High Performance Computing for Science and Engineering II

18.2.2019 - Lecture 1: Introduction

Instructors:
- Prof. Dr. Petros Koumoutsakos
- Dr. Sergio Martin

Teaching Assistants:
- Dr. Georgios Arampatzis (Head TA)
- Dr. Jacopo Canton
- Ermioni Papadopoulou
- Pantelis Vlachas
- Daniel Wälchli

CSElab
Computational Science & Engineering Laboratory
Today's Class

Course Information:
- Lecture Schedule
- Class Resources
- Grading

Uncertainty Quantification & Optimization
- An Overview.
- Why Uncertainty Quantification, Examples.
- The importance of High Performance Computing

Project / Homework
- An simple problem: n-Candles Problem
- (HPC) A review of algorithm optimization techniques.
Course Information
Course Contents

- Methods for Uncertainty Quantification and Optimization
- High Performance Computing (Advanced Concepts)
<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Lecture Topic</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.02</td>
<td>(SM)</td>
<td>Introduction</td>
<td>(SM)</td>
</tr>
<tr>
<td>25.02</td>
<td>(PK)</td>
<td>Bayesian UQ I - Likelihood &amp; Priors</td>
<td>(PK)</td>
</tr>
<tr>
<td>04.03</td>
<td>(PK)</td>
<td>Bayesian UQ II - Laplacian Methods, Variational Inference</td>
<td>(PK)</td>
</tr>
<tr>
<td>11.03</td>
<td>(PK)</td>
<td>Markov Chain Monte Carlo</td>
<td>(PK)</td>
</tr>
<tr>
<td>18.03</td>
<td>(SM)</td>
<td>High Throughput Computation for UQ - UPC++ &amp; Korali</td>
<td>(SM)</td>
</tr>
<tr>
<td>25.03</td>
<td>(SM)</td>
<td>Many-Core Processor Programming (CUDA) I (Basic Concepts)</td>
<td>(SM)</td>
</tr>
<tr>
<td>01.04</td>
<td>(SM)</td>
<td>Many-Core Processor Programming (CUDA) II (Optimizations)</td>
<td>(SM)</td>
</tr>
<tr>
<td>08.04</td>
<td>(PK)</td>
<td>Holiday?</td>
<td>(PK)</td>
</tr>
<tr>
<td>15.04</td>
<td>(SM)</td>
<td>Advanced MPI Concepts, MPI + OpenMP Hybridization</td>
<td>(SM)</td>
</tr>
<tr>
<td>29.04</td>
<td>(SM)</td>
<td>MPI+CUDA, Heterogeneous Computing</td>
<td>(SM)</td>
</tr>
<tr>
<td>06.05</td>
<td>(SM)</td>
<td>Communication-Tolerant Programming, The MATE Model</td>
<td>(SM)</td>
</tr>
<tr>
<td>13.05</td>
<td>(PK)</td>
<td>TBD</td>
<td>(PK)</td>
</tr>
<tr>
<td>20.05</td>
<td>(PK)</td>
<td>TBD</td>
<td>(PK)</td>
</tr>
<tr>
<td>27.05</td>
<td></td>
<td>--- Exam</td>
<td></td>
</tr>
</tbody>
</table>
Resources

Course Website:
- http://www.cse-lab.ethz.ch/teaching/hpcse-ii_fs19/
- Main resource for everything course-related.
- Homework will be posted here after class.

Discussion Board:
- All questions and comments should be sent through Piazza.
- You will receive an invitation to join.

Moodle:
- Exclusively for homework submission.
- https://moodle-app2.let.ethz.ch/course/view.php?id=10724 - For Engineers
- https://moodle-app2.let.ethz.ch/course/view.php?id=10725 - For CSE

Git Repository:
Prerequisites

**High Performance Computing**
- Intermediate C/C++ skills.
- Performance Optimization Techniques.
- Concepts of concurrency and parallelism.
- Computer Architecture (Cache, multi-core processors)
- Basic knowledge of MPI / OpenMP / Vectorization.

**Mathematics & Statistics**
- Basic knowledge of PDE systems.
- Finite Difference Methods.
- Basic knowledge of statistics and probability theory.

There's plenty of review material in the course website!
Homeworks
- 6 Homework Assignments (coding and/or report).
- Accounts for 30% of the course grade (5% per Each HW).
- Submitted via Moodle (not email).
- Issued/graded every two weeks.
- We will provide intermediate milestones for assessment.
- Practice sessions: Mondays, 10.15 – 12.00 (HG G3) -- Starting Next Week.

Exam:
- Accounts for 70% of the course grade.
- Tip: If you do the all the homework, you will be much better prepared for the exam.
**Working Together**

**Collaboration:**
- Submitting as your own: work, or parts thereof, of another person (whether it is from a book, the web, a program library or ANY other source) without proper credit constitutes academic misconduct.

- You are encouraged to DISCUSS and WORK TOGETHER with other colleagues on the homework problems, especially through Piazza.

- If the source is properly cited then it is OK: homework credit should reflect use of other’s work.

**Plagiarism:** One incident = Course Fail.
Uncertainty Quantification & Optimization
The Imperfect Paths to Knowledge

We, scientists and engineers, dedicate our work towards developing **tools** to better understand the **universe** and help us make **decisions**.

**However:** Every tool we have to understand the physical reality has an uncertainty associated to it. We need ways to measure/quantify them.
Why Quantify Uncertainty? (I)

For Decision Making. Example: Hurricane.

Image Credit: National Oceanic and Atmospheric Administration | U.S. Department of Commerce
Why Quantify Uncertainty? (II)

For Validation. Example: Speed of Light.


Definitions

- **System**: Any part of the world, natural or man-made, that we want to conceptually isolate to study. Inputs (**Parameters**) and outputs give its connections with the rest of the world. E.g., structural, mechanical, chemical, electrical, biological, and economic systems.

- **Mathematical Model**: A collection of laws and mathematical equations introduced to describe the behavior of a system (usually based on physical laws or observations). E.g., Algebraic equations, ordinary or partial differential equations, discrete equations.

- **Computational Model**: A numerical approximation or discretization of the mathematical model in a form that can be implemented in computers. Most mathematical models are too complicated to solve them exactly and **numerical approximations** are most of the time introduced to solve the problem in reasonable time.
How do we optimize/quantify uncertainty? (I)

Mathematical Model

Parameter Guess

Discretize / Approximate

Improved

Better understanding of the system and models uncertainties.

UQ Methods
CMA-ES
Markov-Chain Monte-Carlo
TMCMC

Computed Results

Compare

Experimental Measurements

More about these next week(s).

Physical System

Computational Model

Mathematical Model
Uncertainty. Finding most likely distribution of parameters \( v_1 \) and \( v_2 \).

Parameter Space

Likelihood \( v_1 = v_2 \)

* i.e. Parameters Reflect Experimental Observations

How do we optimize/quantify uncertainty? (III)
Example: Shaping wings for reduced drag.

Source: State-of-the-art in aerodynamic shape optimisation methods. N. Skinner, H. Zare-Behtash
How do we optimize parameters?

**Optimization.** Minimizing a two-parameter function.

Parameters to Optimize

Global Minimum

Local Minima

Initial Guess
Fact I: Uncertainty Quantification and Optimization methods require many model evaluations.

Fact II: For problems with many parameters, the number of model evaluations can be vast.

Our Goals:
- Develop fast and scalable computational models
- Ways to execute many evaluations given limited time and computational resources.

Our Tools:

---

High-Throughput Computing

Efficiently running many tasks for long-periods of time, using many computational resources

In this Course You’ll Learn:
Korali and UPC++

---

High Performance Computing

Build scalable algorithms that run as fast as possible in short periods of time (FLOPs)

In HPCSEI: MPI, OpenMP, PThreads
In HPCSEII: More MPI, CUDA, MPI+X
Communication-Tolerant Programming
Research @ CSELab

**Multiple Patient Data**

```
\[
\begin{align*}
\frac{dC}{dt} &= -\phi_1 C \\
\frac{dP}{dt} &= \phi_1 P(1 - \frac{P}{K}) + \phi_2 Q_P - \phi_3 P - \phi_4 \phi_3 C P \\
\frac{dQ_P}{dt} &= \phi_1 \phi_2 C - \phi_5 Q_P - \phi_6 Q_P \\
C(0) &= 0, \ P(0) = \phi_7, \ Q(0) = \phi_8, \ Q_P(0) = 0
\end{align*}
\]
```

**Hierarchical Bayesian Model**

**Prediction Per Patient**

**Nuclear Chemistry: Density Distribution of Argon Gas**

```
M X = -\nabla V(X) \\
V(r) = 4\pi \left( \frac{\sigma}{r} \right)^{\frac{P}{6}} - \left( \frac{\sigma}{r} \right)^6
```

**Material Science: Dynamics of Granular Materials**

```
F^n = -k^n y - \gamma n \frac{dy}{dt} |y|^{n-1} \\
F' = \min \left( \frac{2}{3} k^n \xi, \mu F^n \right) \\
\xi' = \int_{t_0}^{t} w_{\text{int}}(\tau) d\tau \\
k^n = \frac{2}{3} E_{\text{eff}} \sqrt{E_{\text{eff}} |y|}
```

**Bayesian Inference**

**Most Probable Model Given the Data**
A simpler case:

The n-Candles Problem

Also, your course project! 😎
We study the steady state (final) temperature distribution on a metal plate, given:
- Initial Temperature
- External Temperatures (Boundary)
- External Heat Sources.
Physical System

Example: 🔥Plate - ❄Boundaries - No Heat Sources
Physical System

Example: ❄️ Plate - ❄️ Boundaries - 3 Heat Sources🔥🔥🔥
The n-Candles Problem

We know there are 3 sources of heat (candles)
We don't know their location (x,y), intensity nor width.
Neither we know the thermal diffusivity of the material.

Total: 13 unknown parameters

You receive:
A set of experimental observations.
(Temperature measured a different points)

Your Task:
- Determine the most likely values of each of the parameters.
- Quantify the overall uncertainty for each parameter.
Mathematical Model

2D Heat Equation

\[
\frac{\partial T}{\partial t} - \alpha \left( \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) = f(x, y)
\]

Rate of Change over Time

Thermal diffusivity (Constant)

Laplace Operator

External Heat Sources

Steady-state equation with \( \frac{\partial T}{\partial t} = 0 \)

\[
- \left( \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) = f(x, y)
\]
Spatial Discretization:

$[0, 1] \times [0, 1] \rightarrow N \times N \ Grid, \ with \ h = \Delta x = \Delta y = \frac{1}{N}$
Computational Model (II)

Second-order central difference approximation to 2nd Derivatives:

\[
\frac{\partial^2 T}{\partial x^2} \approx \frac{T(x + h, y) - 2T(x, y) + T(x - h, y)}{h^2}
\]

\[
\frac{\partial^2 T}{\partial y^2} \approx \frac{T(x, y + h) - 2T(x, y) + T(x, y - h)}{h^2}
\]

\[
\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \approx \frac{T(x - h, y) + T(x, y + h) - 4T(x, y) + T(x - h, y) + T(x, y - h)}{h^2}
\]

Approximation Error!

From Steady State Heat Eq:

\[-\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) = f(x, y) \quad \text{and} \quad \alpha = 1\]

\[-T(x - h, y) - T(x, y + h) + 4T(x, y) - T(x - h, y) - T(x, y - h) \approx h^2 f(x, y)\]

System of linear equations (Ax=b).
We approach $Ax = b$ iteratively: Jacobi Method

$$A = \begin{bmatrix}
4 & -1 & -1 \\
-1 & 4 & -1 \\
-1 & -1 & 4 & -1 & -1 \\
\vdots \\
-1 & -1 & 4 & -1 & -1
\end{bmatrix} \quad x = \begin{bmatrix}
T_{1,1} \\
T_{1,2} \\
\vdots \\
T_{N,N}
\end{bmatrix}, \quad b = h^2 \begin{bmatrix}
f_{1,1} \\
f_{1,2} \\
\vdots \\
f_{N,N}
\end{bmatrix}$$

We approach $Ax = b$ Iteratively: Jacobi Method

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, n.$$ 

$$T_{i,j}^{(k+1)} = \frac{1}{4} \left( h^2 f_{i,j} - \sum_{j \neq i} a_{ij} T_{i,j}^{(k)} \right), \quad i = 1, 2, \ldots, N \quad \text{and} \quad j = 1, 2, \ldots, N.$$ 

**Problem:** Jacobi converges too slowly.
The **Multigrid method** accelerates convergence by 'smoothing' the error at multiple frequencies.

Multigrid Method (II)

Standard multigrid V-cycle

Requires special consideration for communication and cache optimizations (Great for an HPC Course project!)

Recommended Reads:
- Multigrid Methods (G. Strang)

How about other cycles? (e.g., W)

Source: Streaming Multigrid For Gradient Domain Operations On Large Images. M. Kazhdan, H. Hoppe
Homework I: The Basics

Steps:

1. Download Homework I PDF from our website. [Follow Instructions!]
   http://www.cse-lab.ethz.ch/teaching/hpcse-ii_fs19/

2. Download/Clone our C/C++ Multigrid solver code from the git repository

3. Find optimal Multigrid configuration:
   1. How many grids, given a problem size.
   2. How many relaxations upon restriction/ prolongation.

4. Apply algorithm optimization techniques to improve its performance.
   1. Cache/Memory Locality.
   2. Vectorization
   3. Pointer / Constant Optimizations.

5. Answer a few questions and fill out report.
Homework Grading

How we will grade:

1. You will only work on a single `heat2d.cpp` file containing the important parts to optimize.
2. Submit `heat2d.cpp` and `report.pdf` only (through Moodle).
3. We will test time-to-convergence (less = better) of your submission on Euler compute nodes.
4. Self-grading:
   1. If your code passes a two correctness (--verify) tests: Problem2 and Problem3 and,
   2. Reaches a certain baseline performance.
      The code is -almost- guaranteed to pass (unless it fails our tests for some other reason).
5. First-Week Milestone: A goal to be achieved by next Monday (practice).
   Not graded, but highly recommended.
6. Optional assignments. Not graded, but really really recommended.
7. PDF report: will be manually graded (should be easy if you worked with the code).

Friendly Competition: We will post the TOP 10 Homework I performances in our page.
(Lets us know if you'd like to opt-out)
Review:
Algorithm Optimization
Cache structures in modern processors benefit from both **Temporal** and **Spatial** locality.

**High Cache Line Reuse**  

**Frequent Cache Fails**

*Source:* Optimizing for instruction caches, part 1, Amir Kleen, Livadariu Mircea, Itay Peled, Erez Steinberg, Moshe Anschel, Freescale
Cache Locality (III)

Problem 2) Solver iterates the 2D space row-wise first, instead of column-wise.

Can you improve locality even further?  
Hint: **Cache Blocking (L1, L2 or both)**
Cache Locality (II)

The base Heat2d.cpp has two severe problems:

Problem 1) Rows for Unew, Uold, f(x,y), and Residual are intertwined:

How would you solve this problem?

Memory

- Unew Row 0
- Uold Row 0
- f(x,y) Row 0
- Residual Row 0
- Unew Row 1
- Uold Row 1
- f(x,y) Row 1
- Residual Row 1

Poor Spatial Locality!
Vectorization (Review)

Single Instruction, Multiple Data Paradigm
Apply the same instruction to an array of elements, instead one-by-one.

Three ways to go about it:
1. Compiler-Enabled Auto-Vectorization (GNU or Intel). (Easiest option - It's ok)
2. Intel SPMD Program Compiler (ISPC)
3. Intel Intrinsics
Auto-Vectorization


Load the Intel module and compile with `-O3` and `-qopt-report=3` to generate a detailed vectorization report:

```
icc -O3 -D NOFUNCALL -qopt-report=3 -qopt-report-phase=vec Multiply.c
```

The Auto-Vectorization optimizer requires that:
- Allocations are aligned to 16-bit
- No pointer aliasing exists
- No inter-loop dependencies are violated.

Even if your program satisfies these conditions, the compiler has no way to know unless we provide **hints**!

Investigate the effect of these hints:
- `-D NOALIAS`
- `#pragma vector aligned`
- `#pragma ivdep`

**for GNU:** `-ftree-vectorize -ftree-vectorizer-verbose=X`
Problem:
The Jacobi method requires exchanging $U_{old} \leftarrow U_{new}$ at every smoothing step.

```c
// Update $U_{old} \leftarrow U_{new}$
for (int j = 0; j < N; j++)
    for (int i = 0; i < N; i++)
        U[i][j] = U[i][j];
```

And then $U_{new}$ is updated with new values calculated by the stencil.

```c
// Apply central difference stencil
for (int j = 1; j < N-1; j++)
    for (int i = 1; i < N-1; i++)
        U[i][j] = (U[i-1][j] + U[i+1][j] + U[i][j-1] + U[i][j+1])/4 + f[i][j]*pow(h,2)/4;
```

How can we optimize the $U_{old} \leftarrow U_{new}$ update step?

How about the Gauss-Seidel Method?
Constant Optimizations

Typically handled by the compiler. But we can help! (can we?)

```c
// Apply central difference stencil
for (int j = 1; j < N-1; j++)
  for (int i = 1; i < N-1; i++)
    U[i][j] = (Un[i-1][j] + Un[i+1][j] + Un[i][j-1] + Un[i][j+1])/4 + f[i][j]*pow(h,2)/4;
```

**Hint 1:** `pow(h,2)` is a costly function.

**Hint 2:** Although it is a loop invariant, it is not taken out by the compiler. Why?

**Hint 3:** Division takes many more CPU cycles than multiplication.

**Hint 4:** Can we use associativity to simplify this operation?
Amdahl's Law (Review)

Typically used to evaluate the impact of parallelization:
E.g., How much speedup will we gain if we parallelize a section of our application.

Can also be used to estimate the impact of optimizations in sequential codes.
E.g., How much speedup will we gain by applying vectorization.

Example: Execution time of program made of A, B, and C sections.

<table>
<thead>
<tr>
<th>Section</th>
<th>Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>50</td>
</tr>
<tr>
<td>B</td>
<td>25</td>
</tr>
<tr>
<td>C</td>
<td>15</td>
</tr>
</tbody>
</table>

Question:
If vectorization reduces computing time by half and we can only apply it to one section. What's the maximum speedup we could get?

\[
Speedup_A = \frac{90}{25 + 15 + \frac{50}{2}} = 1.38x \\
Speedup_B = \frac{90}{50 + 15 + \frac{25}{2}} = 1.16x \\
Speedup_C = \frac{90}{50 + 25 + \frac{15}{2}} = 1.09x
\]
### Amdahl's Law

#### Let's Analyze: Multigrid Case.

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Grid0</th>
<th>Grid1</th>
<th>Grid2</th>
<th>Grid3</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smoothing</td>
<td>35.711</td>
<td>7.913</td>
<td>2.047</td>
<td>0.224</td>
<td>45.895</td>
</tr>
<tr>
<td>Residual</td>
<td>9.819</td>
<td>2.338</td>
<td>0.557</td>
<td>0.070</td>
<td>12.784</td>
</tr>
<tr>
<td>Restriction</td>
<td>1.776</td>
<td>0.425</td>
<td>0.113</td>
<td>0.000</td>
<td>2.315</td>
</tr>
<tr>
<td>Prolongation</td>
<td>0.000</td>
<td>12.057</td>
<td>2.855</td>
<td>0.750</td>
<td>15.661</td>
</tr>
<tr>
<td>L2Norm</td>
<td>2.150</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>2.150</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>49.456</td>
<td>22.733</td>
<td>5.572</td>
<td>1.044</td>
<td>79.525</td>
</tr>
</tbody>
</table>