High Performance Computing for Science and Engineering II

P. Hadjidoukas, C. Papadimitriou
ETH Zentrum, CTL E 13
CH-8092 Zürich

Spring semester 2017

Exam

Issued: May 29, 2017, 13:00
Hand in: May 29, 2017, 16:00

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:

Grading table

Question	Maximum score	Score	TA 1	TA 2
Question 1	40			
Question 2	5			
Question 3	20			
Question 4	25			
Question 5	15			
Question 6	10			
Question 7	10			
Question 8	10			
Question 9	15			
Question 10	10			
Question 11	20			
Question 12	25			
Total:	205			

A perfect score (6) is 180 points. (out of 205 available).

Question 1: Posterior Distribution Derivation (40 points)

Let $\mathbf{y} = (y_1, \dots, y_N)$ with $y_i \in \mathbb{R}$, be N independent observations obtained at positions $x_i \in \mathbb{R}$. Assume that we want to model each observation with a linear function, $f(x; \vartheta) = \vartheta x$ for $\vartheta \in \mathbb{R}$. Moreover, assume that the observations are linked with the model through the following relation,

$$y = f(x; \vartheta) + \zeta\,, \tag{1}$$

where ζ is a normal random variable with zero mean and variance σ^2.

We want to update our knowledge on the parameters ϑ after obtaining the N observations using Bayes theorem with an improper prior[1], i.e. $p(\vartheta) \propto 1$.

a) Identify the posterior distribution of ϑ conditioned on the set of observations $\mathbf{y} = y_1$, i.e., in the presence of only one observation.

b) Let $\mathbf{A} \in \mathbb{R}^{N \times M}$, $\mathbf{y} \in \mathbb{R}^N$, $\boldsymbol{\vartheta} \in \mathbb{R}^M$, and $\hat{\boldsymbol{\vartheta}} := (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top \mathbf{y}$.
Verify that the following holds:

$$(\mathbf{y} - \mathbf{A}\boldsymbol{\vartheta})^\top (\mathbf{y} - \mathbf{A}\boldsymbol{\vartheta}) = (\boldsymbol{\vartheta} - \hat{\boldsymbol{\vartheta}})^\top \mathbf{A}^\top \mathbf{A} (\boldsymbol{\vartheta} - \hat{\boldsymbol{\vartheta}}) + \text{const.}\,, \tag{2}$$

where terms in const. do not depend on $\hat{\boldsymbol{\vartheta}}$ or $\boldsymbol{\vartheta}$.

c) Use the result of the previous question to show that the posterior distribution of $\vartheta \in \mathbb{R}$ given the linear model in (1) in the presence of N observations is given by,

$$p(\,\vartheta \,|\, \mathbf{y}\,) = \mathcal{N}(\,\vartheta \,|\, \hat{\vartheta},\, \sigma^2 (\mathbf{x}^\top \mathbf{x})^{-1}\,)\,, \tag{3}$$

for $\mathbf{x} = (x_1, \dots, x_N)^\top$.

d) Show that in the limit $N \to \infty$ the posterior distribution converges to a Dirac distribution supported on $\hat{\vartheta}$.

[1] Improper is a prior distribution that does not integrate to one. Here is the uniform distribution over all real numbers.

Question 2: Metropolis-Hastings (5 points)

We want to sample a distribution with a probability density function $p(x)$, $x \in \mathbb{R}$ using the Metropolis-Hastings algorithm. The distribution is shown in Fig. 1. Assuming that we start at $x_0 = 4$, give a description of points x_1 which should be necessarily accepted at the next step if the algorithm proposes them. Explain your answer.

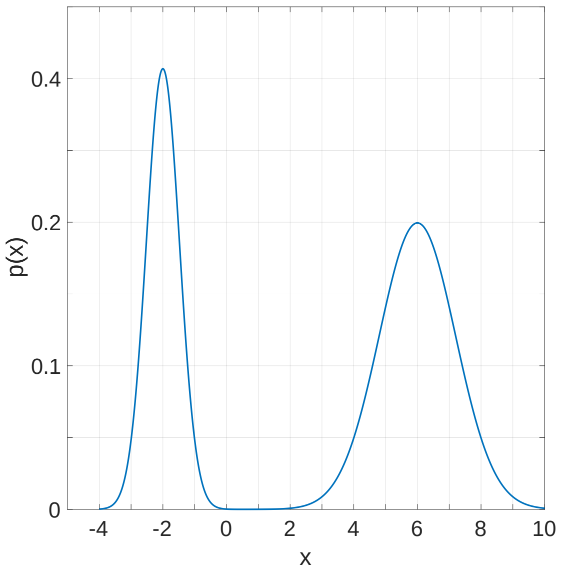


Figure 1: Probability density function $p(x)$.

Question 3: Non-linear model (20 points)

Consider the mathematical model of a physical process represented by the equation,

$$y = \vartheta^2 + \zeta, \tag{4}$$

where $\vartheta \in \mathbb{R}$ is the uncertain parameter of the model, $y \in \mathbb{R}$ is the output quantity of interest, and ζ represents the model error which is quantified by a Gaussian distribution $\zeta \sim \mathcal{N}(0, \sigma_e^2)$, where σ_e^2 is known. The prior uncertainty in ϑ is quantified by a Gaussian distribution with mean μ_π and standard deviation σ_π. Given N independent measurements $y_1, \ldots, y_N$, show, using Bayesian central limit theorem, that the posterior uncertainty in ϑ is given by,

$$\hat{\sigma} = \sqrt{\frac{\hat{\vartheta}\sigma_\pi^2\sigma_e^2}{\mu_\pi\sigma_e^2 + 4N\hat{\vartheta}^3\sigma_\pi^2}},$$

where $\hat{\vartheta}$ is the most probable value of the posterior distribution.

Question 4: Optimal Experimental Design (25 points)

Consider a linear model $f(\boldsymbol{\vartheta}) = \boldsymbol{\Phi}\boldsymbol{\vartheta}$, where $\boldsymbol{\Phi} \in \mathbb{R}^{N \times M}$, $\boldsymbol{\vartheta} \in \mathbb{R}^{M}$, N are the number of states that the model can predict, M is the number of model parameters, $f_i(\boldsymbol{\vartheta}) = \boldsymbol{\Phi}_i \cdot \boldsymbol{\vartheta}$ is the prediction of the model at the state i, and $\boldsymbol{\Phi}_i \in \mathbb{R}^{1 \times M}$ is the i-th row of $\boldsymbol{\Phi}$. Let the prediction error $e_i \in \mathbb{R}$ between the data $y_i \in \mathbb{R}$ collected at state i and the model prediction f_i at state i satisfy the prediction error equation

$$y_i = \boldsymbol{\Phi_i} \cdot \boldsymbol{\vartheta} + e_i \,. \tag{5}$$

Consider the case of $M = 1$. It is assumed that the prediction error follows a normal distribution $e_i \sim \mathcal{N}(0, a_i^2\sigma^2)$ with mean zero and variance $a_i^2\sigma^2$, where (i) $a_i = 1$ for the case of measurement error and negligible model error, and (ii) $a_i = \phi_i$ for the case of model error and negligible measurement error. Implicitly it is assumed in case (i) that the sensor error is constant, independent on the value of the state. In case (ii), it is assumed that this error is a fraction of the value of the measured quantity. Assume a Gaussian prior for the model parameter ϑ with mean μ_π and standard deviation σ_π.

a) Show that mutual information for the case of a single sensor, measuring at state i, is

$$I(\vartheta; y|d_i) = \frac{1}{2}\log|\phi_i^2 a_i^{-2}\sigma^{-2} + \sigma_\pi^{-2}| + c\,, \tag{6}$$

where d_i denotes the location of the sensor at state i.

b) Find the optimal location of a single sensor (the best state to measure given that one sensor is available). Consider two cases: (i) $a_i = 1$, and (ii) $a_i = \phi_i$. ~~Give a physical justification of your findings in each case.~~

c) Show that mutual information for the case of two sensors, measuring at states i and j, is

$$I(\vartheta; y|d_i, d_j) = \frac{1}{2}\log|\phi_i^2 a_i^{-2}\sigma^{-2} + \phi_j^2 a_j^{-2}\sigma^{-2} + \sigma_\pi^{-2}| + c\,, \tag{7}$$

where d_i and d_j denotes the locations of the two sensors at states i and j.

d) Find the optimal locations of two sensors (the best states to measure given that two sensors are available) for the case (i) $a_i = 1$.

Hint: Based on the theory, the mutual information for linear models of the form $f(\vartheta) = A\boldsymbol{\vartheta}$, where $A \in \mathbb{R}^{N\times M}$ and $\boldsymbol{\vartheta} \in \mathbb{R}^M$ with Gaussian prior is given by

$$I(\vartheta; y|d) = \frac{1}{2}\log|A^\top \Sigma_e^{-1} A + \Sigma_\pi^{-1}| + c\,, \tag{8}$$

where Σ_e is the covariance of the prediction error, Σ_π the covariance matrix of the prior, and c is a constant.

Question 5: Parallel Speedup (15 points)

Assume you implemented a serial Monte Carlo application that computes N samples, and you want to parallelize it with MPI in the following way:

- One MPI process reads the data and distributes to the other MPI processes.
- Each of the n_p MPI processes starts an independent Markov-Chain, performs some initialization (burn-in), and then computes N/n_p samples.
- At the end the results are collected at the root process.
- All the communication is performed using `MPI_Bcast` and `MPI_Reduce` only.

The profile of the serial application gives the following results:

- Reading the data from the input files takes T_{setup}.
- The burn-in phase takes $T_{\text{burn-in}}$.
- Each sample takes T_{sample}.

a) Estimate the time needed to run the serial application.

b) How do MPI collectives `MPI_Bcast` and `MPI_Reduce` scale with the number of MPI processes n_p?

c) Estimate the time needed to run the parallel application. Assume that any point to point communication takes T_{comm} time. Identify which is the serial part, the parallel part and the communication part.

d) Estimate the **weak scaling** of the application. Sketch the curve and determine the asymptotic weak scaling.

e) Estimate the **strong scaling** of the application. Note that for $n_p > N$, $n_p - N$ processes will be idle. Sketch the curve and determine the asymptotic scaling.

Question 6: OpenMP Tasks (10 points)

The following serial code checks if the numbers contained in the input vector are prime numbers. The input vector `numbers` has been filled with uniformly distributed random numbers between `lower_limit` and `upper_limit`.

```cpp
template <class integer_type>
bool is_prime(integer_type i)
{
    if (i==1) return false;
    if (i==2) return true;
    if (i%2 == 0) return false;

    integer_type limit = static_cast<integer_type>(std::sqrt(i));
    for(integer_type j=3; j<limit; j+=2)
        if (i%j == 0) return false;

    return true;
}

template <class integer_type>
std::vector<int> is_prime_serial(std::vector<integer_type>& numbers)
{
    size_t n = numbers.size();
    std::vector<int> res(n, true);

    for(size_t i = 0; i < n; ++i)
        res[i] = is_prime(val[i]);

    return res;
}

template <class integer_type>
std::vector<int> is_prime_parallel(std::vector<integer_type>& numbers)
{
    // --- TODO ---
}
```

a) Implement the function `is_prime_parallel`. This function should behave as `is_prime_serial` and be parallelized with OpenMP tasks.

b) Explain why parallelizing this code with static OpenMP loop scheduling is not a good choice.

Question 7: OpenMP Taskloop (10 points)

The following code snippet includes two nested loops that cannot be collapsed and have been parallelized with OpenMP.

```
// A: matrix of size NxN

// nested parallelism is enabled
#pragma omp parallel for
for (int i = 0;  i < N; i++)
{
    // code here

    // TODO: use OpenMP tasks for this loop
    int chunksize = 10;
#pragma omp parallel for schedule(dynamic,chunksize)
    for (int j = 0; j < N; j++)
    {
        A[i][j] = func(i, j);
    }

    // code here
}
```

a) Provide an equivalent parallel implementation of the above code using OpenMP tasks for the innermost loop.

b) Discuss which parallelization approach (original or task-based one) is more efficient and explain why.

Question 8: MPI Datatypes (10 points)

In the following MPI code, we want to send some elements of a square matrix from rank 0 to rank 1.

```
if (rank == 0)
{
   double A[N][N];
   // initialization of A

   // TODO: send the anti-diagonal elements of matrix A to rank 1
}
if (rank == 1)
{
   double buffer[N];

   // TODO: receive elements from rank 0

   // buffer can be used here
}
```

a) Build and use an appropriate MPI datatype to send the anti-diagonal elements of matrix A.

b) What modifications to your solution code are required in order to send the diagonal elements of matrix A?

Question 9: MPI Groups and Communicators (15 points)

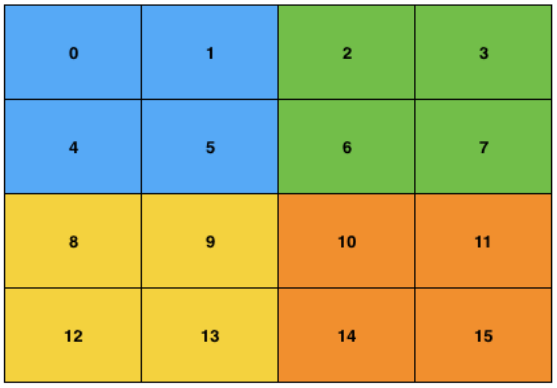
Consider the function `void diffusion_mpi(MPI_Comm comm, int N, int id)`, which solves the diffusion equation in two dimensions on a NxN grid using MPI. The communicator `comm` might be other than MPI_COMM_WORLD. The last argument enables users to perform multiple calls to `diffusion_mpi()`, as in the following example:

```
// MPI initialization

// TODO: perform the four calls in parallel, using np=P^2 ranks in total
diffusion_mpi(MPI_COMM_WORLD, 1024, 0);
diffusion_mpi(MPI_COMM_WORLD, 1024, 1);
diffusion_mpi(MPI_COMM_WORLD, 1024, 2);
diffusion_mpi(MPI_COMM_WORLD, 1024, 3);

// MPI shutdown
```

Use MPI groups and communicators to perform the 4 calls to `diffusion_mpi()` at once and within a single MPI application. Assume that the application uses $n_p = P^2$ processes and each call must be performed by a subgroup of processes. There are 4 subgroups of equal size and each subgroup corresponds to a square sub-grid of MPI processes. An example for $n_p = 16$ is depicted in the following figure:



Extend the above MPI code so as to perform the 4 calls in parallel as described above.

Note: Assume that P is even and $P >= 2$.

Question 10: Non-blocking MPI Barrier (10 points)

MPI_Ibarrier() is a non-blocking version of MPI_Barrier(). By calling MPI_Ibarrier(), a process notifies that it has reached the barrier. The call returns immediately, independent of whether other processes have called MPI_Ibarrier().

An example of using MPI_Ibarrier() is given in the following code:

```
MPI_Ibarrier(comm, &request);
work(); /* computation, other MPI communications */
MPI_Wait(&request, &status);
```

Use OpenMP to implement an equivalent version of the above code for cases where only the blocking version of MPI_Barrier() is provided by the MPI library. State any other requirements that must be satisfied by your code.

Question 11: Hybrid Quadratic Form (20 points)

The following serial code computes the quadratic form

$$Q = \vec{v}^T \cdot A \cdot \vec{w} = \sum_{ij} v_i A_{ij} w_j \tag{9}$$

```c
#include <stdio.h>


int main(int argc, char** argv)
{
    int n = 1024;


    double A[n][n];
    double v[n], w[n];


    // init A_ij = (i + 2.*j) / n^2
    for (int i=0; i<n; ++i)
        for (int j=0; j<n; ++j)
            A[i][j] = (i + 2.*j) / n / n;


    // init v_i = 1. + 2. / (i+.5)
    for (int i=0; i<n; ++i)
        v[i] = 1. + 2. / (i + 0.5);


    // init w_i = 1. - i / 3. / n
    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];


    printf("result: %f\n", result);


    return 0;
}
```

Parallelize the above code for multicore clusters using the hybrid MPI + OpenMP programming model. Using a block-row distribution every process must initialize only the needed portion of the objects v, w, A. Assume that size n is a multiple of the number of processes.

Question 12: GEMV on GPUs (25 points)

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

$$\vec{a} = M\vec{b}, \quad \text{where} \quad a_i = \sum_{j=0}^{N-1} M_{i,j} b_j \,. \tag{10}$$

```cpp

#include <vector>

#define VECSIZE 512

void random_init(std::vector<float> & v, std::vector<float> & m)
{
    // Some random initialization - this could be anything
    for(std::size_t i = 0; i < v.size(); ++i)
        v[i] = i;
    for(std::size_t i = 0; i < m.size(); ++i)
        m[i] = i;
}

void gemv_cpu(float * a, float * m, float * b, unsigned int n)
{
    for(unsigned int i = 0; i < n; ++i)
        for(unsigned int j = 0; j < n; ++j)
            a[i] += m[n*j+i] * b[j];
}

int main()
{
    unsigned int const n = VECSIZE;
    std::vector<float> a(n);
    std::vector<float> b(n);
    std::vector<float> matrix(n*n);

    random_init(b, matrix);

    //
    // TODO replace gemv_cpu.
    // The code should perform the matrix-vector multiplication
    // on the GPU and store the result in the CPU vector 'a'.
    //
    gemv_cpu(&a[0], &matrix[0], &b[0], n);

    return 0;
}
```

a) Modify the code such that the matrix-vector product is computed on the GPU. The final result should be available on the CPU.

b) Write and call a CUDA kernel (`gemv_gpu1`) for square matrices of arbitrary dimension N **employing $\sim$ N GPU threads in total**.

c) Write and call a CUDA kernel (`gemv_gpu2`) for square matrices of arbitrary dimension N **employing $\sim$ N/2 GPU threads in total**.

d) Write and call a CUDA kernel (`gemv_gpu3`) for square matrices of fixed dimension $N = 64$ **employing $\sim$ $\mathbf{N^2}$ GPU threads in total**. (*Hint:* Use N blocks.)

Notes:

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