ETH zürich

High Performance Computing for Science and Engineering II

P. Koumoutsakos., S. Martin
ETH Zentrum, HG G3
CH-8092 Zürich

2019

Practice Exam

Issued: May 20, 2019, 10:15
Hand in: May 20, 2019, 12:00

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: Bayesian Inference (15 points)

Let X be a random variable that depends on a parameter ϑ. The conditional probability density function is given by

$$p(x \mid \vartheta) = \begin{cases} \frac{2x}{\vartheta^2}, & 0 < x < \vartheta \\ 0, & \text{otherwise}\,. \end{cases}$$

Assume that you have observed one realization of the random variable $X = x_1$.

a) First, verify that $p(x \mid \vartheta)$ is a proper density function.

b) Write the likelihood function $p(x_1 \mid \vartheta)$ as a function of ϑ. You may want to use indicator functions: $\chi_{[a,b]}(x)$ is equal to 1 if $x \in [a, b]$ and 0 otherwise.

c) Find the posterior distribution $p(\vartheta \mid x_1)$ for a uniform prior in $[0, 1]$.

d) Estimate the uncertainty in the random variable X after observing x_1 by calculating the conditional distribution of X on x_1.

Question 2: Applied Optimization & UQ (20 points)

Rumours spread that you are amongst the most notorious programmers and statisticians in the region of Appenzell. The hedge fund *Appenance LTD* approaches you and wants you to have a look at their price prediction model $\mathcal{M}_1$ for cattle trading. They say that their strategy is presumably predicting wrong confidence intervals and hence is heavily underperforming. This particular cattle breed is traded over the counter only during weekends at the local markets in *Herisau* and for that reason trades data $d = \{d_i\}_{i=1}^N$ is very limited and difficult to collect.

Without blinking an eye, you accept their offer and you search for the parameters $\hat{\theta} = (\hat{x_0}, \hat{\sigma}^2)$ that maximize the likelihood of the model, i.e.,

$$\hat{\theta} = \arg\max_{\theta \in \Theta} \mathcal{L}(\theta; d).$$

For obvious reasons the model can not be disclosed here, but we can tell that x_0 is a computational parameter and σ^2 describes the variance of the model error.

You use Korali's optimizer (CMA-ES) to find the parameters that maximize the loglikelihood. After 4 consecutive runs you observe the following outputs on the screen:

```
[Korali] CMA-ES Finished
[Korali] Optimum LogLikelihood (Maximize) found: -8.49194e-01
[Korali] Optimum LogLikelihood (Maximize) found at:
         X0 = +1.503e+00
         Sigma2 = +3.501e+00
[Korali] Stopping Criterium: Function value differences < (1.00e-12)
```

Listing 1: Run 1

```
[Korali] CMA-ES Finished
[Korali] Optimum LogLikelihood (Maximize) found: -8.49193e-01
[Korali] Optimum LogLikelihood (Maximize) found at:
         X0 = +3.004e+00
         Sigma2 = +0.998e+00
[Korali] Stopping Criterium: Function value differences < (1.00e-12)
```

Listing 2: Run 2

```
[Korali] CMA-ES Finished
[Korali] Optimum LogLikelihood (Maximize) found: -8.49192e-01
[Korali] Optimum LogLikelihood (Maximize) found at:
         X0 = +3.002e+00
         Sigma2 = +0.999e+00
[Korali] Stopping Criterium: Function value differences < (1.00e-12)
```

Listing 3: Run 3

```
[Korali] CMA-ES Finished
[Korali] Optimum LogLikelihood (Maximize) found: -8.49193e-01
[Korali] Optimum LogLikelihood (Maximize) found at:
          X0 = +1.503e+00
          Sigma2 = +3.500e+00
[Korali] Stopping Criterium: Function value differences < (1.00e-12)
```

Listing 4: Run 4

a) You observe that CMA-ES converges to different optima although the parameter of CMA-ES are kept the same in each run (except for the seed). Answer the following questions:

- Provide an explanation of the fact CMA-ES returns different results given the same objective function and configuration.
- What can you conclude about the problem regarding these results?
- Why does this pose a problem in predicting the price of cattle?

b) Assume that we know that the likelihood function is a multivariate gaussian mixture with local covariance Σ, i.e

$$\mathcal{L}(\theta; d) = \sum_{i=1}^{2} \omega_i \mathcal{N}(\theta|\mu_i, \Sigma),$$

with $\omega_1 + \omega_2 = 1$ and

$$\Sigma = \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix}.$$

In order to further analyze the model at hand, you want to sample the posterior distribution of the parameters $\theta = (x_0, \sigma^2)$ given the data d,

$$p(\theta|d) = \frac{p(d|\theta)p(\theta)}{p(d)}.$$

Since we know nothing about the parameters we choose a non informative prior distribution $p(\theta)$. What output do you expect from your sampling algorithm? Please provide 3 qualitative sketches and answer the questions:

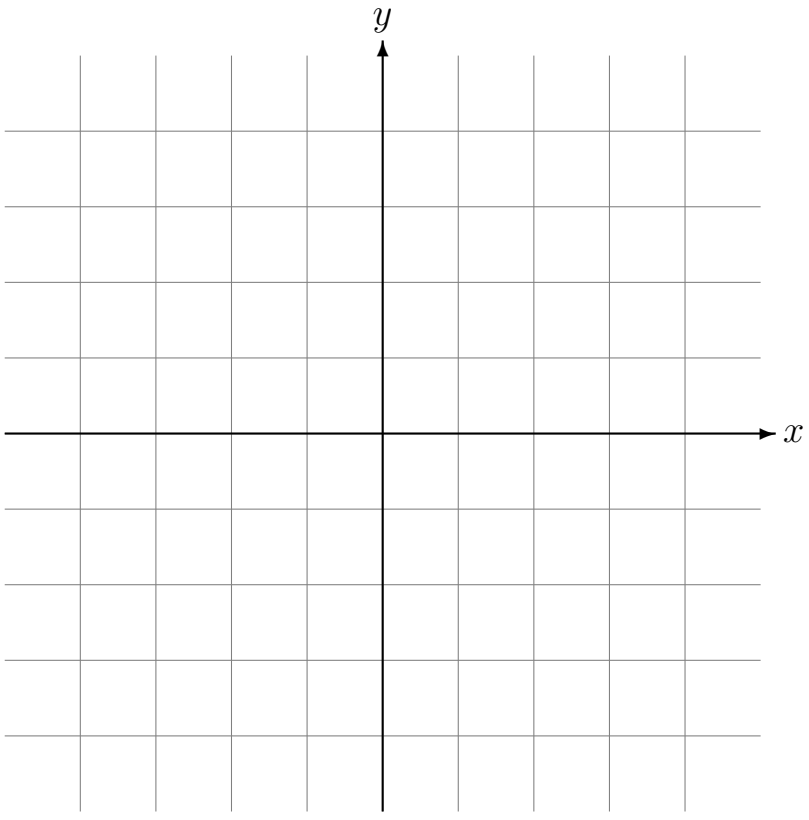
- Define a prior distribution $p(\theta)$ for x_0 and σ^2.
- Specify the weights ω_i considering the output of CMA-ES from the previous subquestion.

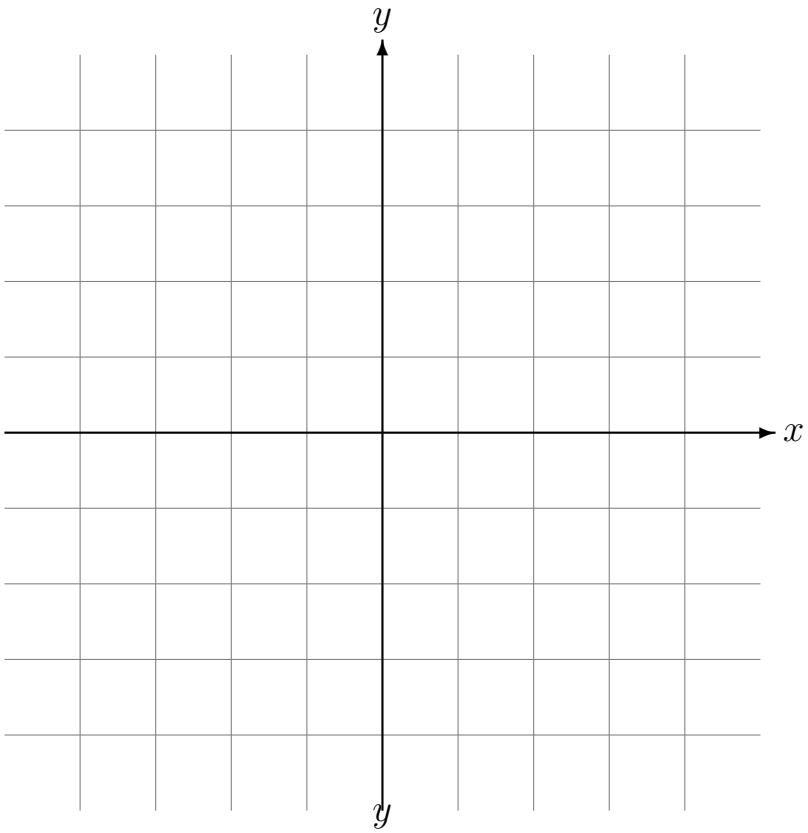
- Sketch the sampling, i.e. draw a possible outcome of the samples in a scatter plot x_0 vs. σ^2. Include some contour lines of $p(\theta|d)$ if it supports the interpretability of your figure. Hint: the eigenvalue decomposition of Σ is given by

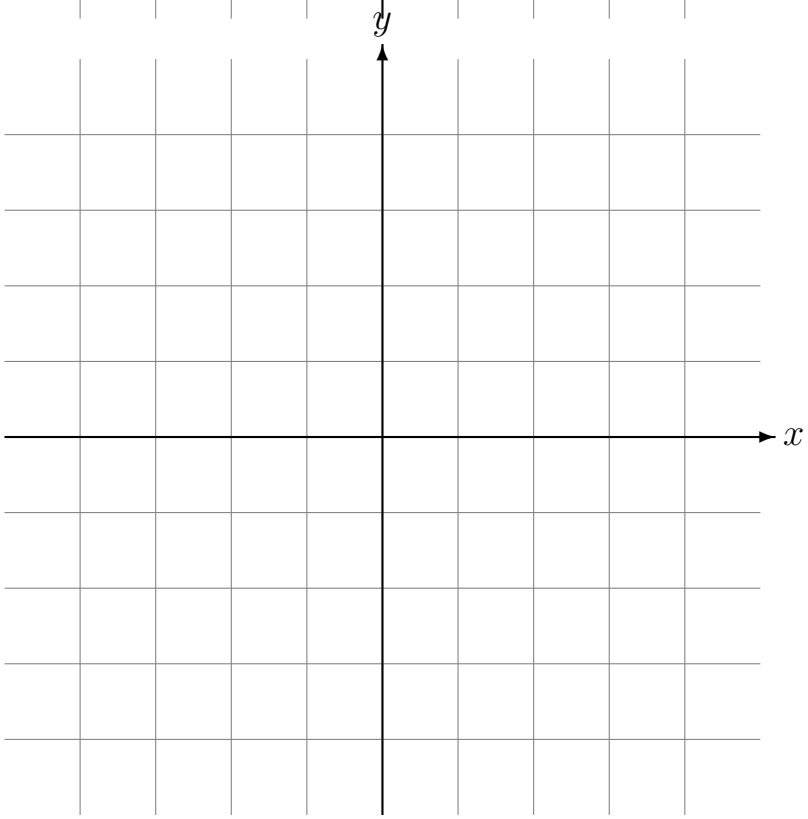
$$\Sigma = \frac{1}{\sqrt{2}} \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0.5 & 0 \\ 0 & 1.5 \end{bmatrix} \cdot \frac{1}{\sqrt{2}} \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}^T .$$

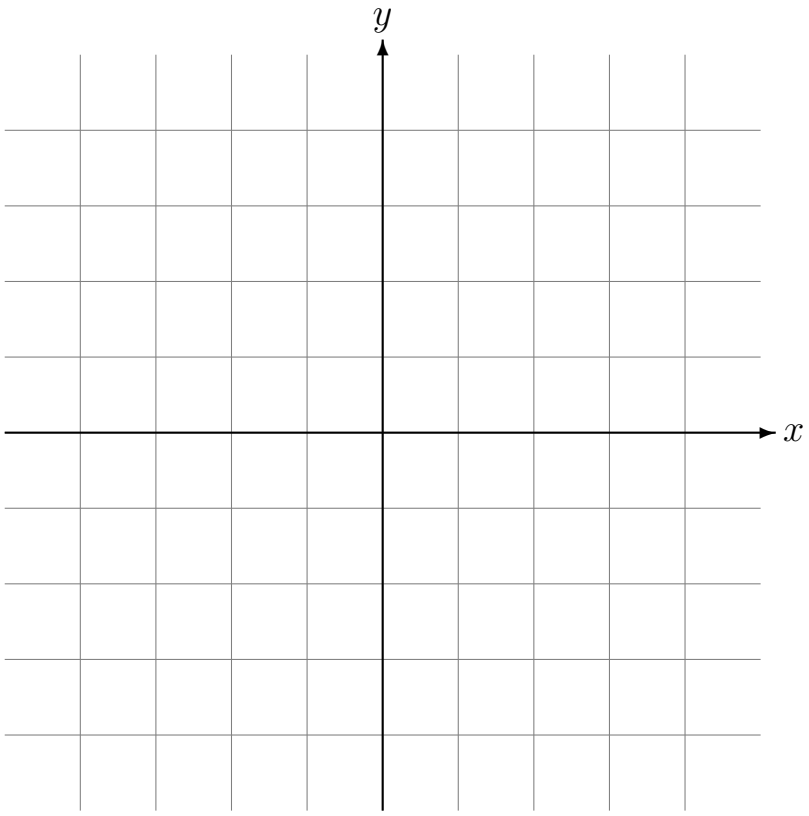
- Sketch the marginal distribution $p(\sigma^2|d) = \int_{-\infty}^{+\infty} p(\theta|d)\,\mathrm{d}x_0$.

- Sketch the marginal distribution $p(x_0|d) = \int_{-\infty}^{+\infty} p(\theta|d)\,\mathrm{d}\sigma^2$.




y
x


y
x



Note: 3 sketches are required, 1 graph is spare. Don't forget to label the axes, and add ticks to the axes where possible (vertically and horizontally) in order to localize your graphs. If you make assumptions, please document it. Please make sure that your plots are consistent throughout.

c) In the meantime the research department of *Appenance LTD* came up with a more sophisticated pricing model $\mathcal{M}_2$, taking into account the PH-value of the grass the cows eat and the amount of days they graze in rain. You discuss this model with your friend Reto Küehli who is a cow specialist and he assures that these parameter have no influence on the development of the cows. Nevertheless the researchers find that the maximal likelihood for the new model $\mathcal{M}_2$ is larger than the maximal likelihood for $\mathcal{M}_1$ (on data set d). Answer the following questions in 3-5 sentences each:

- How can you explain that model $\mathcal{M}_2$ achieves a "better" likelihood although it includes irrelevant factors.
- What other measures do you know from this course that allow you to select between $\mathcal{M}_1$ and $\mathcal{M}_2$? Also argue why comparing the likelihoods of $\mathcal{M}_1$ and $\mathcal{M}_2$ is not best practice and why you would choose a different measure.

Question 3: Parallel Tasking Theory (20 points)

You work at a newly instituted supercomputing center and every morning you receive a queue of 12 jobs that need to be executed. Each job can be executed **independently** and cannot be parallelized. The computational cost of each job is given in the following in terms of a reference computational load C, but you do not know this load a-priori to design an optimal parallel tasking strategy. **Communication costs are ignored in this exercise.**

Job order:	1	2	3	4	5	6	7	8	9	10	11	12
Comp. load:	12	1	1	12	1	1	1	1	1	1	1	1

We assume for simplicity that this load can be directly translated to a fixed reference computational time, e.g. that a job of load $C = 7$ can be executed by a single rank in $T = 7$ time units. Time measurements in the following need to be provided in terms of these time units. You are provided with a small compute node with **4 cores**, each capable of running a single job at a time, and you are asked to propose algorithms taking advantage of various parallel tasking strategies.

a) In the classical divide-and-conquer method the tasks are divided to the available processors. You cannot affect the ordering of the jobs. This leads to the following partition:

Job order:	1	2	3	4	5	6	7	8	9	10	11	12
Comp. load:	12	1	1	12	1	1	1	1	1	1	1	1
Exec. by rank:	1	1	1	2	2	2	3	3	3	4	4	4

Compute:

1. the total runtime T_{total}
2. the average rank run-time T_{avg}
3. the load imbalance ratio I
4. the average rank idle time W_{avg}.

b) Now you are asked to apply the producer-consumer strategy. Note that you cannot affect the ordering of the jobs assigned to the ranks. Which rank will execute each job? Please fill in the following array and answer the questions that follow:

Job order:	1	2	3	4	5	6	7	8	9	10	11	12
Comp. load:	12	1	1	12	1	1	1	1	1	1	1	1
Exec. by rank:												

1. What is the total runtime T_{total} in this case?
2. What is the average rank run-time T_{avg}?
3. What is the load imbalance ratio I?
4. What is the average rank idle time W_{avg}?
5. What is the disadvantage of the producer-consumer strategy in general? Is the extra effort to implement this strategy worth it in this case?

c) The problem with the jobs is that the computational load is not balanced and we do not know the load a-priori to design an optimal tasking strategy. A different ordering of **the same** jobs might cause the **worst case** scenario (in terms of run-time) for a **divide-and-conquer** tasking strategy.

1. What is an **example** of an ordering that causes this worst case scenario? To which rank is each job assigned in your example?
2. Assuming that the jobs are assigned a random uniform ordering by an external client, what is the probability of a worst case scenario ordering occuring?
3. How much faster is the producer-consumer strategy in this case? Report the speed-up compared to the divide-and-conquer method.

Question 4: Parallelizing the Quadratic Form (20 points)

In this exercise, you are asked to parallelize a quadratic form with UPC++. The quadratic form is given by

$$Q = \boldsymbol{v}^T \cdot A \cdot \boldsymbol{w} = \sum_{ij} v_i A_{ij} w_j \tag{1}$$

where the matrix $A \in \mathbb{R}^{n \times n}$ and $v, w \in \mathbb{R}^n$. The entries of A, w, and v are given by:

$$A_{ij} = \frac{i + 2j}{n}, \quad v_i = 1 + \frac{2}{i + 0.5}, \quad w_i = 1 - \frac{i}{3} \tag{2}$$

In the following you are provided with the serial program.

```
#include <stdio.h>
#include <vector>
#include "matrix.hpp"

int main(int argc, char** argv) {

  int n = 1024;

  hpcse::matrix<double, hpcse::row_major> A(n,n);
  std::vector<double> v(n), w(n);

  for (int i=0; i<n; ++i)
      for (int j=0; j<n; ++j)
          A(i,j) = (i + 2.*j) / n / n;

  for (int i=0; i<n; ++i)
      v[i] = 1. + 2. / (i + 0.5);

  for (int i=0; i<n; ++i)
      w[i] = 1. - i / 3. / n;

  double result = 0.;
  for (int i=0; i<n; ++i)
      for (int j=0; j<n; ++j)
          result += v[i] * A(i,j) * w[j];

  if(rankId==0)
    printf("Rank 0 - Total result %lf.\n", result);

  upcxx::finalize();
}
```

a) Fill in the 20 missing **gaps** after the TODO: comments in the following parallel implementation.

```
#include <stdio.h>
#include <vector>
#include "matrix.hpp"
#include <upcxx/upcxx.hpp>

int main(int argc, char** argv) {
  upcxx::init();
  int rankId = upcxx::rank_me(), rankCount = upcxx::rank_n();
  int n = 1024, chunk = n / rankCount;

  // TODO: Initialize a global pointer with name globalResults
  ____________________________ globalResults;
  if(rankId==0) globalResults = ___________________<double>(
      ____________________);
  // TODO: Broadcast the adress of the global pointer to all ranks
  ________________________(_________________________, 1, 0).wait();


  hpcse::matrix<double, hpcse::row_major> local_A(chunk,n);
  // TODO: Define local_v and w as vectors of doubles of the correct
      size
  std::vector<double> local_v(_____________), w(_____________);


  for (int i=0; i<chunk; ++i)
      for (int j=0; j<n; ++j)
          local_A(i,j) = ((rankId*chunk + i) + 2.*j) / n / n;

  for (int i=0; i<chunk; ++i)
       // TODO: Fill in the entries of the local v vector
      local_v[i]=1.+2./( (_________*_________+ i ) +_______);


  for (int i=0; i<n; ++i) w[i]=1.-i/3./n;

  double local_result = 0.;
  for (int i=0; i<chunk; ++i)
      for (int j=0; j<n; ++j)
          // TODO: Add the result to the local result
          __________ += __________ * local_A(i,j) * __________;


  // TODO: Update the globalResults pointer with the local result
 _____________(_____________,globalResults + _____________, 1).wait();
  upcxx::barrier();

  if (rankId == 0){
    double global_result = 0.;
    // TODO: Enable access of the global results pointer locally
    double* _____________ = globalResults._____________;
    for (int i = 0; i < rankCount; i++){
```

```
50          // TODO: Add each rank result to the global result
51          _________________ += _________________;
52        }
53        printf("Rank 0 - Total result %lf.\n", global_result);
54      }
55      upcxx::finalize();
56    }
```

b) What is the parallel tasking strategy used to parallelize the quadratic form?

c) Comment on the load balance of each rank. Do you think the strategy employed is reasonable, or another strategy might be much more efficient for this task?

d) The parallel implementation contains a barrier in line 43. Do you think this barrier is needed? Why?

Question 5: GEMV on GPUs (20 points)

The code below computes a matrix-vector product (GEMV) using an $N \times N$ matrix

$$\boldsymbol{a} = M\boldsymbol{b}\,, \quad \text{where} \quad a_i = \sum_{j=0}^{N-1} M_{i,j} b_j\,. \tag{3}$$

```
1
2  #include <vector>
3
4  #define VECSIZE 512
5
6  void random_init(std::vector<float> & v, std::vector<float> & m)
7  {
8      // Some random initialization - this could be anything
9      for(std::size_t i = 0; i < v.size(); ++i)
10         v[i] = i;
11     for(std::size_t i = 0; i < m.size(); ++i)
12         m[i] = i;
13 }
14
15 void gemv_cpu(float * a, float * m, float * b, unsigned int N)
16 {
17     for(unsigned int i = 0; i < N; ++i)
18         for(unsigned int j = 0; j < N; ++j)
19             a[i] += m[N*j+i] * b[j];
20 }
21
22 int main()
23 {
24     unsigned int const N = VECSIZE;
25     std::vector<float> a(N);
26     std::vector<float> b(N);
27     std::vector<float> matrix(N*N);
28
29     random_init(b, matrix);
30
31     //
32     // TODO replace gemv_cpu.
33     // The code should perform the matrix-vector multiplication
34     // on the GPU and store the result in the CPU vector 'a'.
35     //
36     gemv_cpu(&a[0], &matrix[0], &b[0], N);
37
38     return 0;
39 }
```

a) Modify the host code such that the matrix-vector product is computed on the GPU. The final result should be available on the CPU.
Write and call a CUDA kernel (`gemv_gpu`) for square matrices of arbitrary dimension N.

Notes:

- Show how the kernel is called in the `main()` function.
- Checking CUDA for errors is not mandatory.

Question 6: Advanced MPI: Communicators (30 points)

The $N \times N$ Lehmer matrix A is defined as $A_{i,j} = \min(i,j)/\max(i,j)$, where $i, j = 1, 2, \ldots, N$. In this task you will compute the L_∞ norm

$$L_\infty = \max_i \left\{ \sum_j |A_{ij}| \right\} \tag{4}$$

of this matrix distributed across a **two dimensional** grid of MPI processes.

a) In the first part of this task you have to complete the following code snippet to initialize the matrix. You have 16 available MPI processes, and should divide the matrix in a 4×4 grid. You can assume that the size of the matrix N is always divisible by 4. Modify the code between `TODO_START` and `TODO_END` in order to initialize the matrix using an MPI cartesian grid topology.

```
int size, rank, size_x, size_y;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

// TODO_START
// assume that we have size = 16 ranks in total
// then size_y = size_x = 4
size_x = 4;
size_y = size/size_x;
assert(N % size_x == 0 && N % size_y == 0);

// local matrix size
const int Nx_loc = N / size_x;
const int Ny_loc = N / size_y;

const int chunk_size = Nx_loc*Ny_loc;

// compute coordinates (i,j) of rank
int coord_x = rank/size_x;
int coord_y = rank % size_x;
// TODO_END

 double* matrix_loc = new double[chunk_size];

// initialize local matrix
for(int i = 0; i < Ny_loc; ++i)
    for(int j = 0; j < Nx_loc; ++j)
    {
        // compute global indices
        int i_glo = coord_y*Ny_loc + i;
        int j_glo = coord_x*Nx_loc + j;
        matrix_loc[i*Nx_loc + j] =
```

```
34              double(std::min(i_glo+1,j_glo+1)) / std::max(i_glo+1,j_glo
                    +1);
35      }
```

b) A code snippet used for the computation of the infinity norm L_∞ is provided below. Complete the code between the two `TODO / END_TODO` "brackets" in order to perform the reductions using row and column communicators.

```
1  // compute L_inf = max_col (sum_row (a_ij))
2
3  double sums_loc[Ny_loc]; // local sum
4
5  for(int i = 0; i < Ny_loc; ++i){
6      double sum = 0.0;
7      for(int j = 0; j < Nx_loc; ++j)
8          sum += std::abs( matrix_loc[i*Nx_loc + j] );
9      sums_loc[i] = sum;
10 }
11
12 double sums_row[Ny_loc]; // overall sums of rows
13
14 // TODO: create row communicator and perform sum reduction
15
16 // END_TODO
17
18 double max_strip = 0.0; // local max of every sums_row
19 for(int i = 0; i < Ny_loc; ++i)
20     max_strip = std::max(max_strip,sums_row[i]);
21
22 double L_inf = 0.0; // overall max
23
24 // TODO: column communicator and max reduction with it
25
26 // END_TODO
27
28 if(rank == 0)
29     std::cout << "L_inf norm: " << L_inf << std::endl;
```