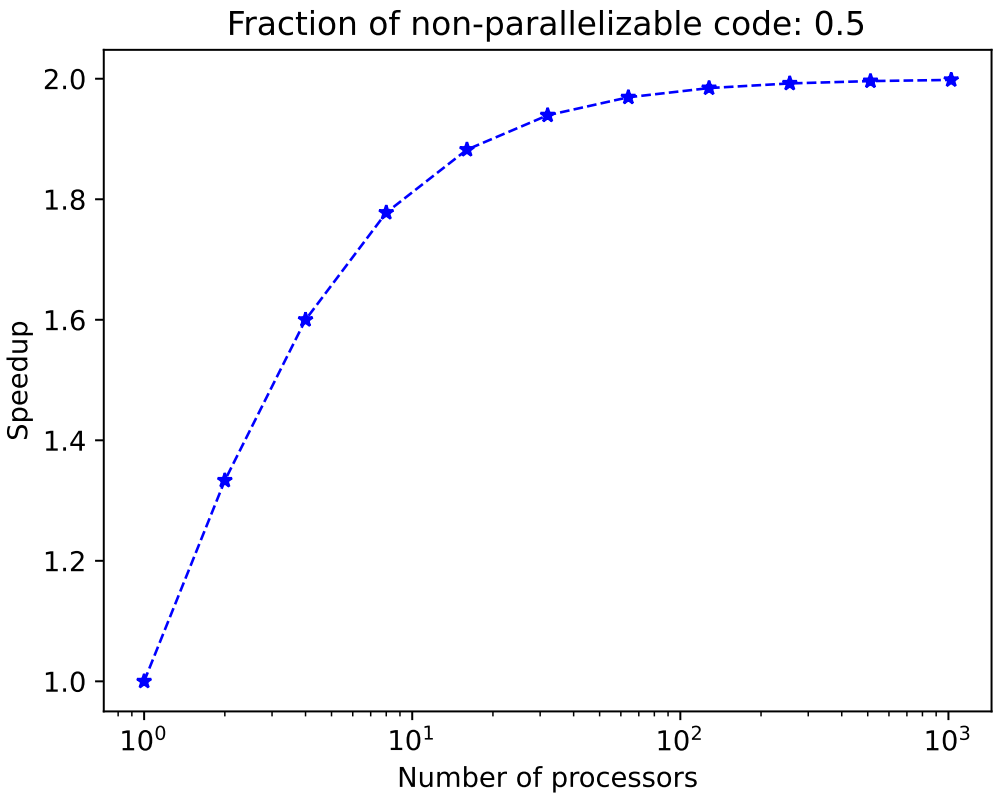

Figure 2 Speedup for a fixed fraction of non-parallelizable code over the number of processors

(b)

When a is the fraction of non-parallelizable code, Ahmdals law describes the time T_p to run on p processors as

$$T_P = (1-a)\frac{T_1}{p} + aT_1,$$

where T_1 is the time needed to solve the problem on 1 node and p is the number of nodes. A computation derives the parallel efficiency as

$$E = \frac{T_1}{pT_p} = \frac{T_1}{p(1-a)\frac{T_1}{p} + apT_1} = \frac{1}{1-a+ap}.$$

(c)

As an example we can consider the method of lines for the solution $u(x,t)$ of a scalar hyperbolic PDE with two dimensional space component $x \in \Omega \subset \mathbb{R}^2$ and one dimensional time component t. Due to finite speed of information propagation, coming from hyperbolicity, it is enough to use $\Delta t = \mathcal{O}(\Delta x)$ to ensure stability. We can and will therefore assume for now that $\Delta x = \mathcal{O}(\frac{1}{n})$ and $\Delta t = \mathcal{O}\left(\frac{1}{n}\right)$. Put simply, the method of lines discretizes space and computes the time update through independent ODEs. The cost for computing these updates with Euler's method is $\mathcal{O}\left(\frac{1}{\Delta t}\right) = \mathcal{O}(n)$. Further, this time update is a sequential operation that is not parallelizable. Since space is two dimensional here, this has to be done independently $\mathcal{O}\left(\frac{1}{(\Delta x)^2}\right) = \mathcal{O}(n^2)$ times, once at each grid point. This can be done completely in parallel. Concluding, the overall cost of the problem grows as

$$\mathcal{O}(n^2) \cdot \mathcal{O}(n) = \mathcal{O}(n^3),$$

whereas the non-parallelizable part of it grows as

$$\mathcal{O}(n).$$

Translating this into the language of the previous parts, the non-paralellizable fraction a evolves as

$$a = \mathcal{O}\left(\frac{n}{n^3}\right) = \mathcal{O}\left(\frac{1}{n^2}\right),$$

and the parallelizable fraction evolves as

$$1 - a = \mathcal{O}\left(1 - \frac{1}{n^2}\right).$$

Choosing an appropriate $p = \mathcal{O}(n^2)$ gives here constant parallel efficiency.

(d)

Let all the notions be given as in the exercise sheet. We assume that the time necessary to solve the problem without parallelization is T_1. The time T_P to run on the maximal number of nodes is then given by summing the parallelized runtimes for the different classes of work:

$$T_P = \sum_{p=1}^{P} a_p \frac{T_1}{p},$$

The respective speedup is given by

$$\frac{T_1}{T_P} = \frac{1}{\sum_{p=1}^{P} \frac{a_p}{p}}.$$

Exercise 2

(a)

We assume the setting to be as in the homework sheet and the latencies as in the slides. Let us assume that we desire to wash n loads of laundry. Without parallelization, the time to complete this task is given by

$$T_1 = n \cdot (30 + 40 + 20) \text{ minutes } = 90n \text{ minutes.}$$

The next student always starts drying when the previous student finished drying. Hence, under optimal execution, n students solve the task in

$$T_n = 30 + n \cdot 40 + 20 \text{ minutes.}$$

Thus, the speedup is given as follows and grows asymptotically:

$$\frac{T_1}{T_n} = \frac{9n}{3 + 4n + 2} \to \frac{9}{4} = 2.25 \text{ as } n \to \infty.$$

Hence, the parallel efficiency goes to zero,

$$E = \frac{T_1}{nT_n} = \frac{9}{3 + 4n + 2} \to 0 \text{ as } n \to \infty.$$

Since the parallel efficiency is not increasing but in fact decreasing and going to zero asymptotically, we can only conclude that the problem creator means speedup rather than parallel efficiency. We will continue with that assumption until the end of the exercise. The maximum speedup is (asymptotically) 2.25.

(b)

We obtain the following values for the speedup:

No. of students	1	2	3	4	5	6	7	8	9	10	11
Speedup	1	1.384	1.588	1.714	1.8	1.862	1.909	1.945	1.975	2	2.037

With eleven students we are in the 10% range, i.e. $\frac{2.25-2.037}{2.25} \leq 0.1$.

Exercise 3

(a)

With $c(p) = c$ we find

$$\partial_p T(p,n) = -\frac{a(n)}{p^2} \stackrel{!}{=} 0,$$

i.e. we should ideally use an infinite number of processors.

(b)

With $c(p) = \log(p)$ we find

$$\partial_p T(p,n) = -\frac{a(n)}{p^2} + \frac{1}{p} \stackrel{!}{=} 0,$$

i.e. $p = a(n)$.

(c)

With $c(p) = \sqrt{p}$ we find

$$\partial_p T(p,n) = -\frac{a(n)}{p^2} + \frac{1}{2\sqrt{p}} \stackrel{!}{=} 0,$$

i.e. $p = (2a(n))^{\frac{2}{3}}$.

(d)

With $c(p) = p$, we find

$$\partial_p T(p,n) = -\frac{a(n)}{p^2} + 1 \stackrel{!}{=} 0,$$

i.e. $p = \sqrt{a(n)}$.

Exercise 4

(a)

Following the lecture slides on the derivation we have the discrete mass conservation

$$\begin{aligned}\frac{\Delta x\Delta y\Delta z}{\Delta t}(\rho_{ijk}^{l+1}-\rho_{ijk}^{l}) = &-\Delta y\Delta z\left[(\rho u)|_{i+1/2,jk}^{l}-(\rho u)|_{i-1/2,jk}^{l}\right]\\ &-\Delta x\Delta z\left[(\rho v)|_{i,j+1/2,k}^{l}-(\rho v)|_{i,j-1/2,k}\right]\\ &-\Delta x\Delta y\left[(\rho w)|_{ij,k+1/2}^{l}-(\rho w)|_{ij,k-1/2}^{l}\right].\end{aligned}$$

To estimate the staggered quantities on the right hand side we average over the neighboring cells:

$$(\rho u)|_{i+1/2,jk}^{l}\approx\frac{1}{2}\left[(\rho u)|_{ijk}^{l}+(\rho u)|_{i+1,jk}^{l}\right]\approx\frac{1}{2}\left[\rho_{ijk}^{l}u_{ijk}^{l}+\rho_{i+1,jk}^{l}u_{i+1,jk}^{l}\right],$$

and analogous for the other terms. Substituting this into the discrete mass conservation from above, we obtain

$$\begin{aligned}\frac{\Delta x\Delta y\Delta z}{\Delta t}(\rho_{ijk}^{l+1}-\rho_{ijk}^{l}) = &-\frac{\Delta y\Delta z}{2}\left[\rho_{i+1,jk}^{l}u_{i+1,jk}^{l}-\rho_{i-1,jk}^{l}u_{i-1,jk}^{l}\right]\\ &-\frac{\Delta x\Delta z}{2}\left[\rho_{i,j+1,k}^{l}v_{i,j+1,k}^{l}-\rho_{i,j-1,k}^{l}v_{i,j-1,k}^{l}\right]\\ &-\frac{\Delta x\Delta y}{2}\left[\rho_{ij,k+1}^{l}w_{ij,k+1}^{l}-\rho_{ij,k-1}^{l}w_{ij,k-1}^{l}\right].\end{aligned}$$

Assuming that the density is known at time level l we can write down the update step as

$$\begin{aligned}\rho_{ijk}^{l+1} = \rho_{ijk}^{l} &-\frac{\Delta t}{2\Delta x}\left[\rho_{i+1,jk}^{l}u_{i+1,jk}^{l}-\rho_{i-1,jk}^{l}u_{i-1,jk}^{l}\right]\\ &-\frac{\Delta t}{2\Delta y}\left[\rho_{i,j+1,k}^{l}v_{i,j+1,k}^{l}-\rho_{i,j-1,k}^{l}v_{i,j-1,k}^{l}\right]\\ &-\frac{\Delta t}{2\Delta z}\left[\rho_{ij,k+1}^{l}w_{ij,k+1}^{l}-\rho_{ij,k-1}^{l}w_{ij,k-1}^{l}\right].\end{aligned}$$

(b)

There are three types of special cases at the boundary. The first one is the corner case. Here, the cell to be updated is located in one of the four corners of the cube and three ghost cells need to be introduced to perform the time update, since only three of the six cells required to do the update are part of the domain.
The second type is the edge case. Here, the cell to be updated is located in on of the twelve edges of the cubed domain but not in the corners. We need to introduce two ghost cells and density values at these to complete the update step.
The third type is the side case. Here, the cell to be updated is located at one of the six sides of the cubed domain. We need to introduce one ghost cell and a density value for it to make the update step.
There are different possibilities to set values for these ghost cells. They should come from the specific application as meaningful boundary condition. Without having the authority, since I am not a physicist nor an engineer, I could imagine that periodic boundary conditions could make sense here due to conservation of mass. Or one could set them to the same value as the closest cell inside the domain. Since ρ is after all a continuous function, it makes sense that when Δx is small, these values have to be close anyway.

Exercise 5

(a)

Exercise 6

Exercise 7

Exercise 8