High Performance Computing Systems (CMSC714)

Lecture 14: Autotuning

Abhinav Bhatele, Department of Computer Science

UNIVERSITY OF MARYLAND

Summary of last lecture

- Isoefficiency

$$W = K \times t_o$$

- Helps us understand scalability and computation-communication tradeoffs
- Performance modeling
- Analytical: LogP, alpha-beta model

Autotuning

- Ultimate goal: performance portability — reasonable performance as we move from one architecture to the next
- Generation and exploration of a search space to identify the best performing option
  - Evaluated through models or empirical measurement
- Search space:
  - Code variants
  - Application parameters
  - System parameters

Different approaches

- Empirical autotuning
  - Execute each code variant or parameter combinations to identify the best performing one
  - Can also use runtime prediction models instead of running code
- Code variants
  - Code organization, data structures, algorithms
  - Parallelization strategies
  - Data movement optimization: data placement, blocking/tiling

Exploring the search space

- Brute force: try every option in the search space empirically
- How to limit the search space to a subset?
- Model-free: simulated annealing, genetic algorithms
- Model-based: analytical/empirical/machine learning models
  - Limited by accuracy of models

Software Engineering Challenges

- Offline auto-tuning can make compilation slow
  - Many variants need to be executed
- Empirical auto-tuning involves the developer in the process
- Build process for auto-tuned code can be complex
- Debugging auto-tuned code can be challenging

Libraries

- Isolate performance-critical sections behind a standard API
- ATLAS, Spiral, FFTW

Application-level Tools

- Tools allow expressing tunable parameters and exposing code variants
- If performance depends on input, tuning must be done at runtime
  - Active Harmony

ATLAS

- Automatically Tuned Linear Algebra Software based on Automated Empirical Optimization of Software (AEOS)
- Goal: Portable efficient implementation of BLAS
  - Blocking factor, different source code implementations
- BLAS has three levels:
  - Level 1 is vector-vector
  - Level 2 is vector-matrix
  - Level 3 is matrix-matrix

ATLAS

- Level 2 and 3 can benefit from memory blocking
  - Reduce movement of vector operands in Level 2
  - Reduce movement of both operands in Level 3
- Goal: Generate an L1 cache-contained matrix multiply kernel

Questions

Autotuning in High-Performance Computing Applications

- How is the optimal time for a code determined (i.e. when does an optimizer stop/ consider it has converged to the optimal solution)
- How are code variants generated concretely
- how much of the autotuning that occurs is made public to the programmer? Or does he/she just see the updated executable after autotuning completes? I'm not referring to application-level autotuning, rather compiler-level and library-level.
- The use of model-free vs model-based selection mechanisms is described in this paper (section IV). Model-free referring to the use of global and local search algorithms and model-based referring to the use of analytical performance models to predict performance metrics. Which is more commonly used and why? Or is the hybrid approach the most popular?

Questions

Automated empirical optimizations of software and the ATLAS project

- What are some concrete linear algebra optimizations that are highly architecture dependent?
- What is cache blocking?
- Do AEOS packages ever go obsolete or are they just updated whenever deemed necessary?
- It seems that the main benefit of AEOS is being able to optimize programs to run on different architectures. How does operating system factor in?
- The paper talks about the benefits of an AEOS package being written in lower-level languages vs higher-level languages. Which tend to be used more frequently and why?

Questions?

UNIVERSITY OF
MARYLAND

Abhinav Bhatele

5218 Brendan Iribe Center (IRB) / College Park, MD 20742

phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu