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

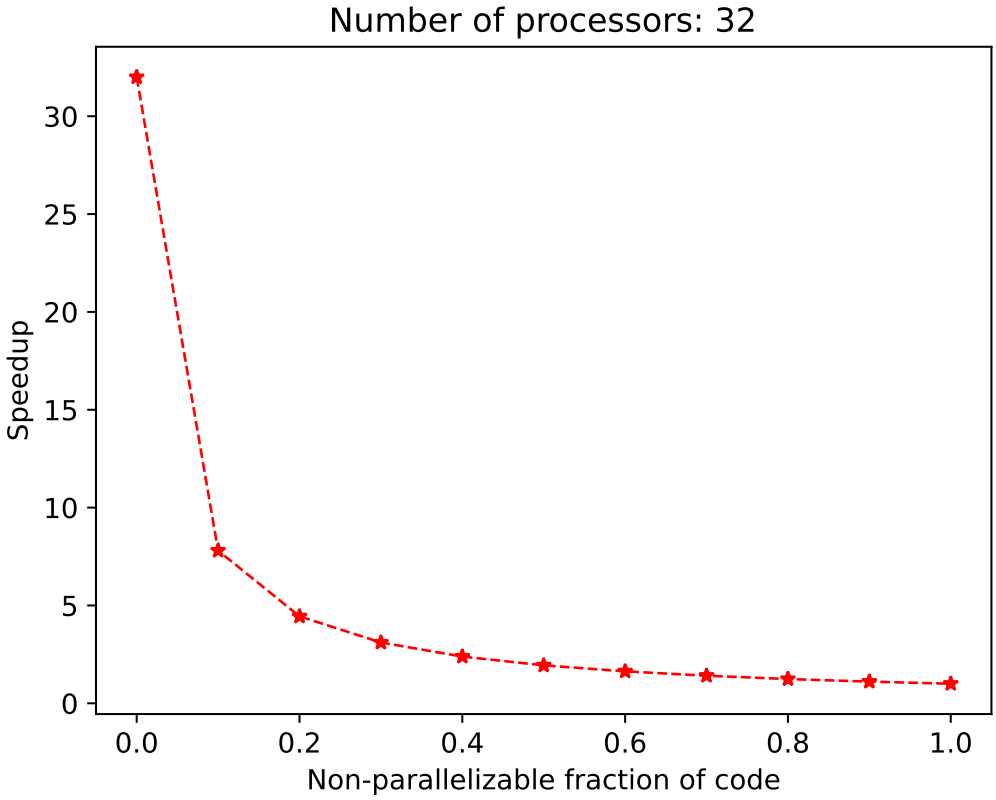

Figure 1 Speedup for a fixed number of processors over the fraction of non-parallelizable code

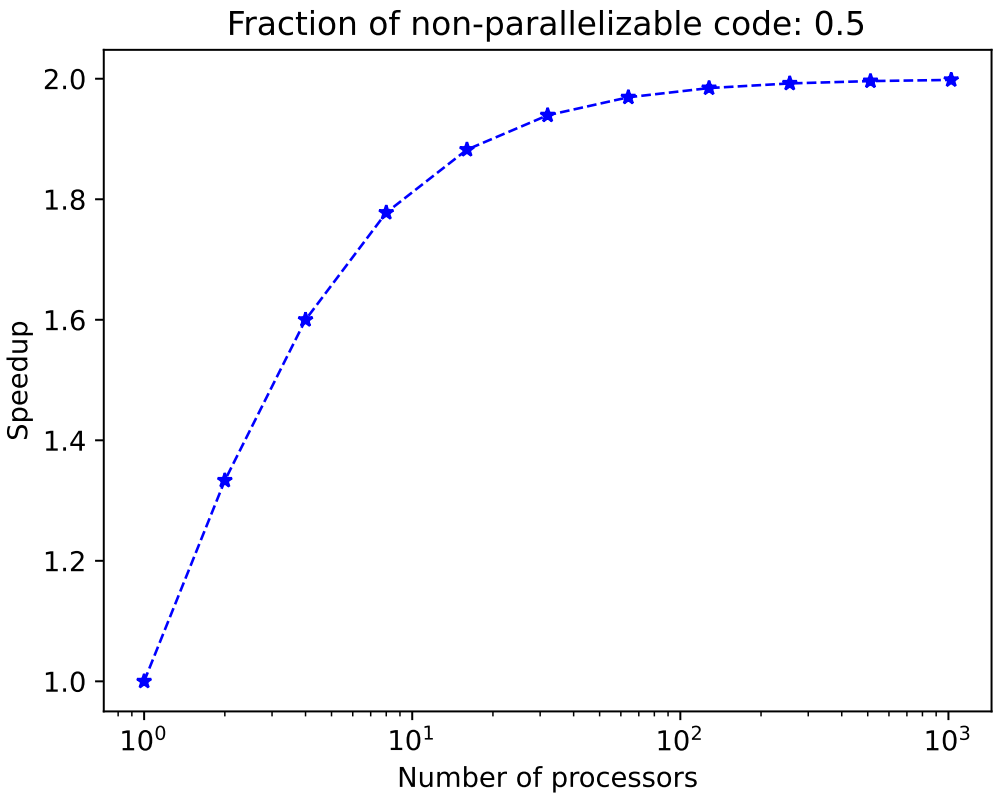

Figure 2 Speedup for a fixed fraction of non-parallelizable code over the number of processors

(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:

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

Independent of n, this is executed on three processors (washing, drying, folding). Hence, the parallel efficiency is given by,

$$E = \frac{T_1}{3T_n} = \frac{3n}{3 + 4n + 2} \to 0.75 \text{ as } n \to \infty.$$

The maximum parallel efficiency is thus (asymptotically) 0.75.

(b)

We obtain the following values for the parallel efficiency:

No. of students	1	2	3	4	5	6	7	8	9	10	11	12
Parallel efficiency	0.333	0.461	0.529	0.571	0.6	0.62	0.636	0.648	0.658	0.666	0.673	0.679

With twelve students we are in the 10% range, i.e. $\frac{0.75-0.679}{0.75} \leq 0.1$.

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)

Let A be the discrete laplacian as given on slide 22 from unit 2. We find

$$Ac = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \; Ar = \begin{pmatrix} -4 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -4 \end{pmatrix}, \; Ap = \begin{pmatrix} 14 \\ -2 \\ -2 \\ -2 \\ -2 \\ -2 \\ 14 \end{pmatrix}, \; As = \begin{pmatrix} -1 \\ 0 \\ -1 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}, \; Ao = \begin{pmatrix} -3 \\ 4 \\ -4 \\ 4 \\ -4 \\ 4 \\ -3 \end{pmatrix}.$$

The continuous laplace operator in one dimension is simply the second derivative. Hence, the discretization gives us an approximation of the second derivative, which is zero for the constant, zero for the ramp, constant for the parabola, nonsense for the step because this one is not differentiable at zero (not even continuous ...), and another ocsillatory function for an oscillatory function (second derivative of sin gives sin again). The scaling and the signs are a bit off above for the second derivatives. This is because to be truly a discretization of the continuous laplace operator, one needs a factor of $-\frac{1}{4}$ in front of the matrix A on slide 22.

(b)

With A the discrete Laplacian we have $A^{-1}l_4 = u_4$. This is because by definition of the inverse $A^{-1}A = I$, where I denotes the identity matrix. Moreover,

$$A^{-1}u_4 = A^{-1}_{:,4} = \frac{1}{8}\begin{pmatrix} 4 \\ 8 \\ 12 \\ 16 \\ 12 \\ 8 \\ 4 \end{pmatrix}.$$

This is because $A^{-1} = A^{-1}I$. One could argue, that integrating a Dirac distribution gives a step function and integrating a step function gives a ramp.

(c)

Applying the inverse of the laplacian to the constant gives

$$A^{-1}c = \begin{pmatrix} 3.5 \\ 6 \\ 7.5 \\ 8 \\ 7.5 \\ 6 \\ 3.5 \end{pmatrix},$$

which is a parabola. This makes sense because its second derivative is constant.

Exercise 6

(a)

The Top500 HPL List is ranked by the High-Performance Linpack (HPL) Benchmark. The latter uses the LU decomposition algorithm with partial pivoting to solve a dense system of linear equations. The associated matrix has randomly distributed elements between -1 and 1. The performance is measured in flop/s and the highest sustained number of flops/s achievable during this computation determines a supercomputer's ranking in the list.
The Top124 list is ranked by the High Performance Conjugate Gradient (HPCG) benchmark. the latter uses the conjugate gradient algorithm to solve large, sparse, symmetric, positive-definite linear systems. Performance is measured in flop/s and the highest sustained number of flops achievable during this computation determines a supercomputer's ranking in the list.

(b)

Sum	HPL list	Green list	Manufacturer	Model	Nickname	Country
19	1	18	HPE	HPE Cray EX255a	El Capitan	USA
21	7	14	HPE	HPE Cray EX254n	Alps	Switzerland
22	10	12	HPE	HPE Cray EX255a	Tuolumne	USA
24	18	6	ParTec/EVIDEN	BallSequana XH3000	JETI	Germany
24	2	22	HPE	HPE Cray EX235a	Frontier	USA
26	5	21	HPE	HPE Cray EX235a	HPC6	Italy
29	13	16	HPE	HPE Cray Ex254n	Venado	USA
33	20	13	HPE	HPE Cray Ex255a	El Dorado	USA
33	8	25	HPE	HPE Cray Ex235a	LUMI	Finland
49	30	19	HPE	HPE Cray EX235a	Adastra	France

Exercise 7

(a)

Nine challenges:

1. Scalable multicore systems bring a growing cost of communication relative to computation. Especially across different nodes (multicore processors) the cost of data transfer becomes large

2. Static distribution of tasks and adaptive multiscale algorithms introduce load imbalances from dynamically changing computation

3. Since 32-bit (single precision) operations are at least twice as fast as 64-bit operations on modern architectures and have smaller storage and memory traffic, we need mixed precision algorithms to utilize heterogenous hardware effectively.

4. Memory movement is increasingly expensive compared with the cost of computation, need to develop and study communication avoiding algorithms

5. Numerical libraries need to be able to adapt to possibly heterogeneous environments to remain the same for the user but be consistent independent of scale and processor heterogeneity. (Auto-tuning)

6. Due to scale and complexity of supercomputer architectures, faults become often and current restarting techniques are not scalable to highly parallel systems. New faults will occur before the application can be restarted. (Fault Tolerance and Robustness)

7. Energy consumption is becoming a problem. Depends on hardware and software.

8. One needs to study sensitivity of high fidelity models to parameter variability and uncertainty.

9. With more powerful machines the dimensions of the problems increase, but algorithms that work well for moderate dimensions might fail in subtle ways for larger problems. Reproducability independent of scale and number of cores is an open issue.

(b)

In today's exascale environment challenge 6 becomes much more critical. Most restarting techniques are not scalable to highly parallel systems, so that a specific application might not be able to restart before the next fault occurs. This results in a program getting stuck. For algorithms in computing molecular dynamics this is a hard challenge because these computations are typically long-running and data intensive. An adaption could be to use redundant computing. If some redundant computations get lost due to a fault they don't need to be redone and the algorithm can restart quicker. Otherwise, one can let algorithms adaptively run on a dynamic number of cores, so that in case of a fault the algorithm can continue running after a unit of the computer is shut off. This would include ongoing error measurements and decisions on whether possibly corrupted data should be excluded combined with modular composition of the simulation.

Exercise 8

(a)

The Mandates are as follows:

1. Better resolve a model's full, natural range of length or time scales
2. Accommodate physical effects with greater fidelity
3. Allow the model degrees of freedom in all relevant dimensions
4. Better isolate artificial boundary conditions
5. Combine multiple complex models
6. Solve an inverse problem or perform data assimilation
7. Perform otimization or control
8. Quantify uncertainty
9. Accomplish predictions without physical models using statistical models based on large data sets

(b)

Accomodating physical effects with greater fidelity is an important opportunity in aerodynamic modeling for e.g. aircraft design. This includes the modeling of turbulent flows and shock waves. Still, many costly experiments need to be carried out in the wind tunnel including people and material as their virtual numerical pendants in computational fluid dynamics are not feasible. This includes especially numerical simulations of high-Reynolds-number flows. Also, wind tunnel experiments are physically not optimal to see all possible effects on a moving airplane as even here many approximations need to be made to translate the results into a real life situation. One of the potentials of HPC in this area is that given enough computation power, simulations might become more accurate than wind tunnel experiments. This raises a second important point as it will not only safe costs in the production, but it will make airplanes safer and more efficient in practice.