High Performance Computing: Achievements and Challenges

Ken Kennedy
Center for Research on Parallel Computation
Rice University

1. High Performance Computing and Communications Program
   diversity of parallel architectures
   challenges for software support

2. Machine-Independent Parallel Programming
   Example: High Performance Fortran

3. Future Architectures and Applications
   petaflops architectures
   distributed heterogeneous computer systems
   new programming architectures

http://www.cs.rice.edu/~ken/Presentations/Hounix.pdf
Status of Scalable Parallelism

• Dream
  - virtually limitless computing power at low cost
  - performance scalable from one to thousands of processors
    applications would be programmed to scale automatically
  - easy portable programming

• Reality
  - successful at only moderate levels of scalability
    state of the art: up to 32-way multiprocessor workstations
  - modest progress in programmability and scalability
  - usage primarily in research
  - limited penetration in industry
    independent software vendors (ISVs) still reluctant
    limited protection of programming investment
Trends in Architecture

- Ascendancy of workstation and desktop technology
  - shared-memory multiprocessors
  - multiprocessor personal computers

- Deeper memory hierarchies
  - Petaflops: may go to 10 levels

- Distributed shared memory
  - microprocessors moving to 64 bit addressing
  - desire to make maximum use of previous advances in cache structure

- Clusters of workstations or PCs
  - DSM and message-passing

- Heterogeneous networks
  - nodes of different power, architecture, data representation
  - varying network bandwidths
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
HPC Software Successes

• Compiler Memory Hierarchy Management
  - register allocation, register blocking, cache blocking, cache prefetching
  - Memory reorganization for parallelism
    reduction of false sharing

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

• Support for Portable Parallel Programming
  - HPF, MPI, HPC++, OpenMP, Java

• Parallel Libraries
  - Communication, Math, Data Structures

• Integrated Tools
  - Performance Analysis and Tuning
  - Debugging
Portability: The HPF Story

• Goals:
  - machine independent parallel programming support
  - high level of abstraction
    single thread, shared memory, implicit communication
  - interoperability

• Strategy
  - distribution directives for memory layout
  - compiler implemented for each machine
tailors code and communications to target architecture
  - EXTRINSIC interface

• Problems
  - compilers slow to mature
    early performance unsatisfactory
  - incomplete application coverage
  - need for library support
  - lack of tools
HPF Portability Strategy

- Fortran 90 Program
- Data Distribution Directives
- HPF Program
- Portable HPF Compiler
- Native HPF Compiler
- Fortran 77 Plus MPI
- Target Machine 1
- Target Machine 2
- Target Machine 3
REAL A(1023,1023), B(1023,1023), APRIME(511,511)
!HPF$ TEMPLATE T(1024,1024)
!HPF$ ALIGN A(I,J) WITH T(I,J)
!HPF$ ALIGN B(I,J) WITH T(I,J)
!HPF$ ALIGN APRIME(I,J) WITH T(2*I-1,2*J-1)
!HPF$ DISTRIBUTE T(BLOCK,BLOCK)

!HPF$ INDEPENDENT, NEW(I)
DO J = 2, 1022  ! Multigrid Smoothing Pass (red-black relaxation)
!HPF$   INDEPENDENT
DO I = MOD(J,2), 1022, 2
   A(I,J) = 0.25*(A(I+1,J) + A(I+1,J) + A(I,J-1) + A(I,J+1)) + B(I,J)
END DO
END DO

!HPF$ INDEPENDENT, NEW(I)
DO J = 2, 510   ! Multigrid Restriction
!HPF$   INDEPENDENT
DO I = 2, 510
   APRIME(I,J) = 0.05*(A(2*I-2,2*J-2) + 4*A(2*I-2,2*J-1) + &
&                A(2*I-2,2*J) + 4*A(2*I-1,2*J-2) + 4*A(2*I-1,2*J) + &
END DO
END DO

! Multigrid convergence test
ERR = MAXVAL( ABS(A(:,,:) - B(:,:)) )
HPF: The Good News

• Compilers
  - Now available for every major platform
  - Language support more complete
  - Supported by all vendors

• Benchmarks and Applications
  - HPF within a factor of 2 of MPI on NAS benchmarks
  - Over 20 major applications in the CRPC database
    some over 100K lines
    performance record: 85 GF on 256 processor T3E
  - Standard migration path for CM-5 users

• Libraries and Tools
  - Portable interface to ScaLAPACK almost ready
  - Interfaces for other libraries underway
  - Pablo performance tool available, others under development

• Coverage
  - Much improved by extensions in HPF 2
An Assessment of HPC Software Efforts

• Impact on End Users
  - 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
  - Progress in research has not been effectively transferred to practice

• Implications
  - Little support beyond MPI for machine-independence
  - Node performance in parallel configurations may be 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
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

• HPCC investment strategy counterproductive
  - direct investment in software technologies inadequate
  - grand challenge applications focused on performance and results at the expense of technology development

• Technology transfer mechanisms are flawed
  - 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
Future Architectures

- **Petaflops/Petaops Architectures**
  - Faster processors, high processor count
    - Projection for 2007: \(10,000\) 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
Memory Hierarchy Management: Key Ideas

- **Program Reorganization**
  - register and cache blocking
  - loop splitting

- **Software Prefetching**
  - prefetch selection and placement

- **Memory Reorganization**
  - variable grouping on cache lines
  - array storage reorganization
  - dynamic reorganization schemes

- **Inclusion of I/O in Memory Hierarchy**
  - extension of cache techniques
  - reorganization, prefetching
  - improvement factors in the hundreds
Distributed Heterogeneous Computing

• Program Decomposition
  - Distributed objects
  - Distributed data structures
  - Adaptive distribution of standard data structures

• Scheduling
  - Static and dynamic performance estimation
  - System performance parameterization
  - Adaptive load matching

• Latency Management
  - Interaction with Quality-of-Service facilities
  - Fast translation of data formats
Compilation for Heterogeneous Grids

• Challenges
  - dynamic, changing nature of target
  - difficulty of management by hand

• Solutions
  - A new program preparation architecture
Programming in the Future

• Challenges
  - programming is hard
  - professional programmers are in short supply
  - high performance will continue to be important

• A Solution: Make the End User a Programmer
  - professional programmers develop components
  - users integrate components using:
    problem-solving environments (PSEs)
    scripting languages (possibly graphical)
    examples: Tcl, Visual Basic, AVS, Khoros

• Compilation for High Performance
  - translate scripts and components to common intermediate language
  - optimize the resulting program using interprocedural methods
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
- Intermediate Code
- Target Machine 1
- Target Machine 2
- Target Machine 3
- Whole-System Compiler
- Translation System
- Portable IC Compiler
- Native IC Compiler
- Program Component
- Program Component
Compilation with Data

Program

Extended Optimizing Compiler

Rarely Changing Data

Frequently Changing Data

Object Program

Answers
New Compiler Architecture

• **Flexible Definition of Computation**
  - **Parameters**
    - program scheme
    - subprogram source files \( (s_1, s_2, \ldots, s_n) \)
    - run history \( (r_1, r_2, \ldots, r_k) \)
    - data sets \( (d_1, d_2, \ldots, d_m) \)
    - 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
New Technology Program Environments

- Program Definition
- Input Data Definition
- Run History
- Source Files
- Whole-Program Compiler
- Machine Code
- Data and Trace Files
Trusted Compiler

- Source from Company A
- Source from Company B
- Library Source from Vendor C
- Target Machine

Source encrypted using public key for <compiler,machine>

Validated to compile on machine x

Trusted Compiler

Trusted Loader
The Role of Tools

- User
  - Source
  - Execution Record Keyed to Source
- Compiler
  - Compiled Code
  - Info about Transformations
- Target Machine
- Debugging and Analysis Tool
  - Execution Record

Source

Compiled Code

Info about Transformations

Execution Record
Composition of Tools

User

HL Compiler

HL Debugger

Compile Info

Virtual Machine

Source

Execution Record

LL Compiler

LL Debugger

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