Prospects for Architecture-Independent Parallel Programming

Ken Kennedy
Center for Research on Parallel Computation
Rice University

1. Parallel Computing
   What We Did Wrong
   Architecture-Independent Parallel Programming

2. High Performance Fortran
   Strategy and Features
   Obstacles to Success
   Research Directions
The Dream of Parallel Computing

- Unlimited Computing Power
  - inexpensive commodity processors
  - LEGO-style interconnection
  - scalable from one to thousands of processors
  - use of networks of existing workstations

- Easy Porting of Applications
  - automatic parallelization
  - widely-available machine-independent parallel programming languages
  - parallel implementation of libraries
  - simple parallelization of existing algorithms and applications

- Are We There Yet?
  - NO
  - but we have made a lot of progress
NAS Parallel Benchmarks

Source: NAS Parallel Benchmark Results 10-94
Nas Technical Report NAS-94-001
October 1994
Parallel Computation: What We Did Wrong

- Focused Too Much on Architectures
  - if we build it, they will come — NOT
- Failed to Identify Software as the Problem Early Enough
  - users will not move from easy programming to hard programming without due cause
  - many standard algorithms do not parallelize well
- Underestimated the Software Problem
  - vectorization took 10 years
  - parallelization is much more complicated
- Failed to Ask Non-Research Users What They Must Have
  - industry and independent software vendors must have a machine-independent, standard programming interface (e.g. PVM, MPI or HPF) to protect their programming investment
Architecture-Independent Parallel Programming

• Definition
  - a programming language, its compilers, and run-time system support architecture-independent parallel programming if compiled code \( \cong \) hand code for same algorithm for each target architecture

• Strategies
  - Portable libraries
    PVM, Linda
  - Software distributed shared memory (DSM)
    simulate DSM using system paging mechanisms
  - Standard interfaces
    MPI
  - Standard languages
    HPF, HPC++
High Performance Fortran Goals

- Support for scalable parallel systems
- Focus on data parallelism
- Architecture independence
  - distributed-memory MIMD
  - shared-memory MIMD
  - synchronous array (SIMD)
- Abstract view of parallelism
  - shared-memory programming interface
  - implicit communications
  - explicit and implicit parallelism
HPF Strategy

- Fortran 90
- Sequential Machine
- Data Distribution Directives
- CM-2
- IBM SP-2
- HPF
- HP/Convex SPP2000
HPF Language

• Fortran 90 + Data Distribution
  
  ```fortran
  !HPF$ TEMPLATE D(256,256)
  !HPF$ ALIGN A(I,J) WITH D(I+1,J)
  !HPF$ DISTRIBUTE D(BLOCK, CYCLIC)
  ```
  
  virtual processor array
  align data elements
  map to processors

• Explicit Parallelism
  - Fortran 90 array statements

• Implicit Parallelism
  - "owner computes"

• Extended looping and aggregate assignment
  - FORALL (single-statement and block)
  - DO INDEPENDENT
(Block, Cyclic) Distribution

Processes

Template Elements
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 Commercial Interest

• Announced HPF Products
  - Applied Parallel Research
  - CDAC
  - Cray Research
  - Digital Equipment
  - Fujitsu
  - Hitachi
  - HP
  - IBM
  - Intel
  - Meiko
  - Motorola
  - NA Software
  - NEC
  - Pacific Sierra Research
  - Portland Group
  - Sun
  - Transtech

• Announced HPF Efforts
  - ACE
  - Lahey
  - NAG
  - nCUBE

• Interested
  - EPC
  - SGI
  - Tera
HPF Usage

• Installations
  - PGI reports over 100 site licenses

• Applications
  - 2 Grand Challenge projects using HPF
  - 10 Collaborations with PGI (8000-25,000 lines)
  - NCSA: PGI on SGI is migration platform for CM-5
  - European-funded applications (extension of Europort)
  - CRS4 Seismic Migration Code "GeoComp"
  - Amoco: Reservoir Modeling

• Benchmarks
  - NAS Benchmarks: PGI compiler within 20-50% of MPI on Class A benchmarks
  - Cornell HPF Study: IBM HPF compiler outperforms hand-coded MPI on a financial application
Problems for HPF

- Compilers slow to mature
  - Fortran 90 features supported inconsistently
  - compilation for highest efficiency complex
  - initially, efficiency of object programs unsatisfactory
  - early users may become discouraged

- Needed features are missing
  - support for irregular problems
  - task parallelism
  - high performance input/output

- Complex relationship between program and performance
  - explanatory and diagnostic tools are needed
Responses to HPF Problems

• **HPFF 2**
  - new round of standardization
    target completion Supercomputing 96
  - new approved extensions
    irregular distributions, ON clause, task paralleism, asynch I/O

• **CRPC Research**
  - Fortran D95 Compiler
    advanced whole-program compilation framework
  - support for irregular computation
    irregular distributions
  - parallel I/O: out-of-core arrays
    automatic restructuring and I/O generation
  - D System tools
    program construction tools with performance hints
New Language Organization

• Concern:
  - Slow progress toward highly-optimizing compilers

• Solution
  - Focus on a simple, implementable language, not much different from previous standard (HPF 1.1)
    some changes (HPF 1.2)
    some deletions (e.g. REALIGN and REDISTRIBUTE)
  - Identify strategies for portably efficient programming
  - Specify new or hard-to-implement features as Approved Extensions
    REALIGN and REDISTRIBUTE
    computation partitioning via ON clause
    irregular problem support
    task parallelism
• Rice Fortran 77D
  - emphasis on implicit parallelism
  - extensive deep optimizations
  - interprocedural analysis and optimization
  - generation of Fortran 77 + message passing

• Syracuse Fortran 90D
  - emphasis on explicit parallelism
  - strip mining of array operations
  - scalarization and generation of F77 + communication
  - parallelism in highly-optimized run-time library

• Lesson: both approaches needed for HPF
Fortran D95 Experimental Compiler

Goal: to provide a framework for experimentation with advanced language features and optimizations for High Performance Fortran and Fortran D

• Phases:
  - computation partitioning
    relaxation of owner computes rule
  - communication placement
    use of general placement framework
  - code generation
    Fortran 77 + MPI calls
    specific tailoring to target machine

• Interprocedural analysis and optimization

• D System Class Library
  - abstract syntax tree, dependence graph, etc.
Goals for the Fortran D95 Compiler

• Better optimizations for regular and irregular applications:
  - optimal computation partitioning
  - aggressive communication-computation overlap
  - interprocedural optimization

• Compilation techniques for new architectures:
  - DSM (hardware and software)
  - SMP clusters

• Research on new language features:
  - out-of-core computations
  - advanced data structures:
    - block-structured, dynamic, and pointer-based structures
  - new parallelism models: task parallelism
HPF Optimizations

• **Reducing Communications Overhead**
  - Partitioning to minimize communications while maintaining load balance
  - Control of message granularity and frequency vectorization, coalescing
  - Recognition of collective communication broadcast, shift, global sum

• **Hiding Communications Overhead**
  - Overlap of Communications and Computation loop splitting

• **Storage Management**
  - Strip mining to limit buffer space
Example

!HPF$ TEMPLATE T(1000,1000)
!HPF$ ALIGN A(I,J) WITH T(I,J)
!HPF$ ALIGN B(I,J) WITH T(I,J)
!HPF$ DISTRIBUTE T(BLOCK,BLOCK)

!Assume a 10 by 10 real processor array

DO I = 1, 1000
  DO J = 1, 999
    A(I,J) = A(I,J) + B(I,J+1)
  ENDDO
ENDDO
IF myC ≠ 1 THEN
    SEND B(1:100,1) TO P(myR,myC-1)
ENDIF

IF myC ≠ 10 THEN
    RECEIVE B(1:100,101) FROM P(myR,myC+1)
    DO i = 1, 100
        A(i,100) = A(i,100) + B(i,101)
    ENDDO
ENDIF

DO i = 1, 100
    DO j = 1, 99
        A(i,j) = A(i,j) + B(i,j+1)
    ENDDO
ENDDO
Overlap Area

Local Array Storage for B

Sent to Left

Overlap Area
IF myC ≠ 1 THEN
    SEND B(1:100,1) TO P(myR,myC-1)
ENDIF

DO i = 1, 100
    DO j = 1, 99
        A(i,j) = A(i,j) + B(i,j+1)
    ENDDO
ENDDO

IF myC ≠ 10 THEN
    RECEIVE B(1:100,101) FROM P(myR,myC+1)
    DO i = 1, 100
        A(i,100) = A(i,100) + B(i,101)
    ENDDO
ENDIF
Conclusions: Compiling HPF

• Complex compilation problem
  - standard program analysis
  - propagation of distributions
  - partitioning of loops
  - generation and placement of communication
  - code generation
    - loop splitting
    - storage management

• Interprocedural analysis nearly mandatory
  - distributions
  - communications
  - precision of standard analysis
Programmer Tasks on Irregular Meshes

- **Partitioner:** write code to assign data elements to processors
  - load balance
  - communication minimization

- **Inspector:** write code to determine communications schedule
  - dry run of computation loops to determine communication schedule
  - hopefully, this is done infrequently

- **Executor:** write code to carry out communication during computation
  - optimized by the inspector phase
Fortran D and Irregular Problems

• Partitioner
  - still provided by programmer
  - computes mapping array or function

• Inspector and Executor
  - generated by Fortran D compiler
  - distribution by array

\[
\text{DISTRIBUTE D(INDIRECT(MAP))}
\]
  - system generates communication automatically

• Some partitioners can be automatically generated
  - decomposition by value
Parallel I/O

- What is the Parallel I/O Problem?
  - Out-of-core arrays
  - Checkpointing
  - Real-time I/O

- Extend HPF data distributions to out-of-core arrays

  !SIO$ IO-DISTRIBUTE A(BLOCK, *)

- Extend HPF compilation technology
  - I/O managed implicitly by compiler, run-time system, and I/O system
  - Programmer freed from grubby details of I/O
Out-Of-Core I/O: Sources of Leverage

- **Program Reorganization**
  - techniques developed for memory hierarchy can work for I/O
  - process out-of-core data in "tiles"

- **Explicit I/O Generation**
  - beats less efficient implementations of virtual memory
  - works on systems with NO virtual memory
  - can work with virtual memory system

- **Prefetching**
  - exploits predictable program behavior
  - uses asynchronous I/O to overlap with computation
## Summary of Improvements

<table>
<thead>
<tr>
<th>Source</th>
<th>LU Decomposition</th>
<th>Red-Black Relaxation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reorganization</td>
<td>20,000%</td>
<td>0%</td>
</tr>
<tr>
<td>Explicit I/O</td>
<td>35-40%</td>
<td>38-60%</td>
</tr>
<tr>
<td>Computation- I/O overlap</td>
<td>5-10%</td>
<td>10-27%</td>
</tr>
</tbody>
</table>
The Programming Problem

- Program written at an abstract level
  - Fortran + distribution

- All parallelism and communication generated automatically
  - parallelism determined by data layout
  - communication placement determined by optimization

- **Result:** mysterious relationship between source program and object program performance
  - Need: way to explain compiler actions
  - without forcing user to read the object program
D System

- **Fortran D Compiler**
  - static performance estimates
  - compiler-derived facts
  - mapping information

- **Data Distribution Assistant**
  - data distribution alternatives

- **Whole-Program Analysis**
  - program analysis results

- **D Editor**
  - execution status
  - control
  - performance interpretation

- **D Debugger**
  - execution status
  - control

- **D Performance Tools**
  - data distribution alternatives
D Editor

• Sophisticated Fortran editing
  - mixed text and structure editing
  - incremental recompilation

• Display of deep program analysis
  - dependences

• Feedback from Fortran D compiler
  - cross-processor loops
  - parallel vs pipelined vs serial loops
  - display of communication

• Transformations to improve performance
  - loop interchange, data redistribution, etc.

• Display of Performance Information
Distribute A(BLOCK), B(BLOCK)

DO I = 1, N

A(I) = A(I) + B(I+1)

ENDDO

parallel

generates communication
D分布在A(BLOCK), B(BLOCK)

DO I = 1, N

A(I) = A(I-1) + B(I)

ENDDO
Performance Support in the D System

- **KEY:** Explain performance in terms of source code
- **Compiler support for performance analysis:**
  - Intelligent instrumentation: reduce trace volume
  - Mapping information
  - Automatic tuning of low-level communication parameters
- **Correlating measured performance with source code:**
  - Map message costs to non-local references, arrays
  - Map SPMD loop costs to source loops (preliminary)
  - Compute reference patterns to arrays
- **Performance visualization**
  - Annotations on the source code in the D Editor
Summary

- Machine-independent parallel programming support is essential
- HPF addresses this problem for data parallelism
  - high-level specification of parallelism
- HPF 1 is missing some key ingredients
  - support for irregular problems
  - task parallelism
  - parallel I/O
  - HPF 2 process has begun—target: November 1996
- Powerful tools will be needed
  - program analysis, construction and debugging
  - performance explanation
  - the D System
Future Directions

- **Software Support for Memory Hierarchy**
  - multi-level, parallel hierarchies
  - run-time approaches (e.g., prefetching)

- **Support for Heterogeneous Networks**
  - 1-Way done right
  - functional parallelism
  - standard interfaces (CORBA, etc.)

- **Support for Adaptive Calculations**
  - extensions to irregular support
  - new data structures (library vs language)

- **Implications of Interprocedural Compilation**
  - simplification of interfaces
  - global optimization and code generation
For More Information

- Home Page for CRPC
  
  http://www.crpc.rice.edu/CRPC/

- Home Page for HPF
  
  http://www.crpc.rice.edu/HPFF/home.html

- Home Page for D System
  
  http://www.crpc.rice.edu/fortran-tools/DSystem/DSystem.html