January 21, 2025

Announcements

Goals

- Define problem space
- Talk about class logi.
- Survey
- ...

Review

Languages and Abstractions for High-Performance Scientific Computing

CS598 APK

Andreas Kloeckner

Spring 2025

Outline

Introduction

- Notes
- Notes (unfilled, with empty boxes)
- Notes (source code on Github)
- About This Class
- Why Bother with Parallel Computers?
- Lowest Accessible Abstraction: Assembly
- Architecture of an Execution Pipeline
- Architecture of a Memory System
- Shared-Memory Multiprocessors

Machine Abstractions

Performance: Expectation, Experiment, Observation

Performance-Oriented Languages and Abstractions

Outline

Introduction

- Notes
- Notes (unfilled, with empty boxes)
- Notes (source code on Github)
- About This Class
- Why Bother with Parallel Computers?
- Lowest Accessible Abstraction: Assembly
- Architecture of an Execution Pipeline
- Architecture of a Memory System
- Shared-Memory Multiprocessors

Machine Abstractions

Performance: Expectation, Experiment, Observation

Performance-Oriented Languages and Abstractions

Outline

Introduction

- Notes
- Notes (unfilled, with empty boxes)
- Notes (source code on Github)
- About This Class
- Why Bother with Parallel Computers?
- Lowest Accessible Abstraction: Assembly
- Architecture of an Execution Pipeline
- Architecture of a Memory System
- Shared-Memory Multiprocessors

Machine Abstractions

Performance: Expectation, Experiment, Observation

Performance-Oriented Languages and Abstractions

Outline

Introduction

- Notes
- Notes (unfilled, with empty boxes)
- Notes (source code on Github)
- About This Class
- Why Bother with Parallel Computers?
- Lowest Accessible Abstraction: Assembly
- Architecture of an Execution Pipeline
- Architecture of a Memory System
- Shared-Memory Multiprocessors

Machine Abstractions

Performance: Expectation, Experiment, Observation

Performance-Oriented Languages and Abstractions

Outline

Introduction

- Notes
- Notes (unfilled, with empty boxes)
- Notes (source code on Github)
- About This Class
- Why Bother with Parallel Computers?
- Lowest Accessible Abstraction: Assembly
- Architecture of an Execution Pipeline
- Architecture of a Memory System
- Shared-Memory Multiprocessors

Machine Abstractions

Performance: Expectation, Experiment, Observation

Performance-Oriented Languages and Abstractions

Why this class?

- Setting: Performance-Constrained Code
  When is a code performance-constrained?

$$$?

A desirable quality (fidelity, capability) is limited by computational cost.

- If your code is performance-constrained, what is the *best* approach?

Better algorithm.

$O(n^2) \to C \cdot n^2$

$O(n \log n) \to C \cdot n \log n$

- If your code is performance-constrained, what is the *second-best* approach?

Use computer efficiently. ← You

ARE HERE.

Examples of Performance-Constrained Codes

- Simulation codes
- ML training
- ML inference
- HFT
- Computer gaming

} general purpose machines

Discussion:

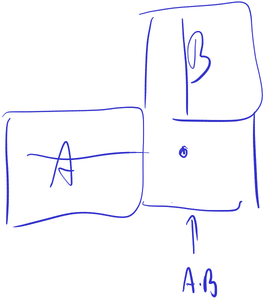
- In what way are these codes constrained?
- How do these scale in terms of the problem size? → linear ✓

What Problem are we Trying To Solve?

$$(C_{ij})_{i,j=1}^{m,n} = \sum_{k=1}^{\ell} A_{ik} B_{kj}$$

- Reference BLAS DGEMM code
- OpenBLAS DGEMM code

Demo: intro/DGEMM Performance



FMA

n^2 dof products

Goals: What are we Allowed to Ask For?

- Goal: “make efficient use of the machine”
- In general: not an easy question to answer
- In theory: limited by *some* peak machine throughput
  - Memory Access
  - Compute
- In practice: many other limits (Instruction cache, TLB, memory hierarchy, NUMA, registers)

Class web page

https://bit.ly/hpcabstr-s25

contains:

- Class outline
- Slides/demos/materials
- Assignments
- Virtual Machine Image
- Piazza
- Grading Policies
- Video
- HW1 (soon)

Welcome Survey

Please go to:

https://bit.ly/hpcabstr-s25

and click on 'Start Activity'.

If you are seeing this later, you can find the activity at Activity: welcome-survey.

Grading / Workload

Four components:

- Homework: 25%
- Paper Presentation: 25%
  - 30 minutes (two per class)
  - Presentation sessions scheduled throughout the semester
  - Paper list on web page
  - Sign-up survey: soon
- Paper Reactions: 10%
- Computational Project: 40%

Open Source <3

These notes (and the accompanying demos) are open-source!

Bug reports and pull requests welcome:
https://github.com/inducer/numerics-notes

Approaches to High Performance

- Libraries (seen)
- Black-box Optimizing Compilers
- Compilers with Directives
- Code Transform Systems
- "Active Libraries"

Q: Give examples of the latter two.

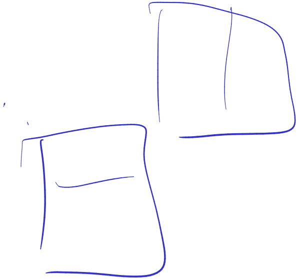
- Halide
- one DNN ← a lot of ML ("active library")

Libraries: A Case Study

$$(C_{ij})_{i,j=1}^{m,n} = \sum_{k=1}^{\ell} A_{ik} B_{kj}$$

Demo: intro/DGEMM Performance

-
-



Do Libraries Stand a Chance? (in general)

- Tremendously successful approach — Name some examples
- Saw: Three simple integer parameters suffice to lose 'good' performance
  - Recent effort: "Batch BLAS" e.g. http://www.icl.utk.edu/files/publications/2017/icl-utk-1032-20
- Separation of Concerns

  Example: Finite differences – e.g. implement ∂_x, ∂_y, ∂_z as separate (library) subroutines — What is the problem?
- Flexibility and composition