S. Martin, J. H. Walther
ETH Zentrum, CLT
CH-8092 Zürich

Mock Exam

Issued: December 21, 2022, 08:00
Hand in: -

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.

Computer Setup

Carefully read the following instructions before starting with the exam!

Machines automatically logon in exam mode. To start the session please enter your **first and last name**. Afterwards, you will be asked to enter your **nethz** user name.

In your home directory (~) you will find the three directories `documentation/`, `questions/` and `questions_backup/`. The directory `documentation/` contains the course material files **we** provide. The directory `questions/` will contain **your** solutions. Only files in the directory `questions/` will be fetched from your local machine and considered for the grading of the exam. Please make sure to work only in this directory.

Inside your `questions/` directory you will find the skeleton codes you need for the programming exercises.

A clean copy of the skeleton codes is available in your `questions_backup` directory.

Documentation In the directory `~/documentation/manuals` you will find reference manuals and the cppreference.com website. To extract the latter one, you can use the following command:
`$ tar zxvf cppreference_html_book_20141118.tar.gz`
the main page is `reference/en/index.html`.

In `~/documentation/lectures` you will find all handouts of the lecture, and in `~/documentation/exercises` the solution sheets of the exercises.

At the end of the exam

At the end of the exam you must **stop working immediately**. Be sure to have saved all open documents before the end.

Grading table

Question	Maximum score	Score	TA 1	TA 2
Question 1	25			
Question 2	25			
Question 3	25			
Question 4	10			
Question 5	15			
Total:	100			

Question 1: Operational Intensity & Roofline Model (25 points)

In this exercise we want to compare the operational intensity for the viscous and the inviscid Burger's equation. The viscous Burger's equation for the scalar field $u(x,t)$ is given by

$$\frac{\partial u(x,t)}{\partial t} + \underbrace{u(x,t)\frac{\partial u(x,t)}{\partial x}}_{\text{advection}} = \underbrace{\nu \frac{\partial^2 u(x,t)}{\partial x^2}}_{\text{diffusion}} . \tag{1}$$

Here, ν denotes a constant diffusion coefficient. If we set $\nu = 0$ we arrive at the inviscid Burger's equation

$$\frac{\partial u(x,t)}{\partial t} + u(x,t)\frac{\partial u(x,t)}{\partial x} = 0 . \tag{2}$$

We aim to numerically solve the two equations on a compact domain $x \in [0, L] \subset \mathbb{R}$ and $t \in [0, T] \subset \mathbb{R}$ for given initial $u(x, 0) = u_0(x)$ and boundary conditions $u(0, t) = f(t)$ and $u(L, t) = g(t)$. We choose to discretize the spatial and temporal domain with finite differences, using N and M points, respectively. More concretely, we discretize time $t_n = n\Delta t$ with $\Delta t = \frac{T}{M-1}$ and $n = 0, \ldots, M-1$. The spatial discretization is done using $x_i = i\Delta x$ with $\Delta x = \frac{L}{N-1}$ and $i = 0, \ldots, N-1$. For the sake of brevity we write $u_i^n = u(x_i, t_n)$.

We discretize eq. (1) using explicit Euler time-stepping, **one-sided first order** finite difference scheme for the advection term*, and a **second order centered** finite difference scheme for the diffusion term.

The numerical scheme for advection is chosen to be consistent with the direction in which the field is advected. It is usually called first-order upwind and reads

$$\begin{aligned} \frac{u_i^{n+1} - u_i^n}{\Delta t} + a\frac{u_i^n - u_{i-1}^n}{\Delta x} = 0 \quad \text{for} \quad a > 0 \\ \frac{u_i^{n+1} - u_i^n}{\Delta t} + a\frac{u_{i+1}^n - u_i^n}{\Delta x} = 0 \quad \text{for} \quad a < 0 \end{aligned} \tag{3}$$

For the described numerical scheme, the formula to update u_i^{n+1} reads:

$$\begin{aligned} u_i^{n+1} &= u_i^n + \frac{\nu\Delta t}{\Delta x^2}(u_{i+1}^n + u_{i-1}^n - 2u_i^n) + u_i^n\frac{\Delta t}{\Delta x}(u_i^n - u_{i-1}^n) \quad \text{for} \quad u_i^n > 0 . \\ u_i^{n+1} &= u_i^n + \frac{\nu\Delta t}{\Delta x^2}(u_{i+1}^n + u_{i-1}^n - 2u_i^n) + u_i^n\frac{\Delta t}{\Delta x}(u_{i+1}^n - u_{i-1}^n) \quad \text{for} \quad u_i^n < 0 . \end{aligned} \tag{4}$$

Note: The variables Δx, Δt and ν in the above update are constant throughout the time integration.

a) Calculate the total number of floating point operations (FLOP) per update steps for (i) $\nu \neq 0$ and (ii) $\nu = 0$. Write down all assumptions you make to derive your result.

We can treat coefficients $\frac{\nu\Delta t}{\Delta x^2}$ and $\frac{\Delta t}{\Delta x}$ as constants that do not have to be recomputed in each iteration. Otherwise that, we assume no additional reordering of the terms in eq. (4). Counting the number of floating points results in (i) $9N$ and (ii) $4N$.

Grading scheme:

- 2p viscous kernel

- 2p inviscid kernel

b) Calculate the total number of memory movements in bytes per update step for the (i) $\nu \neq 0$ and (ii) $\nu = 0$. Assume we have no cache available and write down any other assumption that you make to derive your result. Count read and write operations separatly. Assume double precision.

- No-cache: We assume that we have to load each value $u_{i-1}^n, u_i^n, u_{i+1}^n$ in the stencil. This implies that in both cases, we have to read $3N$ and write N doubles per update. In total, this amounts to $4 \cdot 8N = 32N$ bytes of memory movements per update.

Grading scheme:
- 2p viscous kernel
- 2p inviscid kernel

c) Define the operational intensity in $\left[\frac{FLOP}{Byte}\right]$ of the simulation for the kernels (i) and (ii) based on your results from the previous subquestions.

The resulting operational intensities in $\left[\frac{FLOP}{Byte}\right]$ for the two kernels are

- No-cache: kernel (i) 9/32, kernel (ii) 4/32

Grading scheme:
- 2p viscous kernel
- 2p inviscid kernel

d) You run the software on a computing architecture that has a peak performance of $10\left[\frac{GFLOP}{s}\right]$. Define the three bandwidth $\left[\frac{GB}{s}\right]$ regimes for the following scenarios

(i) both kernels (i) and (ii) are memory bound.
(ii) both kernels (i) and (ii) are compute bound,
(iii) one kernel is compute and the other kernel is memory bound,

The ridge point is computed as $I_r =$ peak performance/bandwidth and thus we find that with

(i) bandwidths below $10\frac{32}{9}\left[\frac{GB}{s}\right]$ both kernels are memory bound,
(ii) bandwidths above $10\frac{32}{4}\left[\frac{GB}{s}\right]$ both kernels are compute bound,
(iii) bandwidths between $10\frac{32}{9}$ and $10\frac{32}{4}\left[\frac{GB}{s}\right]$ kernel (i) is compute bound and kernel (ii) is memory bound,

Grading scheme:
- 2p correct (i)
- 2p correct (ii)
- 2p correct (iii)

e) Draw the roofline model for a computing architecture with a peak performance of $10\left[\frac{GFLOP}{s}\right]$ and a peak bandwidth of $50\left[\frac{GB}{s}\right]$. In the plot mark the ridge point and label the x- and y-axis. Also, indicate where the arithmetic intensity of the two kernels (i) and (ii) intersects the roofline model.

Grading scheme:

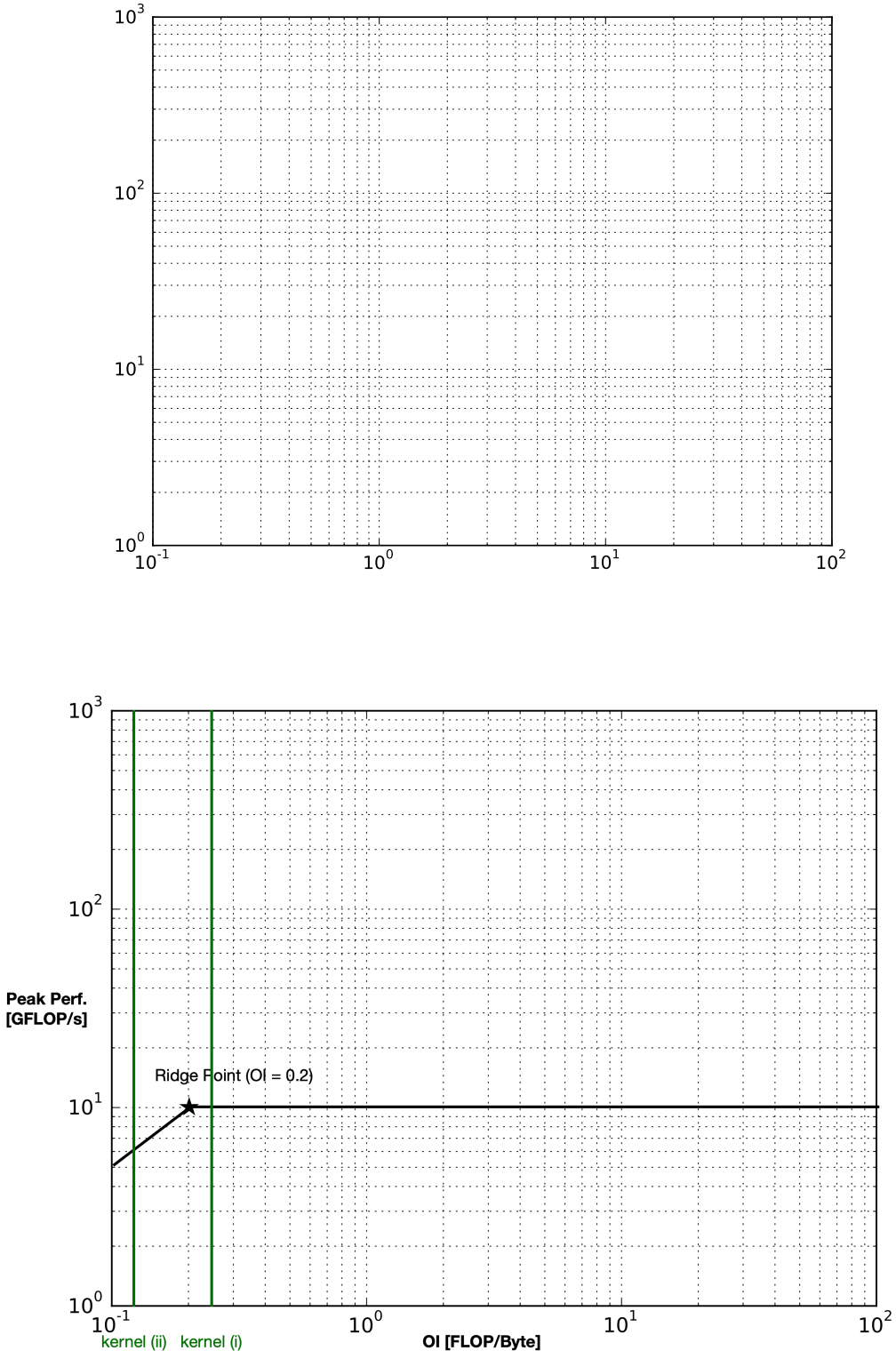

Figure 1: For assumption "No-cache"

- 1p for correct x&y axis
- 2p for correct roofline and ridge point
- 2p viscous kernel (i)
- 2p inviscid kernel (ii)

Question 2: Multiple Choice (25 points)

This question includes general multiple choice questions. You get full points only if you select the correct answer. There is only one correct answer per question.

a) (2 points) What is the correct register size - instruction set correspondences?

- ☐ 256bit register can hold 4 int or 4 floats or 8 doubles
- ☐ 128bit register can hold 4 int or 4 floats or 8 doubles
- ☐ 128bit register can hold 8 int or 8 floats or 4 doubles
- ■ 256bit register can hold 8 int or 8 floats or 4 doubles

b) (7x1 points) Mark if the statement is true or false.

True	False	
■	☐	You can use OpenMP alone to parallelize a C++ application application that fully utilizes the computational power of a computer with multiple CPUs.
☐	■	You can use OpenMP alone to parallelize a C++ application that fully utilizes the computational power of a supercomputer with Nodes.
■	☐	It is not possible to encounter false sharing in a pure MPI application.
☐	■	Matrix-Matrix multiplications routines tend to be memory bound.
☐	■	For sending large messages through MPI the dominant limiting factor will be latency.
☐	■	Blocking communication is preferred to non-blocking since the CPU cannot perform any work before it is completed.
■	☐	Loop unrolling helps to increase instruction level parallelism and decreases loop overhead.

c) (5x2 points) You are given this code below and want to Broadcast this matrix M from rank 3 to all other ranks. Fill in the values MPI_Bcast

```
1 float* A = (float*) calloc(6*5, sizeof(float));
2 // fill A with values;
3
4 MPI_Bcast( data, data_size, datatype, rank, communication);
```

1. What should be `data`?
   - ☐ *A
   - ■ A
   - ☐ A.data()
   - ☐ &A

2. What should be `data_size`?
   - ☐ 5
   - ■ 30
   - ☐ 6
   - ☐ 15

3. What should be `datatype`?
   - ■ MPI_FLOAT

☐ MPI_DOUBLE
☐ MPI_INT
☐ MPI_MATRIX

4. What should be `rank`?
   ☐ 0
   ☐ 1
   ☐ 2
   ■ 3

5. What should be `communication`?
   ☐ MPI_STATUS_IGNORE
   ☐ &status
   ■ MPI_COMM_WORLD

d) (2x3 points) You are given a serial program by your employer which you have to parallelize. You identify that the parallel fraction of the code is 80%. If you deploy an optimized version of the program on your 10 core work station, what is the maximum theoretical speedup that you can achieve on this machine? (Results rounded to 2 significant digits.)

☐ 5.0
☐ 3.3
■ 3.6
☐ 3.0

Unfortunately due to imbalanced work distribution you realize that one core is assigned double the work compared to the other cores. Therefore it has double the runtime compared to the other cores doing the remaining work shared between each other. What is your measured speedup now? (Results rounded to 2 significant digits.)

☐ 1.1
☐ 3.3
■ 2.9
☐ 2.5

Question 3: 2D Diffusion with the ADE scheme and OpenMP(25 points)

Heat flow in a 2D medium can be described by the diffusion equation of the form

$$\frac{\partial \rho(x,y,t)}{\partial t} = D\nabla^2 \rho(x,y,t) \tag{5}$$

where $\rho(x,y,t)$ is a measure for the amount of heat at position $\boldsymbol{r}$ and time t and the diffusion coefficient D is constant. Lets define the domain Ω in two dimensions as $x, y \in [-1,1]$. We will use the boundary condition:

$$\rho(x,y,t) = 0 \quad \forall\, t \geq 0 \,\text{and}\, (x,y) \notin \Omega \tag{6}$$

and an initial distribution:

$$\rho(x,y,0) = \begin{cases} 1 & |x,y| < 1/2 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

In this question the diffusion equation is discretized using a new discretization scheme, called Alternating Direction Implicit (ADI). The skeleton code provides the implementation of a serial version for the diffusion equation with the ADI scheme.

a) Make any necessary changes in the provided `Makefile` such that the code compiles using OpenMP.
   - 3 points for adding `-fopenmp` in the `Makefile` CFLAGS.

b) Parallelize function `initialize_rho` in the provided skeleton code using OpenMP.
   - 3 points for adding *pragma omp parallel for*.

c) Parallelize function `advance` in the provided skeleton code using OpenMP.
   - 3x4 points for adding *pragma omp parallel for* before each of the 4 for loops. Total 12 points.

d) Parallelize function `compute_diagnostics` in the provided skeleton code using OpenMP.
   - 3 points for adding *pragma omp parallel for*.
   - 4 point for adding *reduction(+:heat)*.

Question 4: 1D Diffusion equation with MPI (10 points)

The one-dimensional diffusion equation models the heat flow in a thin rod of length L

$$\frac{\partial u(x,t)}{\partial t} = D\frac{\partial^2 u(x,t)}{\partial x^2},$$

where $u(x,t)$ is the temperature field at position $x \in [0, L]$ of the rod and at time t and D is the heat diffusivity, considered constant for the material used.

In order to solve the diffusion equation numerically, we discretize space using $i = 0, \ldots, N-1$ equidistant grid points $x_i = ih$, where $h = \frac{L}{N-1}$. The time discretization is $t^n = n\Delta t$, where $\Delta t = \frac{1}{2}\frac{h^2}{D}$ (larger values would render the used method unstable).

We use explicit Euler method to advance the temperature field in time and a second-order finite difference scheme for the discretization in space:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = D\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2}$$

and the update of the temperature field in one time step reads:

$$u_i^{n+1} = u_i^{n+1} + \frac{D\Delta t}{h^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n)$$

Write your solution in
`~/questions//codingMPI/main.cpp`.
Use the provided `Makefile` *in the same folder to compile and run your program using* `make`.

Directions to compile, run and plot:

- You can compile the code (serial or parallel version) with `make`.
- You can run the serial version with `make run1`.
- You can run the parallel version with 2 ranks with `make runp`.
- Once the program runs, you can plot the diagnostics with `make plot`.
- Make sure you save the diagnostics plot with the serial and the parallel implementation under different names.

a) Parallelize the code with MPI by filling the parts marked by `TODO` in the function *advance*
The implementation can be found in `solution_codes/codingMPI/diffusion.cpp`. Grading scheme: Inside `advance`:

- 1p: For taking care that at the ends of the domain we do not exchange
- 1p: For taking care that the rank handling the *left boundary* exchanges data only for the *right value* of its sub-domain.
- 1p: For taking care that the rank handling the *right boundary* exchanges data only for the *left cell* of its sub-domain.
- 1p: For exchanging data on both sides (left/ right) of sub-domains that are not at the left/right domain boundaries.
- 1p: For using the correct MPI datatype (`MPI_DOUBLE`).
- 1p: For exchanging only 1 cell (right and left ghost cells).
- 2p: For MPI solutions that do not cause deadlocks.

Total: 8pt

b) For a given time, compute the integral of $u(x,t)$ over the domain (the total sum of the temperature). Fill the missing MPI parts in `compute_diagnostics`, and plot the result as a function of time using $D = 1$, $L = 2$ and $N = 256$. Make sure that the resulting plot of diagnostics (i.e. the integral of temperature over the simulation time) with the MPI implementation and ranks ≥ 2 matches the one produced with the serial version.

Note: there is no need to write anything in the exam sheet. Simply provide the two resulting plots for the serial and the parallel implementation.

The implementation can be found in `solution_codes/codingMPI/diffusion.cpp`. Grading scheme:
Inside `advance`:

- 2p: For correct use of `MPI_Reduce`

Total: 2pt

Question 5: Power Method (15 points)

The power method is an iterative technique for locating the dominant (largest) eigenvalue of a matrix and the associated eigenvector. Consider the symmetric Lehmer matrix $A \in \mathbb{R}^{n \times n}$ with $a_{ij} = \min(i,j)/\max(i,j)$ for all $i, j = 1, \ldots, n$. The power method produces a sequence of column vectors $\boldsymbol{q}^{(k)} \in \mathbb{R}^{n \times 1}$ given by

Algorithm 1 Power Method

for $k = 1, 2, \ldots$ **do**
  $\boldsymbol{z}^{(k)} \leftarrow A\boldsymbol{q}^{(k-1)}$
  $\boldsymbol{q}^{(k)} \leftarrow \boldsymbol{z}^{(k)}/\|\boldsymbol{z}^{(k)}\|_2$
  $\lambda^{(k)} \leftarrow [\boldsymbol{q}^{(k)}]^T A\boldsymbol{q}^{(k)}$
end for

If A is not deficient and its largest eigenvalue is unique, then $\boldsymbol{q}^{(k)}$ converge to an eigenvector with eigenvalue $\lambda^{(k)}$.

a) You are given a serial skeleton code which calculates the dominant eigenvalue of the Lehmer matrix for $n = 8192$ using the power method. It stops at the k-th iteration if the condition $|\lambda^{(k)} - \lambda^{(k-1)}| < 10^{-6}$ is satisfied. It uses $\boldsymbol{q}^{(0)} = [1, 0, 0, \ldots]^T$ for the initial guess and prints the dominant eigenvalue. Parallelize the skeleton code with MPI. The strategy you are asked to implement is based on splitting the rows of A among the available processors. Assume that n is divisible by the total number of processors. You do not need to split $\boldsymbol{q}^{(k)}$ among processes. *Hint:* you will need to use `MPI_Allgather` and `MPI_Allreduce`.

By splitting the work among the rows of A the serial algorithm must be extended with an `MPI_Allgather` in order to compute the inner products in the subdomain as well as an `MPI_Allreduce` to reduce the eigenvalue among the processes. The parallel algorithm is as follows

```
_gemv(myN, N, A, q0, q1);
while (true)
{
    MPI_Allgather(q1, myN, MPI_DOUBLE, q0, myN, MPI_DOUBLE, MPI_COMM_WORLD);
    _norm(N, q0);
    _gemv(myN, N, A, q0, q1);
    lambda1 = 0.0;
    double mylambda = 0.0;
    for (int i = 0; i < myN; ++i)
        mylambda += q0[off+i] * q1[i];
    MPI_Allreduce(&mylambda, &lambda1, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
    ++iter;
    if (std::abs(lambda1 - lambda0) < tol)
        break;
    lambda0 = lambda1;
}
```

where the `std::swap` is no longer required due to the gather operation. Note that `_gemv` and `_norm` must not be modified, except for the reduced number of rows `myN` of the matrix-vector multiplications. All ranks initialize the full vector `q0` in order to save one

`MPI_Allgather` call at the beginning. Finally, the eigenvalue must be reduced among the processes. Note the computation of `mylambda` involves only `myN` elements, where `off` is the row offset of the process.

- 1pt: for initializing and finalizing MPI
- 3pts: for correct MPI gather operation
- 3pts: for correct MPI reduce operation
- 3pts: for correct offsets when distributing the matrix to MPI processes

b) How do you expect the parallel power method to scale (in terms of strong scaling)? Is the problem compute bound or memory bound? Explain your reasoning.

The dominant component of the algorithm is matrix-vector multiplication (GEMV). The operational intensity of this kernel is $\mathcal{O}(1)$ and the kernel is expected memory bound. The memory bandwidth utilization is high and the algorithm is not well parallelizable due to inner products with low data reusage (as opposed to matrix-matrix multiplication). The scaling is expected poor, especially for small matrix sizes.

- 3pts: for identifying the algorithm is memory bound

c) Compute the strong scaling speedup for 2 and 4 processors using $n = 8192$.

The scaling has been performed with an Intel Xeon E5 2680v3 node on Euler. Options used are `-map-by ppr:core:1` and `-bind-to none`. The obtained strong speedup for two problem sizes is tabulated below.

Number of ranks	$n = 8192$	$n = 16384$
2	1.64	1.75
4	1.84	2.74
8	1.63	3.20
16	1.05	2.82

- 2pts: for strong scaling with at least 3 different values of processors