Programming Support Software for High Performance Computing

Ken Kennedy
Center for Research on Parallel Computation
Rice University

1. Scope
   languages, compilers, libraries, tools
   high-end science and engineering

2. Successes
   technologies that have made a difference

3. An Assessment
   how have we done in technology transition?

4. Challenges for the Future
   and what we must do to meet them
Parameters of This Talk

- **Focus**
  - Compilers, Languages, and Libraries
  - Integrated Tool Technologies
  - High Performance Computing
  - Scientific, Engineering, and Analytic Commercial Applications

- **Goal**
  - To assess the impact of technologies developed by the computer and computational science research community on commercial practice and the user community

- **Concerns**
  - Is research producing the right ideas and capitalizing them?
  - Are the technology transfer mechanisms working?
The HPC Landscape

• Architectures
  - Vector processor
  - Shared-memory multiprocessor (SMP)
  - Distributed-memory message-passing
  - Distributed shared memory (DSM)
  - SMP clusters
  - Networks of workstations or PCs
  - Distributed, heterogeneous networks
  - Multithreaded shared memory

• Languages
  - Fortran
    Fortran 77, Fortran 90, HPF
  - Non-Fortran
    C, C++, Java
  - Message-passing libraries, thread packages
Challenges for HPC Software

• Machine Independent Parallel Programming
  - need for protection of programming investment
    write once and tune, single source image
  - need for high performance on each target machine
    close to hand coded parallel program in native programming interface
    independent of algorithm choice

• Ease of Use
  - high level of abstraction
    freeing programmer from gory details of managing complex hardware
    increasing accessibility of parallelism

• Programming Tools
  - mechanisms for assisting in building, debugging and tuning parallel programs
    user control from a high-level interface

• Market Penetration
  - must have a familiar environment on each commercial platform
  - users must accept languages and tools
Software Successes I

- **Compiler Memory Hierarchy Management**
  - Single-node memory hierarchy
    - register allocation, register blocking, cache blocking, cache prefetching
  - Memory reorganization for parallelism
    - reduction of false sharing, co-location for variables used together
  - Inclusion of I/O in memory hierarchy

- **Compiler Extraction of Parallelism**
  - Automatic methods
    - effective for loops on shared-memory multiprocessors
  - Language and compiler support for data parallelism
    - HPF available on every parallel platform

- **Support for Portability**
  - Standards
    - HPF, MPI, HPC++, OpenMP
  - Portable platforms
    - Java
Software Successes II

• Libraries
  - Communication Libraries
    MPI, PVM, Active Messages, BLACS, Chaos ...
  - Math Libraries
    ScaLAPACK, PETSc, CMSSL, ARPACK...
    templates
  - Data Structure Libraries
    distributed array data structures (DAGH, P++, POOMA)
    quad and oct trees (fast multipole methods)

• Integrated Tools
  - Performance Analysis and Tuning
    distributed-memory performance monitoring
    Pablo, Gist, Upshot,...
    performance related to source language text (Pablo+HPF)
    dynamic instrumentation (Paradyn)
  - Debugging
    source level debuggers on a per-processor basis
    postmortem replay
• Support Technologies

  - **Interprocedural analysis and optimization**
    - interprocedural transformations
    - whole-program analysis
    - base technology developed in research systems
    - several commercial implementations
    - some interprocedural analysis at load time

  - **Integer set analysis methods**
    - precise dependence analysis
    - precise loop splitting
    - issue: can it be efficient enough for commercial compilers?

  - **Communication analysis**
    - placement of communication to overlap with computation
    - multiprocessor prefetching
    - strategies for regular and irregular computations
An Assessment

• Impact
  - Overall the harvest of compiler and tool technologies for scalable parallel computing has been meager, with few unqualified successes
    Message-passing libraries (MPI, PVM)
    Performance analysis and tuning tools

• Implications
  - Little support beyond MPI for machine-independence
  - Node performance in parallel configurations is worse due to memory hierarchy, remote communication
  - Limited penetration of scalable parallelism in real applications
  - Divergence of programming models
    Increased reliance on multiple-language solutions
  - Shaky foundation for distributed computing problems will be much harder
Reasons for Limited Success

• Pace of architectural change
  - shared memory, distributed memory, distributed shared memory, clusters, heterogeneous distributed networks
  - difficult to establish a general strategy

• Investment strategy flaws
  - direct investment in software technologies low
  - grand challenge applications often focused on performance and results at the expense of technology development

• Technology transfer mechanism problems
  - users want technologies that are mature, reliable, and standard
  - software technologies are complex
  - small size of high-end HPCC market
    - limited corporate resources for software investment
    - companies reluctant to gamble on new technologies
  - early deployments of new technologies are notoriously unreliable
    - users become disaffected
Software Technology: Disappointments

- Long lag time for technology transfer to industry
  - example: HPF compilers beginning to mature only today
  - compiler infrastructure may help (Lam talk)
  - compiler reliability is weak
    - compiler technologies are complex
  - tools have been slow to emerge
    - source debuggers, performance tools tied to language

- Machine independence still elusive
  - problems keeping up with pace of architecture
  - shortage of tunable portable libraries

- Software support for dynamic problems is weak
  - example: irregular problems require run-time support

- Few great leaps forward in level of language abstraction
  - focus remains on back end
Future Architectures

• Petaflops/Petaops Architectures
  - Faster processors, high processor count
    Projection for 2007: \textbf{10,000 to 100,000} processors at 10-100 GF each
  - Deep memory hierarchy
    up to 10 levels
  - High levels of parallelism required
    at least 10 million way parallelism must be found in the application
    used to exploit parallel processors and hide memory latency

• Distributed Heterogeneous Systems
  - geographically distributed high-end systems
  - networks of PCs
  - message passing across nodes, shared memory within
  - nodes and links have varying performance
    number and power of processors, network bandwidth
    availability and performance may differ over time
What We Must Do

• Attack Performance Bottlenecks
  - ameliorate the memory hierarchy problem
    including I/O
  - find more parallelism

• Support Portable High-Performance Computing
  - develop and support standards

• Raise the Level of Programming Abstraction
  - abstract specification of parallelism
  - problem-solving environments, scripting languages
  - support for Interoperability
    multiple languages, multiple applications

• Rethink Compiler Design
  - optimizations postponed until load and run time
  - integration of data and run-time information into compilation
  - integration of tool support
  - increased reliability through component structure
  - mechanisms to protect source code
Programming in the Future

• Assertions
  - standard programming too hard
    large programs, legacy code (MADIC: 10,000 programs)
  - programs will be built from preexisting components
    different programming languages
    standard libraries
  - high performance will continue to be important

• Conclusions
  - future programming: two developer classes
    specialist and end user
  - future programming languages for end user:
    interfaces to component libraries
    problem-solving environments (PSEs)
    scripting languages
    languages may be graphical
Script-Based Programming System

• Compilation of all components to single intermediate code
  - script, whole applications, all single-language components
  - intermediate language must support parallelism and memory hierarchy

• Whole-system optimization of intermediate code
  - interprocedural analysis
  - systematic inlining

• Translation to high-performance parallel programs
  - standard intermediate code + communication

• Compilation to native machines
  - portable or native compilers
Script-Based Programming System

- **Script**
- **Program Component**
  - **Translation System**
  - **Intermediate Code**
    - **Whole-System Compiler**
    - **Portable IC Compiler**
    - **Native IC Compiler**
      - **Target Machine 1**
      - **Target Machine 2**
      - **Target Machine 3**
Compilation with Data

- **Program**
- **Frequently Changing Data**
- **Rarely Changing Data**

**Extended Optimizing Compiler**

**Object Program**

**Answers**
New Compiler Architecture

• Flexible Definition of Computation
  - Parameters
    - program scheme
    - subprogram source files \((s_1, s_2, ..., s_n)\)
    - run history \((r_1, r_2, ..., r_k)\)
    - data sets \((d_1, d_2, ..., d_m)\)
    - target configuration

• Compilation = Partial Evaluation
  - may be several compilation steps
    - information available at different times

• Program Management
  - Must decide when to back out of previous compilation decisions in response to change
  - Must decide when to invalidate certain inputs
    - previous run histories
Trusted Compiler

Source from Company A

Source from Company B

Library Source from Vendor C

Source encrypted using public key for <compiler,machine>

Validated to compile on machine x

Trusted Compiler

Target Machine

Trusted Loader
The Role of Tools

User

Compiler

Source

Compiled Code

Target Machine

Debugging and Analysis Tool

Execution Record Keyed to Source

Info about Transformations

Execution Record
Composition of Tools

User

HL Compiler

Compile Info

Source

Virtual Machine

Execution Record

LL Compiler

Compile Info

Target Machine
Challenges for Compilation Research

• Complex Architectures
  - more parallelism
  - deeper memory hierarchies
  - heterogeneous distributed systems
    late binding of actual machine configuration

• Complex Applications
  - irregular, adaptive, dynamic computations
  - multiple programs, multiple languages, multiple parallelism styles
  - script-based system compositions

• What We Must Do
  - Attack performance bottlenecks
  - Support portable high-performance computing
  - Raise the level of programming abstraction
  - Rethink compiler design
    support for performance, multilevel compilation, reliability, security