Future Applications and Architectures

And Mapping One to the Other

Ken Kennedy
Rice University

http://www.cs.rice.edu/~ken/Presentations/FutureLACSI06.pdf
• What is the predominant technical computing language?
  — Matlab!
    - Over a million Matlab programmers!
  — Runners up:
    - Python, R/S++, LabView
  — Also-rans
    - Fortran, C, Java, C++

• What is the emerging predominant application structure?
  — Community Codes
  — Domain specific components linked by scripting languages
  — Architecture hidden from end-user
    - Embedded in component libraries

• What is the Problem?
  — Why can't these programmers use scalable parallel computers?
    - Well they can, but they have to rewrite in X+MPI
Architectures

- **Architectures of leading-edge machines are becoming more complex**
  - Nodes have on and off-chip parallelism
  - Chips have many cores
    - Different cache hierarchies on chip (shared, separate, partially shared)
  - Nodes and even chips are heterogeneous
  - Data transfer rates differ at different levels in the hierarchy
  - Network bandwidths differ widely
    - As does IO bandwidth

- **Principle**
  - The Matlab developer does not really want to know about this
    - Even though the heroic LANL developer does
Performance Issues In Matlab

- Variables are dynamically typed
  - Can assign to an non-existent row or column
    - Expands the array

- Whole array assignment
  - Requires scalarization in compiling
  - Loop fusion and tiling needed to enhance locality

- Until recently, no parallel implementation
  - Mathworks: added MPI
    - Distributed arrays and unsynchronized loops in planning
  - Research implementations
    - Distributed arrays with one distribution: block-cyclic
    - ScaLAPACK for array operations

- Comment: these parallelism strategies limit the range of applicability
Problems in Component Integration

• Component Integration Frameworks Add Overhead for Cross-component Calls
  – CCA: Factor of 3 over Fortran (best known)
  – Reasons: Components treated as interchangeable black boxes
    - Good for development, but not performance
    - No cross-component optimizations (e.g., inlining)
    - Developers may be willing to sacrifice performance when application is mature

• Proposal: New Architecture for Component Integration Framework
  – When application is mature, invoke whole-application compiler
  – Reduce compile time by pre-optimizing components
Telescoping Languages

- Component Library
- Application

Optimizer Generator

Application Translator

Application Optimizer

Vendor Compiler

Optimized Application

Scripting language or standard language, (Fortran or C++)

Could run for hours

Understands library calls as primitives
Applications of TL

• **Compilation of Matlab + Toolboxes**
  - Translation to C or Fortran
    - Requires constraint-based type analysis (including size and shape)
  - Toolboxes pre-compiled to different contexts without knowledge of calling program
    - Requires that type analysis compute “jump functions” that determine types within the code, and types of outputs, as functions of input parameter types (sizes are variables)

• **Generation of Fortran Versions of a Library (ARPACK)**
  - One page of Matlab = 8 variants $\times$ 50 pages of Fortran
    - $\{\text{real, complex}\} \times \{\text{dense, sparse}\} \times \{\text{symmetric, nonsymmetric}\}$
LibGen ARPACK Performance

![Bar Chart]

- MATLAB 6.1
- MATLAB 6.5
- ARPACK
- LibGen

Comparison of performance metrics for different environments.
Value of Specialization

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

0.072 sec
Applications of TL

• **Compilation of Matlab + Toolboxes**
  - Translation to C or Fortran
    - Requires constraint-based type analysis (including size and shape)
  - Toolboxes pre-compiled to different contexts without knowledge of calling program
    - Requires that type analysis compute “jump functions” that determine types within the code, and types of outputs, as functions of input parameter types (sizes are variables)

• **Generation of Fortran Versions of a Library (ARPACK)**
  - One page of Matlab = 8 variants x 50 pages of Fortran
    - \{real,complex\} x \{dense,sparse\} x \{symmetric,nonsymmetric\}

• **Parallelism in Matlab**
  - Add distributed arrays (many different distributions)
  - Specialize generalized operators to different distributions using HPF
  - Produce Fortran + MPI
Performance: 2D Stencil

512 x 512
(Block, Block)

Single-node compilation improvement
Parallel Efficiency: 2D Stencil

512 x 512
(BLOCK,BLOCK)

Single-node compilation improvement
Summary

• **Goal: Extend HPC to a Broader Technical Community**
  - They do not want to know about architectures

• **Application Structure in the Future**
  - Scripting languages (Matlab, Python, etc.) plus components
  - Components may be written in scripting languages

• **Strategy for Achieving Performance: Telescoping Languages**
  - Pre-compile and optimize components to likely run-time contexts
    - Type analysis, cross-component optimizations, generate standard language
  - When compiling application, select component variants specialized to types of input parameters

• **Parallelism**
  - Distributed arrays, single thread of control
  - Translate to Fortran
  - Specialize operands using HPF compiler technology