Languages and Abstractions for High-Performance Scientific Computing

CS598 APK

Andreas Kloeckner

Fall 2018

Outline

Outline

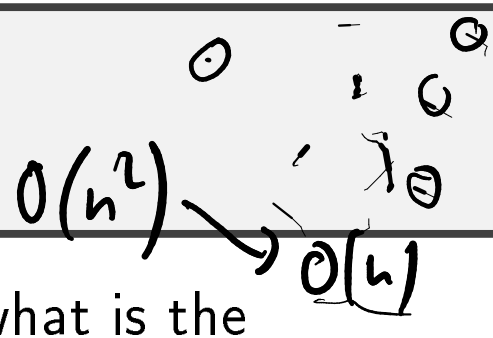
Why this class?

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

  A quantity of interest / a quality indicator is limited by comp. throughput

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

  Find a better algorithm



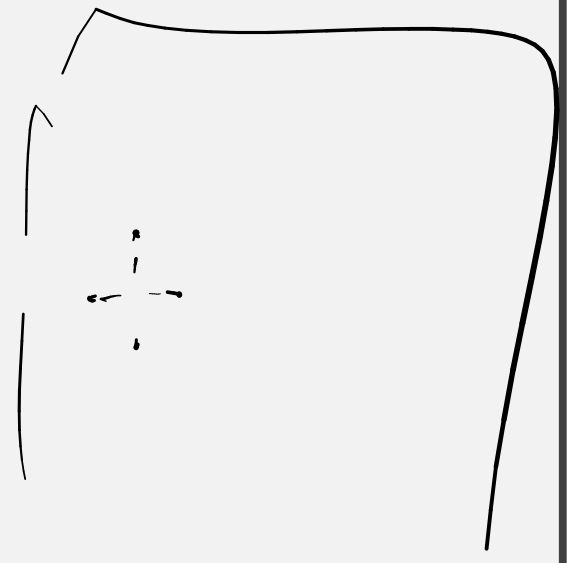
- If your code is performance-constrained, what is the *second-best* approach?

  tune the code

Examples of Performance-Constrained Codes

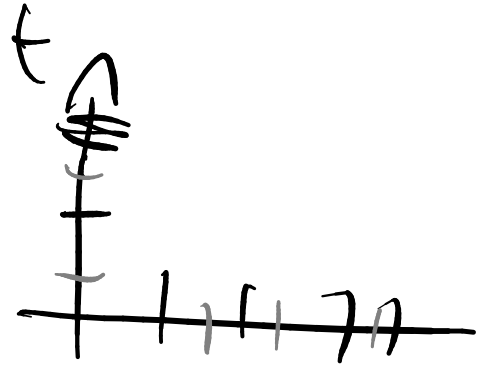
model cplx. × trainingset size × # iterations

- ML
- Data mining
- Simulation codes
  - PDE solvers
  - MD
  - MC

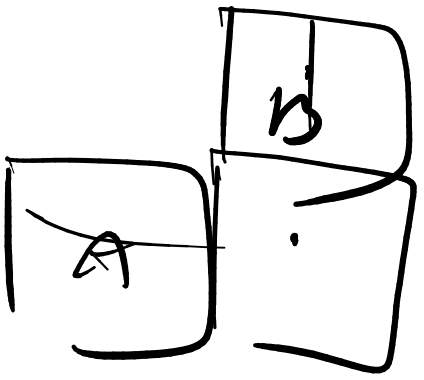


Discussion:

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



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:

https://github.com/Reference-LAPACK/lapack/blob/master/BLAS/

OpenBLAS DGEMM code:

https://github.com/xianyi/OpenBLAS/blob/develop/kernel/x86_

Demo: intro/DGEMM Performance

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-f18

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-f18

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%

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.

- CHiLL
- PyTorch

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

  BLAS, LAPACK, Eigen, deal.II, ...

- 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-1

- Separation of Concerns

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

  Memory cost

- Flexibility and composition

(Black-Box) Optimizing Compiler: Challenges

Why is black-box optimizing compilation so difficult?

- Application developer knowledge lost
  - Simple example: "Rough" matrix sizes
  - Data-dependent control flow
  - Data-dependent access patterns
  - Activities of other, possibly concurrent parts of the program
  -
- Obtain proofs of required properties
- Size of the search space
  Consider
  http://polaris.cs.uiuc.edu/publications/padua.pdf

Directive-Based Compiler: Challenges

What is a directive-based compiler?

- Generally same as optimizing compiler
- Make use of extra promises made by the user
- What should the user promise?
- Provide useful feedback to the user on what promises/directives may be helpful
  - Limited scope

Lies, Lies Everywhere

- Semantics form a contract between programmer and language/environment
- Within those bounds, the implementation is free to do as it chooses
- True at every level:
  - Assembly
  - "High-level" language (C)

  Give examples of lies at these levels:

One approach: *Lie to yourself*

- "Domain-specific languages" $\leftarrow$ A fresh language, I can do what I want!
- Consistent semantics are notoriously hard to develop
  - Especially as soon as you start allowing subsets of even (e.g.) C's integers