P. Koumoutsakos., S. Martin
ETH Zentrum, ML H44
CH-8092 Zürich

Mockup Exam

Issued: December 17, 2021, 12:15

Last Name:	
First Name:	
Student ID:	

With your signature you confirm that you:

- Have read the exam directives
- You solved the exam without any unauthorized help
- You wrote your answers following the outlined directives

Signature: _____

Exam directives. In order to pass the exam, the following requirements have to be met:

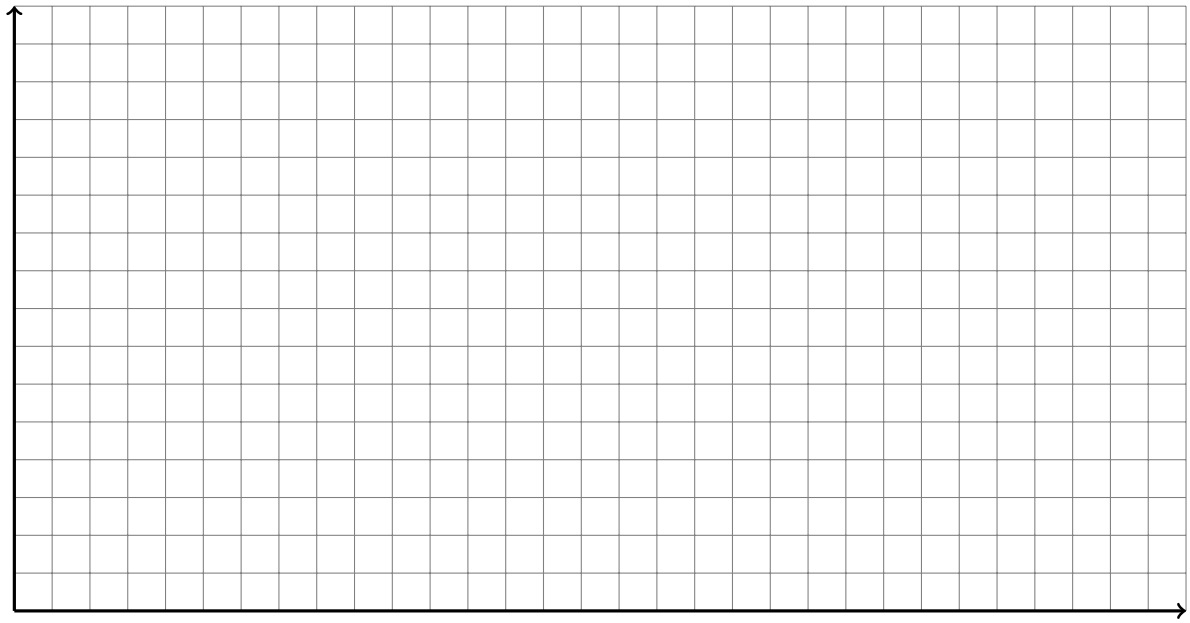
- Clear your desk (no cell phones, cameras, etc.): on your desk you should only have your Legi, your pen and your notes. We provide you with the necessary paper and exam sheets.
- Carefully read the first pages of the exam. Write your name and student ID where requested. Before handing in the exam, **SIGN ON THE FIRST PAGE**.
- Your notes (personal summary) should consist of **no more than four** A4 sheets (eight pages). The personal summary **must be handwritten**. You are not allowed to bring a copy of somebody else's summary.
- Your answers should be handwritten in blue or black pen (no pencils), clearly readable and in English. Only one answer per question is accepted. Invalid answers should be clearly crossed out.
- To answer new questions (e.g. Question 1, not sub-questions!), always use a new page. On the top-right corner of every page write your complete name and Legi-ID. Unless otherwise noted in the question, you should always hand-in your answers on paper!
- You must hand in: the exam cover, any extra notes provided by us, the sheets with the exam questions and your solutions. The exam will not be accepted if any of these items are missing.
- If something is disturbing you during the exam or preventing you from peacefully solving the exam, please report it immediately to an assistant. Later complaints will not be accepted.

Question 1: Parallel Scaling

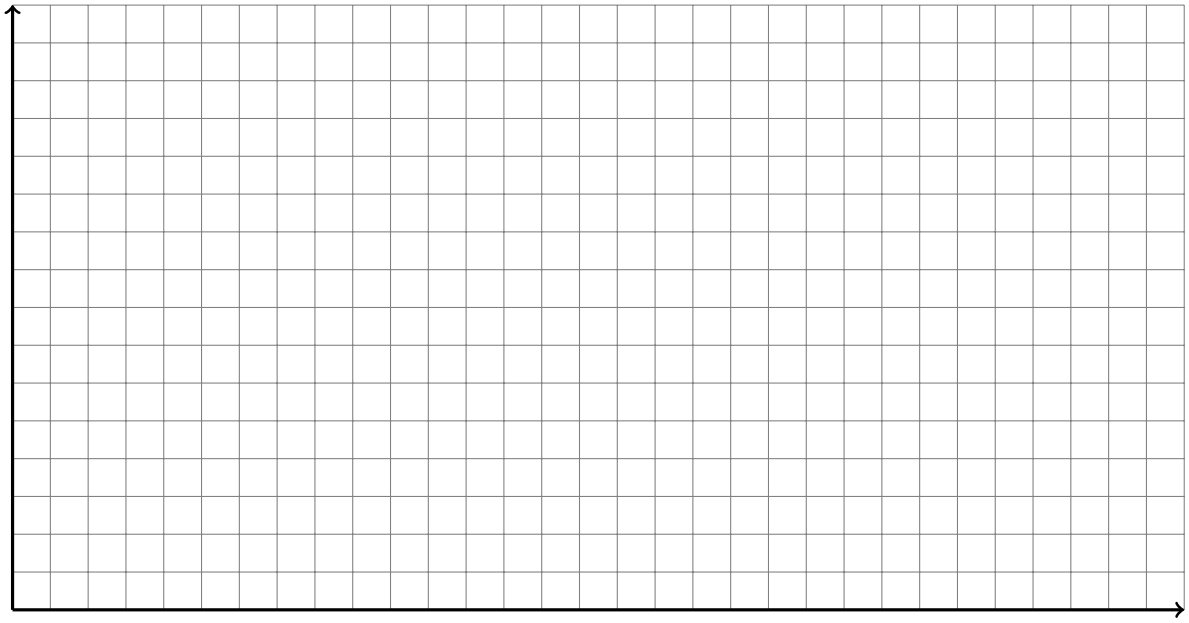
a) Assume we implemented an $\mathcal{O}(N^2)$ brute force solver for an N-body problem. The following table reports timing results for the solver for various number of particles N on P processor cores with a fixed number of time steps:

	runtime [s]			
P	$N = 1000$	$N = 2000$	$N = 4000$	$N = 8000$
1	12.0	60.0	216.0	816.0
2	6.0	30.0	108.0	408.0
4	4.0	15.0	54.0	204.0
8	3.0	9.0	31.0	110.0
16	2.4	6.0	21.6	54.4

1. Draw at least four points of the strong scaling plot for this program using the value for $N = 1000$ at $P = 1$ as a reference. On the solution sheet show all steps of your calculations. Don't forget to label the axes.



2. Draw at least three points of the weak scaling plot for this program, using the value for $N = 1000$ at $P = 1$ as a reference to estimate the parallelization overhead. On the solution sheet show all steps of your calculations. Don't forget to label the axes.

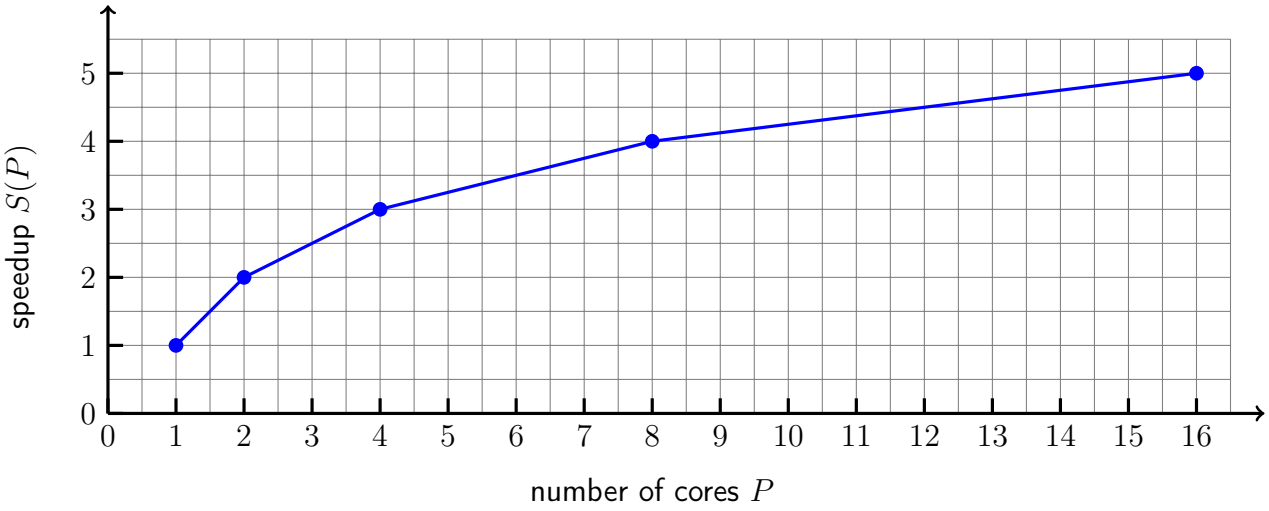


The strong scaling speedup $S(P) = T(1) \,/\, T(P)$ should be computed from

	runtime [s]			
P	$N = 1000$	$N = 2000$	$N = 4000$	$N = 8000$
1	12.0	60.0	216.0	816.0
2	6.0	30.0	108.0	408.0
4	4.0	15.0	54.0	204.0
8	3.0	9.0	31.0	110.0
16	2.4	6.0	21.6	54.4

N	1000	1000	1000	1000	1000
P	1	2	4	8	16
$T(P)$	12.0	6.0	4.0	3.0	2.4
$S(P)$	1.0	2.0	3.0	4.0	5.0

with the corresponding strong scaling plot

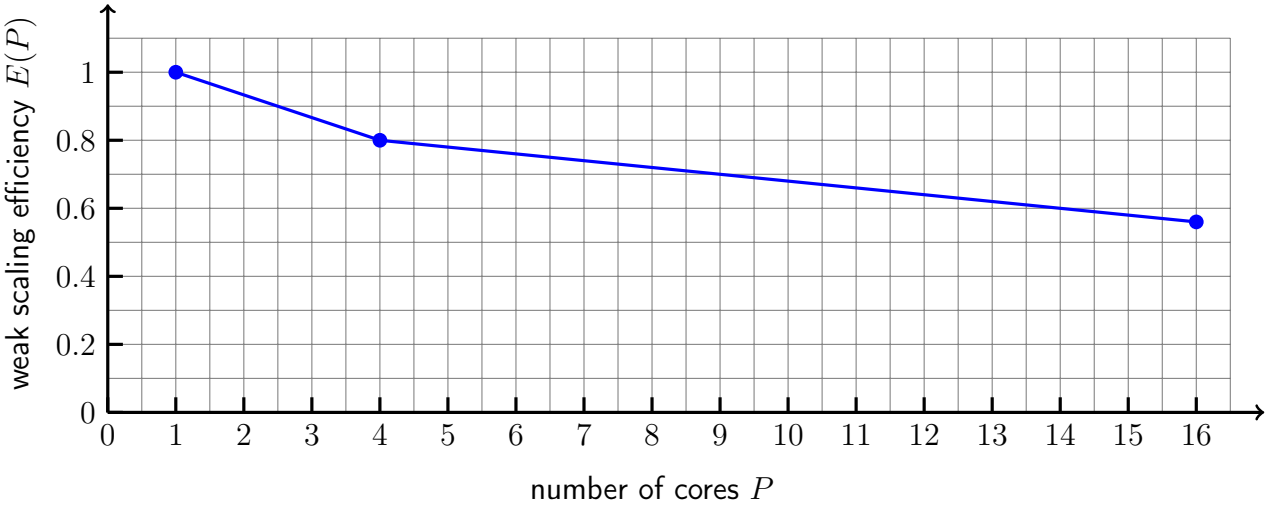


The weak scaling efficiency $E(P) = T(1) \,/\, T(P)$ should be computed from

	runtime [s]			
P	$N = 1000$	$N = 2000$	$N = 4000$	$N = 8000$
1	12.0	60.0	216.0	816.0
2	6.0	30.0	108.0	408.0
4	4.0	15.0	54.0	204.0
8	3.0	9.0	31.0	110.0
16	2.4	6.0	21.6	54.4

N	1000	2000	4000
P	1	4	16
$T(P)$	12.0	15.0	21.6
$E(P)$	1.0	0.8	0.56

with the corresponding weak scaling plot



b) 1. Assume you have 10 cores that you can use to solve a problem in parallel and that 98% of your code is theoretically possible to parallelize. Can you get a speedup of 7? If so, how many cores are needed at least?

Fraction of parallel code: $q = 0.98$.

Speedup according to Amdahl's law with unknown number of cores P: $S = \frac{1}{(1-q)+\frac{q}{P}}$.

Required number of cores is such that $S = 7 \;\Rightarrow\; P = \frac{q}{\frac{1}{S}-(1-q)} = 7.97$.

$\Rightarrow$ Need 8 cores and the speedup can be achieved since $8 \leq 10$.

2. A programmer parallelized so far a part of the code that accounts for 60% of execution time. What is the current maximum speedup he can hope to achieve?

Fraction of parallel code: $q = 0.6$.

Maximum speedup at $P \to \infty$: $S = \frac{1}{1-q} = 2.5$

Question 2: Parallel Dependencies and Bug Hunting

a) Identify and explain any issues in the following OpenMP code snippet and propose a solution.

```
#pragma omp parallel for collapse(2)
for (int i=0; i<N; i++) {
   y[i] = 0;
   for (int j=0; j<N; j++)
    y[i] += A[i][j] * x[j];
}
```

- Issue: the collapse clause requires that the two loops are adjacent (*perfectly nested*), without code between them (y[i] = 0).
- Minor possible issue: race condition on y[i]
- Solution: Remove the collapse clause or move the initialization of the y-vector in the separate parallel loop (partial solution).

b) In the following fragments assume that all buffers have been allocated with sufficient size. Moreover, `rank` and `size` denote the rank of each process and the total number of MPI processes, respectively.
For each fragment note whether it deadlocks or not and explain why. In case of deadlock, propose a solution. Report any possible performance issues.

```
// block1.c
int ireq = 0;
for (int p=0; p<size; p++)
if (p!=rank)
MPI_Isend(sbuffers[p],buflen,MPI_INT,p,0,comm,&(reqs[ireq++]));
for (int p=0; p<size; p++)
if (p!=rank)
MPI_Recv(rbuffer,buflen,MPI_INT,p,0,comm,MPI_STATUS_IGNORE);
MPI_Waitall(size-1,reqs,MPI_STATUSES_IGNORE);
```

```
// block2.c
int ireq = 0;
for (int p=0; p<size; p++)
if (p!=rank)
MPI_Irecv(rbuffers[p],buflen,MPI_INT,p,0,comm,&(reqs[ireq++]));
MPI_Waitall(size-1,reqs,MPI_STATUSES_IGNORE);
for (int p=0; p<size; p++)
if (p!=rank)
MPI_Send(sbuffer,buflen,MPI_INT,p,0,comm);
```

- Block1: No deadlock. Performance issues: MPI_Isend before MPI_Recv and/or MPI_IRecv instead of MPI_Recv and/or one buffer for Recv.
- Block2: Deadlock. MPI_Waitall must be called after MPI_Send's

c) For a known vector $A \in \mathbb{R}^N$, you are asked to compute

$$B_i = \sum_{j=0}^{K-1} A_{i+j} \ , \ i = 0, \ldots, N-K-1, \ , \ K \ll N \tag{1}$$

The following code is a serial implementation of the computation shown above:

```
1   // Assume B[...] are initialized to 0.
2   for (int i = 0; i < N - K; ++i)
3       for (int j = 0; j < K; ++j)
4           B[i] += A[i + j];
```

This loop can be parallelized by adding `#pragma omp parallel for` before line 2. Assuming T available threads, the complexity of that loop would by $\mathcal{O}(NK/T)$. Show that $B_i = B_{i-1} + A_{K+i-1} - A_{i-1}$, $i = 1, \ldots, N - K - 1$ and provide a serial code that implements this recursive relation (you still need to use the non-recursive formula to compute B_0).

```
// Assume B[...] are initialized to 0.
for (int j = 0; j < K; ++j)
B[0] += A[j];
for (int i = 1; i < N-K; ++i)
    B[i] = B[i-1] + A[K+i-1] - A[i-1];
```

d) The complexity of the serial implementation you provided in the previous subquestion is $\mathcal{O}(N + K)$. Propose a parallel implementation with OPENMP and T threads with has lower complexity. You can assume that N is divisible by T. What is the complexity of your solution? Hint: the optimal solution can be obtained by using only one `#pragma omp parallel` region without any `#pragma omp for` directives.

The following implementation has a complexity of $\mathcal{O}(K + N/T)$:

```
// Assume B[...] are initialized to 0.

#pragma omp parallel
{
     int T = omp_get_max_threads();
     int tid = omp_get_thread_num();
     int myN = (N-K)/T;
     int istart = myN * T;
     if (tid == T-1) myN += (N-K)%T;
     for (int j = 0; j < K; ++j)
         B[istart] += A[istart+j];

     for (int i = istart + 1; i < istart + myN; ++i)
         B[i] = B[i-1] + A[K+i-1] - A[i-1];
}
```

Question 3: Operational Intensity & Roofline Model

The operational intensity is a metric that relates the amount of work W (operations) to the number of bytes Q (data) that need to be transferred and is defined as

$$I = \frac{W}{Q} \quad [\text{ops/byte}]. \tag{2}$$

Your tasks

a) Determine the asymptotic bounds on the operational intensity $I(n)$ for the following matrix/vector operations, where n is the dimension of the vector. State your assumptions.

1. DAXPY: $y = \alpha x + y \quad \alpha \in \mathbb{R}; x, y \in \mathbb{R}^n$
2. SGEMV: $y = Ax + y \quad x, y \in \mathbb{R}^n; A \in \mathbb{R}^{n \times n}$
3. DGEMM: $C = AB + C \quad A, B, C \in \mathbb{R}^{n \times n}$
   Hint 1: Assume that A, B and C fit into the cache at the same time.
   Hint 2: Remember that double precision numbers are typically stored in 8 bytes.

The operational intensity I is used to characterize a code and reveals information about the expected utilization of the architecture's execution units and memory bandwidth.
In practice, operations are defined by floating point operations (FLOP) and memory transfer is defined by access from DRAM to the closest processor cache.

Here, we assume that we start with a cold cache and memory transfers are defined from DRAM to the lowest level cache. Furthermore, we define a count W and a count Q which correspond to total number of flops and total number of memory accesses (read and write), respectively.

DAXPY (5 points) We write the operation in index notation

$$y_i = \alpha x_i + y_i \quad i \in [1, n], \tag{3}$$

where α is a constant scalar. For each element i we count

- $w_i = 1_{\text{mul}} + 1_{\text{add}} = 2$ flops
- $q_i = 2_{\text{read}} + 1_{\text{write}} = 3$ doubles

For n elements we get $W = nw_i$ and $Q = nq_i$. The number of actual bytes transferred over the bus depends on the precision of the operation. The DAXPY is double precision for which each element is of size $s = 8$ bytes/double. Therefore, the asymptotic bound on the operational intensity is

$$I(n) = \frac{W}{Qp} = \frac{1}{12} \text{ flops/byte} \Rightarrow \mathcal{O}(1) \text{ flops/byte}. \tag{4}$$

Points:

- 2 for correct W
- 2 for correct Q
- 1 for correct asymptotic behaviour.

SGEMV (5 points) We proceed similar as above. Note that we apply the Einstein summation convention unless noted otherwise. Matrix elements are written in lowercase letters.

$$y_i = a_{ij} x_j + y_i \quad i, j \in [1, n]. \tag{5}$$

For each element i we count

- $w_i = n(1_{\text{mul}} + 1_{\text{add}}) + 1_{\text{add}} = 2n + 1$ flops
- $q_i = 2n_{\text{read}} + 1_{\text{read}} + 1_{\text{write}} = 2n + 2$ floats

For n elements we compute $W = nw_i = 2n^2 + n = 2n^2 + \mathcal{O}(n)$ flops and $Q = nq_i = 2n^2 + 2n = 2n^2 + \mathcal{O}(n)$ floats memory accesses. The SGEMV is a single precision operation, thus $p = 4$ byte/float. To determine the asymptotic bound, we only keep the leading order of operations to get

$$I(n) = \frac{W}{Qp} = \frac{1}{4} \text{ flops/byte} \Rightarrow \mathcal{O}(1) \text{ flops/byte}. \tag{6}$$

Note that for recent processors, the machine balance I_B is roughly 5–10. Since $I(n)$ for the SGEMV is *constant* the operation is memory bound independent of the problem size n.

Points:

- 2 **for correct** W
- 2 **for correct** Q
- 1 **for correct asymptotic behaviour.**

DGEMM (5 points) For this problem, we further assume that the problem size n is small.

$$c_{ij} = a_{ik}b_{kj} + c_{ij} \quad i, j \in [1, n], \tag{7}$$

For each element i, j we count

- $w_{ij} = n(1_{\text{mul}} + 1_{\text{add}}) + 1_{\text{add}} = 2n + 1$ flops
- $q_{ij} = 3_{\text{read}} + 1_{\text{write}} = 4$

For n^2 elements we compute $W = n^2 w_{ij} = 2n^3 + \mathcal{O}(n^2)$ flops and $Q = n^2 q_{ij} \geq 4n^2$ memory accesses. The DGEMM is a double precision operation, thus $p = 8$ byte. The asymptotic bound is then given by

$$I(n) = \frac{W}{Qp} \leq \frac{n}{16} \Rightarrow \mathcal{O}(n). \tag{8}$$

The assumption of small n means that the three matrices A, B and C fit in the cache and must be read only once, where C is written once additionally. For example, if $n = 1000$ the memory footprint in double precision is 23 MB which will fit in the L3 cache of the Intel Xeon E5-2680 v3 on the Euler nodes. Therefore, the number of memory accesses M for this assumption gives a lower bound. A naive implementation of DGEMM will likely require more memory accesses for large n for which the operational intensity will become smaller. However, blocking strategies can be applied to increase temporal locality and therefore reduce cache misses for which the DGEMM operation can be optimized towards operational intensities that lie in the compute bound region.

Points:

- 2 **for correct** W
- 2 **for correct** Q
- 1 **for correct asymptotic behaviour.**

b) Consider the 1D diffusion equation in a periodic domain with length L

$$u_t(x,t) = \alpha u_{xx}(x,t), \tag{9}$$

$$u_0(x) = u(x,0) = \sin\left(\frac{2\pi}{L}x\right), \tag{10}$$

where $0 \leq x < L$, $t > 0$ and $\alpha > 0$ is the diffusion coefficient. You are asked to solve this problem numerically by using a second order centered finite difference scheme and the explicit Euler method to advance in time. The domain is discretized with N uniformly spaced grid points. Discretization of Equation (9) leads to

$$u_i^{n+1} = u_i^n + \frac{\Delta t \alpha}{\Delta x^2}\left(u_{i-1}^n - 2u_i^n + u_{i+1}^n\right), \tag{11}$$

where $u_i^n \approx u(x_i, t^n)$ is the approximate solution with $t^{n+1} = t^n + \Delta t$ and $x_i = i\Delta x$. The constant time step size and uniform grid spacing are denoted by Δt and Δx, respectively. Determine the operational intensity $I(N)$ for the scheme given in Equation (11).

(8 points) As the value at each grid point is computed in the same way, we count the flops and memory accesses per grid point.

In Equation (11) we count $w_i = 5$ flops per grid point. There are 3 extra operations that we do not count as they are between constant values $\frac{\Delta t \alpha}{\Delta x^2}$ and thus can be precomputed.

To count the memory operations (reads and writes) from and to DRAM we can use different approaches:

1. Assume there is no cache and therefore every read and write operation is done directly in main memory. This assumption leads to a worst case scenario to the number of memory accesses.
2. Assume a cache of infinite size, thus each data point is read once and written once at most. This assumption leads to the best case scenario as it only considers compulsory cache misses.
3. Assume a cache of limited size and estimate the number of memory operations given the memory layout used and the principles of data locality (temporal and spatial).

Following the three approaches listed above we have:

1. Assuming there is no cache, we have 4_{read} and 1_{write} operation per grid point (u_i^n is read twice and the computation could be easily reformulated to reduce the number of read operations). The total number of memory accesses is therefore $q_i = 5$.
2. With an infinite sized cache, we only need two memory operations per grid point (1_{read} and 1_{write}). Hence, the total number of memory accesses is $q_i = 2$.
3. In reality we deal with a finite size cache. For the simple 1D stencil of Equation (11) it is safe to assume that the data u_{i-1}^n and u_i^n is already in cache from the previous iteration and only u_{i+1}^n needs to be loaded from DRAM. Thus, the 1D problem computed on a sufficiently large grid will be dominated by compulsory cache misses (rather than capacity misses) similar to the infinite cache above, i.e. $q_i = 2$. However, capacity misses become more problematic for stencils in higher dimensions on sufficiently large grids and even more so for high-order stencil schemes. In such cases a blocked memory layout is used to increase temporal locality of the data in the cache.

The operational intensities for the three cases are (assuming double precision $p = 8$ bytes)

1. $I_1 = \frac{5}{40} = 0.125$ flop/byte
2. $I_2 = \frac{5}{16} \approx 0.313$ flop/byte
3. $I_3 = I_2$

Above, we have used that (same notation as in the first part of this question)

$$I = \frac{W}{Q} = \frac{nw}{nq} = \frac{w}{q}. \tag{12}$$

For single precision the numbers double.

We note that it is not always trivial to estimate the work of the caches and therefore one approach is to localize the operational intensity within a range instead of finding the exact value. An advantage of estimating the range, is that we are able to also estimate a lower bound on the performance.

Points:

In order to get the full amount of points, it is sufficient to give one of the above solutions.

- **1 for realizing that $\frac{\Delta t \alpha}{\Delta x^2}$ can be precomputed,**
- **2 for giving the correct number of operations per grid point f and/or the total number of operations $W = nw_i$,**
- **4 for correctly stating the assumptions about cache size and number of memory accesses per grid point q_i and/or total memory accesses $Q = nq_i$,**
- **1 for correctly calculating the operational intensity I**

c) In your position as an HPC expert, you are presented three possible systems (A, B, and C) for purchase, each with unique peak CPU performance and memory bandwidth configurations. Complete the table below indicating, for each system, whether the previously analyzed operations are memory or compute bound. Assume $n = 256$ for all cases. Write down all your assumptions and calculations.

(5 points) Recapping from the previous answers:

$$I_{DAXPY}(n) = \frac{1}{12} \text{ flops/byte} = 0.083 \text{ flops/byte} \tag{13}$$

$$I_{SGEMV}(n) = \frac{1}{4} \text{ flops/byte} = 0.25 \text{ flops/byte} \tag{14}$$

	System A 10 GFLOP/s 70 GB/s	System B 50 GFLOP/s 125 GB/s	System C 70 GFLOP/s 265 GB/s
DAXPY			
SGEMV			
DGEMM			
1D Diffusion			

For DGEMM, $n = 256$ is within what can be stored in cache, thus we assume no capacity misses and then we take the asymptotic operational intensity bound.

$$I_{DGEMM}(256) = \frac{256}{16} = 16 \text{ flops/byte} \tag{15}$$

For Diffusion, we take the the steady state intensity (assuming no compulsory cache misses):

$$I_{Diffusion} \approx 0.313 \text{ flop/byte} \tag{16}$$

Now we calculate the operational intensity (IR) that corresponds to the ridge point for the three systems. The is one such that finds the same performance limit from memory bandwidth (BW) as that of peak CPU performance (P):

$$IR * BW = PP \tag{17}$$

$$IR = \frac{PP}{BW} \tag{18}$$

$$IR_A = \frac{10}{70} = 0.142 \text{ flop/byte} \tag{19}$$

$$IR_B = \frac{50}{125} = 0.4 \text{ flop/byte} \tag{20}$$

$$IR_C = \frac{70}{265} = 0.264 \text{ flop/byte} \tag{21}$$

To complete the table, it is now sufficient to determine whether the operational intensity of each algorithm is smaller than the system's IR point. In that case, it will be memory bound. Otherwise, it will be compute bound.

Points:

In order to get the full amount of points, it is sufficient to give one of the above solutions.

- **1 for correct answer for DAXPY,**

	System A 10 GFLOP/s 70 GB/s 0.142	System B 50 GFLOP/s 125 GB/s 0.4	System C 70 GFLOP/s 265 GB/s 0.264
DAXPY (0.083)	Memory Bound	Memory Bound	Memory Bound
SGEMV (0.25)	Compute Bound	Memory Bound	Memory Bound
DGEMM (16)	Compute Bound	Compute Bound	Compute Bound
1D Diffusion (0.313)	Compute Bound	Memory Bound	Compute Bound

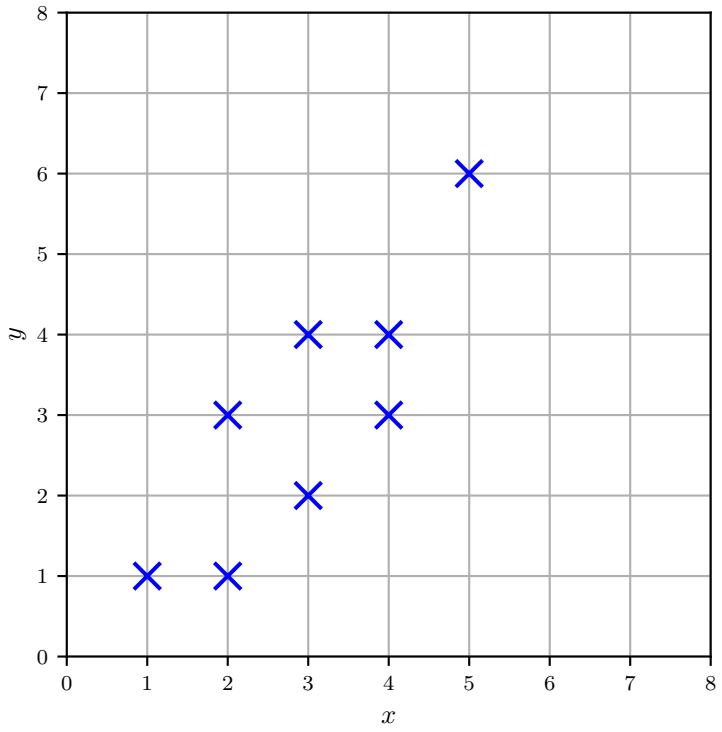
- 1 for correct answer for SGEMV,
- 1 for correct answer for DGEMM,
- 2 for correct answer for 1D Diffusion

Question 4: Principal Component Analysis

In the following, you are given a collection of $N = 8$ data points in a two dimensional space:

$$X = \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 2 & 3 \\ 3 & 2 \\ 3 & 4 \\ 4 & 3 \\ 4 & 4 \\ 5 & 6 \end{pmatrix}, \tag{22}$$

with $X \in \mathbb{R}^{N \times D}$, plotted below:



a) Center the data and calculate the data covariance matrix.

The mean is given by

$$\overline{X} = \frac{1}{8} \begin{pmatrix} 24 & 24 \end{pmatrix} = \begin{pmatrix} 3 & 3 \end{pmatrix}. \tag{23}$$

The centered data are thus:

$$\tilde{X} = X - \overline{X} = \begin{pmatrix} -2 & -2 \\ -1 & -2 \\ -1 & 0 \\ 0 & -1 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \\ 2 & 3 \end{pmatrix}. \tag{24}$$

The data covariance matrix is

$$C = \frac{1}{N-1}\tilde{X}^T\tilde{X} = \frac{1}{7}\begin{pmatrix} 4+1+1+1+1+4 & 4+2+1+6 \\ 4+2+1+6 & 4+4+1+1+1+9 \end{pmatrix} \implies$$
$$C = \frac{1}{7}\begin{pmatrix} 12 & 13 \\ 13 & 20 \end{pmatrix} \approx \begin{pmatrix} 1.71 & 1.86 \\ 1.86 & 2.86 \end{pmatrix}$$

TOTAL POINTS: 6

- 1 pts for correct mean
- 2 pts for correct centered data
- 1 pts for correct formula of covariance matrix (formula of biased estimate with N instead of $N-1$ in the denominator is also correct)
- 2 pts for correct covariance matrix

b) You are given that the eigenvalues of the covariance matrix are $\lambda_1 = 4.23$ and $\lambda_2 = 0.34$. **Compute** the principal eigenvector $\boldsymbol{u}_1$ of the data covariance matrix and **sketch** it in the figure.

Hint: Use the fact that $C\boldsymbol{u} = \lambda\boldsymbol{u}$ and $|\boldsymbol{u}|_2 = 1$ for an eigenvector, and draw the eigenvector starting from the mean. Do not perform eigendecomposition.

We know that $C\boldsymbol{u}_1 = \lambda_1\boldsymbol{u}_1$, where $\boldsymbol{u}_1$ is the principal eigenvector. Assume that $\boldsymbol{u}_1 = (u_x, u_y)$, we have that

$$1.71\,u_x + 1.86\,u_y = 4.23\,u_x \implies 1.34\,u_x = u_y \quad \left(\implies u_x = 0.74\,u_y\right) \tag{25}$$

Moreover, we know that $|\boldsymbol{u}_1|_2 = 1$. As a consequence

$$u_x^2 + u_y^2 = 1 \implies u_x^2(1 + 1.34^2) = 1 \implies 2.8\,u_x^2 = 1 \implies \tag{26}$$
$$u_x = \sqrt{1/2.8} \approx 0.6. \tag{27}$$

As a consequence, $u_y = \sqrt{1 - u_x^2} = 0.8$. The leading eigenvector is thus $\boldsymbol{u}_1 = (0.6, 0.8)$. The eigenvector is plotted in Figure 1 starting from the mean.

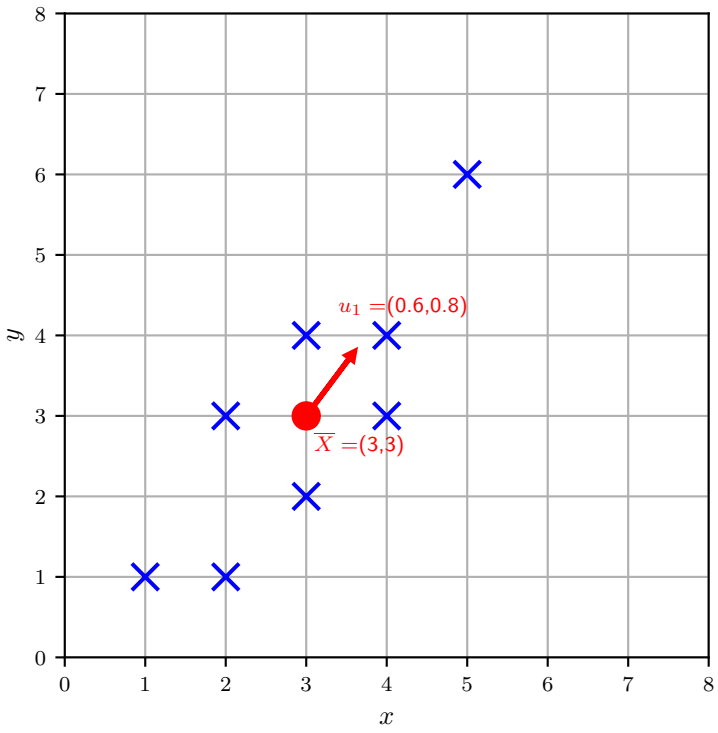


Figure 1

TOTAL POINTS: 5

- 1 pts for correct linear system equation (25)
- 1 pts for correct u_x
- 1 pts for correct u_y
- 2 pts for correct sketch of the eigenvector

c) If we project the data to a reduced order space using the principal component computed by the first eigenvector of the covariance matrix, how much variance of the data (in percentage) do we expect to retain in the reduced order space? Do not project the data.

Hint: Use the previously computed eigenvalues to answer this question.

The eigenvalues give a measure of the variance of the distribution of X on each projection. Projecting on the first eigenvector of the covariance matrix will lead to a reduced order space retaining:

$$\frac{\lambda_1}{\sum_{i=1}^{2} \lambda_i} = \frac{4.23}{4.23 + 0.34} = 0.925 = 92.5\%, \tag{28}$$

of the total data variance.

TOTAL POINTS: 2

- 1 pts for correct formula
- 1 pts for correct percentage

d) Use the principal component $\boldsymbol{u}_1$ computed in the previous questions to project the **centered** data in a reduced order subspace. Compute the variance of the data in the original 2-D

space, and the variance of the projected data. How much variance (in percentage) is retained in the reduced order space? Does this number agree with the value computed in the previous question, where we utilized the eigenvalues?
Hint: Use the unbiased estimator $\sigma_X^2 = \frac{1}{N-1}\sum_{i=1}^{N} X_i^2$ for the variance for a **centered** random variable X. In multiple dimensions, the total variance is summed across dimensions.

The variance $\sigma^2 = \mathbb{E}[x^2]$ in the original two dimensional space is

$$\sigma^2(X) = \sigma^2(\tilde{X}) = \frac{1}{7}\begin{pmatrix} 4+1+1+1+1+4 \\ 4+4+1+1+1+9 \end{pmatrix} = \frac{1}{7}\begin{pmatrix} 12 \\ 20 \end{pmatrix} = \begin{pmatrix} 1.71 \\ 2.86 \end{pmatrix}. \tag{29}$$

As a consequence, the total variance in the original space is $\sigma^2 = \sigma_x^2 + \sigma_y^2 \approx 4.57$.
Using the principal component $V_r = \boldsymbol{u}_1^T \in \mathbb{R}^{2\times 1}$, we can project the data on a one dimensional manifold. The projected (centered) data are given by

$$Z = \tilde{X}V_r = \tilde{X}\boldsymbol{u}_1^T = \begin{pmatrix} -2 & -2 \\ -1 & -2 \\ -1 & 0 \\ 0 & -1 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \\ 2 & 3 \end{pmatrix}\begin{pmatrix} 0.6 \\ 0.8 \end{pmatrix} = \begin{pmatrix} -2.8 \\ -2.2 \\ -0.6 \\ -0.8 \\ 0.8 \\ 0.6 \\ 1.4 \\ 3.6 \end{pmatrix}, \tag{30}$$

with mean $\overline{Z} = 0$.
The total explained variance in the reduced order subspace is:

$$\sigma^2(Z) = \mathbb{E}[(Z - \overline{Z})^2] \tag{31}$$

The variance is given by

$$\sigma_Z^2 = \frac{2.8^2 + 2.2^2 + 2\times 0.6^2 + 2\times 0.8^2 + 1.4^2 + 3.6^2}{7} = \frac{29.6}{7} = 4.23 \tag{32}$$

which corresponds to $4.23/4.57 = 92.5\%$ of the total variance, validating the value obtained using the eigenvalues.

TOTAL POINTS: 10

- 2 pts for correct variance in the original space (0.5 for σ_x^2, 0.5 for σ_y^2 and 1 for total variance)
- 2 pts for correct projected data Z
- 1 pts for correct mean $\overline{Z}$ (even if not explicitly reported, just be taken into acount that it is zero in the variance calculation)
- 2 pts for correct variance in the reduced order space σ_Z^2
- 3 pts for correct percentage

e) The computational cost of many data-driven algorithms (e.g. nearest neighbor search or classification) scales quadratically, i.e. $\mathcal{O}(N^2)$, or even cubically $\mathcal{O}(N^3)$ with the data dimension N. In the era of big data, many datasets have a large dimensionality rendering the direct application of such algorithms computationally intractable. How can the PCA be used to alleviate the problem? Use at most five sentences.

PCA can be used as a dimensionality reduction technique to project the data to a reduced order space, capturing as much data variance as possible. In this way, we first can apply PCA as a pre-processing step to the data and then use the desired algorithm on the reduced order tractable space.

TOTAL POINTS: 2

- 2 pts for correct answer (dimensionality reduction technique, pre-processing step, capturing the main features, projecting the space of the original problem in a tractable subspace, etc.)

Question 5: Particles MPI (30 points)

In this task you will implement an N-body solver describing a system of positively charged inertialess particles with MPI parallelization. The provided skeleton code

- seeds N particles on a unit circle

$$\boldsymbol{x}_i = (x_i, y_i) = \big(\cos\frac{2\pi i}{N},\ \sin\frac{2\pi i}{N}\big), \qquad i = 0, \ldots, N-1;$$

  distributing them over P ranks such that each rank takes N/P particles (assume N is divisible by P);
- performs time steps calling functions `Step()` and `PrintStat()`;
- writes the particle positions to files `init.dat` and `final.dat`.

a) Implement `Step()` which computes the forces and advances the particles

$$\boldsymbol{f}_i = \sum_{\substack{j=0 \\ j\neq i}}^{N-1} \frac{\boldsymbol{x}_i - \boldsymbol{x}_j}{|\boldsymbol{x}_i - \boldsymbol{x}_j|^3},$$

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \Delta t\, \boldsymbol{f}_i^t,$$

where $|\boldsymbol{x}| = \sqrt{x^2 + y^2}$ and Δt is a given time step.
Each rank can make $\mathcal{O}(P)$ MPI calls[1] and use $\mathcal{O}(N/P)$ bytes of memory. Your solution will be given more points for overlapping communication and computation. You may use `make` to build the executable, `make run` to run it on 2 processors and `make plot` to create `image.png` with particle positions as shown in Figure 2.

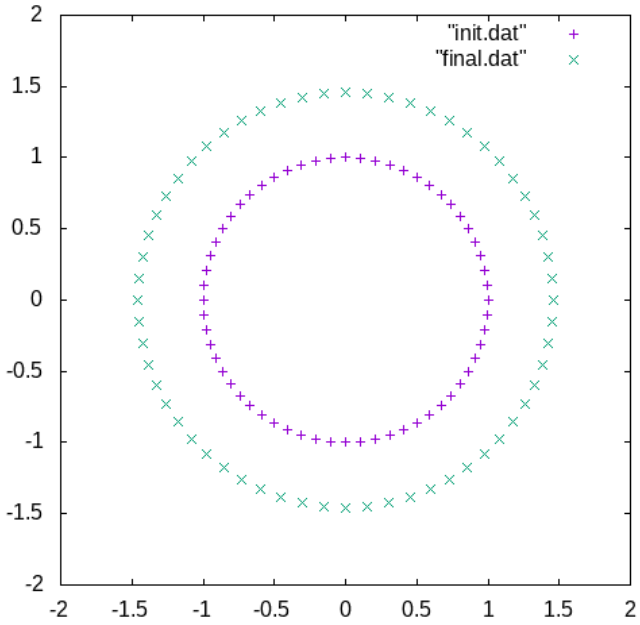


Figure 2: Expected output `image.png` produced by `make plot`.

[1] Notation $p = \mathcal{O}(q)$ means that $p < Mq$ for a large enough constant M.

b) Implement `PrintStat()` which computes and prints the mean and variance of the radial distance $r_i = |\boldsymbol{x}_i|$

$$\mu = \frac{1}{N}\sum_{i=0}^{N-1} r_i,$$

$$\sigma^2 = \frac{1}{N}\sum_{i=0}^{N-1} r_i^2 - \mu^2.$$

Expected output after 10 time steps:

```
mean=1          var=0
mean=1.0672     var=6.6613e-16
mean=1.1261     var=6.6613e-16
mean=1.1791     var=0
mean=1.2274     var=-8.8818e-16
mean=1.272      var=8.8818e-16
mean=1.3135     var=-2.2204e-16
mean=1.3524     var=4.4409e-16
mean=1.3891     var=4.4409e-16
mean=1.4239     var=1.3323e-15
mean=1.4571     var=0
```

Rounding errors may lead to differences up to 10^{-3} for the mean and 10^{-14} for the variance.

Question 6: Minimization with random walks (OpenMP) (30 points)

The following algorithm uses a set of random walks to minimize the function

$$f(x) = \int_0^x g(y)\,dy,$$

where $g(y) = y - 2$. The algorithm has parameters $N = 2000$, $T = 20$ and $M_{\max} = 20000$. Initialize N random walks x_i^t, $i = 1, \ldots, N$ on a regular grid in $[0, 1]$

$$x_i^0 = \frac{i - 0.5}{N}.$$

Perform T steps and at each step $t = 1, \ldots, T$,

- Advance the random walks

  $$x_i^t = x_i^{t-1} + \frac{1}{N}\xi_i, \quad i = 1, \ldots, N,$$

  where $\xi_i \sim \mathcal{N}(0, 1)$ are samples from the standard normal distribution.
- Compute an approximation of $f(x_i^t)$ using the Monte Carlo estimate

  $$f_i^t = \begin{cases} \frac{x_i^t}{M_i} \sum_{m=1}^{M_i} g(Y_m), & x_i^t \geq 0 \\ 0, & x_i^t < 0 \end{cases}, \quad i = 1, \ldots, N,$$

  where $M_i = \lfloor x_i^t M_{\max} \rfloor$ and $Y_m \sim \mathrm{Uniform}(0, x_i^t)$.
- Find the index

  $$j = \arg\min_i f_i^t$$

  of a random walk that achieves the minimum value. Store the corresponding $x_{\min}^t = x_j^t$ and $f_{\min}^t = f_j^t$.

The skeleton code `~/questions/random_walks/main.cpp` provides a serial implementation of the algorithm. Write your solution in the same file.

a) Parallelize the function `minimize()` using OpenMP by splitting the random walks among multiple threads (`TODO.a`). Make sure you do not introduce race conditions and that each thread has its own random generator initialized with a unique seed.

   Grading scheme:

   - Parallelized loop for initial positions: 2pts
   - Each thread has a random engine initialized with unique seed: 4pts
   - Parallelized loop for sampling: 3pts
   - Computed `xmin` and `fmin` without race conditions: 4pts
   - Used private variables for `xmin` and `fmin` to avoid critical sections for every random walk: 4pts
   - Ensured load balancing with `dynamic` schedule, or `static` schedule with a small chunk size: 3pts

b) Compute the total number of evaluations of function `integrand()` for each thread and store in array `evals` (`TODO.b`).

Grading scheme:

- Computed the number of evaluations without race conditions: 2pts
- Avoided false sharing for every random walk: 2pts (0pts if used atomic)

c) Answer the following questions (in your code, `TODO.c`):

- Run your program with two threads and report the values printed after `"Evaluations per thread (1e6):"`.
  Grading scheme: Any values consistent with the implementation (2pts).
  With load balancing:

  ```
  thread 0: 24.9913
  thread 1: 25.0026
  ```

  Without load balancing:

  ```
  thread 0: 12.4955
  thread 1: 37.493
  ```

- Is the number of evaluations the same for all threads? Explain in one sentence why.
  Grading scheme: Any of these possible answers (4pts)
  - Yes, because `schedule(dynamic)` is used.
  - No, because `integrate()` is called with fewer samples for smaller `x`.