Component Integration and Optimization

For High Productivity and Performance

Ken Kennedy
Rice University

Participants

• LANL
  — Staff: Craig Rasmussen
  — Student: Christopher D. Rickett

• Rice
  — Faculty/Staff: Ken Kennedy, Bradley Broom*, Zoran Budimlic, Keith Cooper, Arun Chauhan*, Rob Fowler, Guohua Jin, Tim Harvey, Chuck Koelbel, John Mellor-Crummey, Steve Reeves, Linda Torczon
  — Students: Raj Bandyopadhyay, Alex Grosul, Mack Joyner, Cheryl McCosh, Apan Qasem, Todd Waterman, Rui Zhang, Yuan Zhao

• Tennessee
  — Faculty/Staff: Jack Dongarra, Keith Seymour
  — Students: Haihang You, Jelena Pjesivac-Grbovic, and Jeffery Chen

• Houston
  — Faculty: Lennart Johnsson
  — Students: Ayaz Ali, Purvi Shah, Haiyan Teng
Outline

• Component Integration Systems
  — Support for the maintenance and optimization of component libraries
  — High-productivity languages

• Retargetable High Performance Components
  — Automatic tuning of components for specific computing platforms
  — Design of adaptive components

• Application Drivers from LANL Weapons Program
  — Marmot, Telluride, Project A

• Previous Projects, Phased Down
  — High-Level Java Optimization
  — Program Preparation for Heterogenous Computing Environments (e.g., Grids)
Component Integration System

- Component integration systems are important productivity tools
- Programs constructed from them are often slow
  - No context-based code improvements can be applied
- Claim: Telescoping languages can address this problem
  - Can be applied to construct component integration systems that yield high-performance applications
  - Can make components usable in contexts that have been previously considered impractical
- ASC Relevance
  - Component-based software is critical for productivity and reliability
  - Performance must be high for software to be usable
  - Useful to prototype in high-productivity language (Python, Matlab)
Component Integration Challenge

• Integration of different component libraries that
  — Implement data structures (e.g., sparse matrices)
  — Implement functions on data structures (e.g., linear algebra)

• Problem: Performance
  — High function overhead for data structure access (frequently invoked)
  — Need optimization for special contexts
    - e.g., invocation in loops

• Claim: Telescoping languages can handle this well
  — Advance generation of specialized entries
  — Transformation pass to perform substitution
Telescoping Languages

Component Library

Optimizer Generator

Could run for hours

Application

Application Translator

Understands library calls as primitives

Application Optimizer

Vendor Compiler

Optimized Application

Scripting language or standard language, (Fortran or C++)
Telescoping Language Advantages

• Optimized script compilation times can be reasonable
  — Investment in library analysis speeds script optimization

• High-level optimizations possible
  — Exploit library designer’s knowledge of routine properties
  — Specialize library routines during optimizer generation to exploit expected calling sequences
    - Apply high-level transformations based on identities
    - Factor and/or fuse library primitives as appropriate

• User retains substantive control over performance
  — Mature code can be built into a library, annotated with properties to aid optimization and fed to library compiler

• Reliability can be improved
  — No hand coding to context
What We Have Done

• Developed base-language compiler technology
  — Type inference: Key to generation of C or Fortran from Matlab, S, or Python
    - Useful even if C++ or Fortran is your scripting language

• Conducted preliminary studies
  — Matlab SP (Signal Processing), LibGen (library generation)
    - Six papers, one Ph.D., two Master's
  — R compilation (funded separately by DOD)

• Demonstrated benefits of telescoping languages as component integration system (via LibGen)

• Developed strategy for generalized data structures
  — Including addition of parallelism to scripting languages (funded by ST-HEC program from NSF/DARPA)

• Met with Marmot team to explore collaboration opportunities
Library Generator (LibGen)

• **ARPACK**
  - Prof Dan Sorensen (Rice CAAM) maintains ARPACK, a large-scale eigenvalue solver

• **Methodology**
  - He prototypes the algorithms in Matlab, then generates 8 variants in Fortran by hand:
    - \{Real, Complex\} $\times$ \{Symmetric, Nonsymmetric\} $\times$ \{Single, Double\}
    - Dense vs Sparse handled by special interface

• **Could this hand generation step be eliminated?**
  - **Answer:** YES
  - **Key technology:** Constraint-based type inference
    - Polynomial time algorithm to compute type jump functions
      - Map input types to variable types
Value of Specialization

- sparse-symmetric-real
- dense-symmetric-complex
- dense-symmetric-real
- dense-nonsymmetric-complex

0.072 sec
Distribution and Parallelism

• **Strategy:** Add distribution to Matlab arrays
  – Standard libraries plus user-implemented distributions
  – Distribution libraries (e.g. block) packaged with language

• **Telescoping compiler optimizes distribution accesses**
  – Mimics standard optimizations, such as vectorization of accesses
  - This is simply procedure strength reduction

• **Parallelism by HPF-style computation generation**
  – Computation performed close to data
  – Rice has strong HPF technology in place
  – HPF compilation (slow) applied only to components (not to script)

• **Project spun out into NSF ST-HEC proposal**
  – Funded through DARPA HPCS
LACSI Interactions

• Priorities and Strategies Meetings
  — Inputs from Steven Lee and Ken Koch were pivotal in direction change

• Attended Common Component Architecture (CCA) Workshop
  — LACSI Symposium 2002

• Initial Components Workshop (April 16–17, 2003)
  — Organized by Craig Rasmussen

• Discussions with Marmot Group
  — Monterrey Methods Workshop (March 16–18, 2004)
  — Components Workshop at LANL (June 24, 2004)
    - Developed an outline plan for collaboration
What We Plan to Do

• Seek (and solve) component integration challenge problem
  – Based on work from ASC applications
  – Emphasis on efficiency of frequent component-crossing
    - Integration of data structure and function

• Continue interactions with Marmot Project
  – Goal: build tools to help them on their second or third iteration
    - Build on work on component integration and optimization of object-oriented languages

• Explore opportunities in other ASC codes

• Relevance to ASC
  – Success will make it easier to use modern component-based software development strategies in ASC codes
    - Without sacrificing performance
Automatic Component Tuning

- Participants: Four Groups within LACSI
  - Tennessee: Jack Dongarra
    - Collaboration with LLNL ROSE Group (Dan Quinlan, Qing Yi)
  - Rice: Ken Kennedy and John Mellor Crummey
    - Students Apan Qasem and Yuan Zhao
  - Rice: Keith Cooper, Devika Subramanian, and Linda Torczon
    - Students Todd Waterman and Alex Grosul
  - Univ of Houston: Lennart Johnsson
    - Students Ayaz Ali, Purvi Shah, Haiyan Teng
Automatic Component Tuning

• **Goal:** Pretune components for high performance on different computing platforms (in advance)
  
  — *Models:* ATLAS, UHFFT
  
  — Generate tuned versions automatically

• **Strategy:** View as giant optimization problem with code running time as objective function
  
  — For each critical loop nest:
    
    - Parameterize the search space
    - Prune using static analysis
    - Employ heuristic search to find optimal point and generate optimal code version

— **Typical optimizations:**
  
  - Loop blocking, unroll, unroll-and-jam, loop fusion, storage reduction, optimization of target compiler settings, inlining, optimization of function decomposition
Automatic Tuning

• Successes
  — Experimental infrastructure
    - LoopTool, MSCP, ATLAS2, CODELAB
  — Large-scale experiments
  — Principles demonstrated
    - Effectiveness of heuristic search
  — Papers published
    - Seven refereed publications and one technical report (see web site)

• Relevance
  — Dramatically increases productivity of scientific programming

• Connections to ASC
  — Sweep3D, Marmot, Truchas, Code A
Some Previous Accomplishments

• **JaMake Java Framework**
  - Collaboration with CartaBlanca Project
  - Performs object inlining on arrays of objects
    - Overcomes the cost of using full OO polymorphism
    - Achieved 80% improvement on the LANL Parsek code
  - Results apply to C++ and Python
  Attracted NSF funding, published 6 refereed papers

• **Grid Research**
  - Drove performance prediction research
  - Effective performance-model based scheduling
  - VGrADS: NSF ITR (Large)
  - Ideas for Grid in a box
    - Many future supercomputers will have heterogeneous computing components: good scheduling will be critical for performance
Summary

- **Component integration languages and frameworks**
  - High Level: Matlab, S, Python plus component libraries
  - Low Level: C, C++, Fortran

- **Compilation technology**
  - Type inferencing to drive translation to C or Fortran
  - Telescoping languages to pre-optimize libraries
  - Parallelism in scripting languages
    - Parallelism based on distribution

- **Component Autotuning**
  - Goal: ATLAS-style automatic tuning for generalized applications, UHFFT-style automatic tuning for decomposable (library) components
  - Exploring heuristic search and static search-space pruning

- **Technology Transfer**
  - Focus component integration on problems arising from Marmot project
  - Automatic tuning applicable to general languages