Compilers and Run-Time Systems for High-Performance Computing

Blurring the Distinction between Compile-Time and Run-Time

Ken Kennedy
Center for High Performance Software
Rice University

http://www.cs.rice.edu/~ken/Presentations/CompilerRuntime.pdf
Context

• Explosive Growth of Information Technology
  — Now represents 20 percent of economy, 35 percent of GDP growth
  — Essential to operation of most organizations, especially government
Context

• Explosive Growth of Information Technology
  — Now represents 20 percent of economy, 35 percent of GDP growth
  — Essential to operation of most organizations, especially government

• Enormous Demand for Software

• Shortage of IT Professionals
  — Challenge: double the number of CS graduates
Context

• Explosive Growth of Information Technology
  – Now represents 20 percent of economy, 35 percent of GDP growth
  – Essential to operation of most organizations, especially government

• Enormous Demand for Software

• Shortage of IT Professionals
  – Challenge: double the number of CS graduates

• Complex Computer Architectures
  – Deep memory hierarchies, high degrees of parallelism
  – Heterogeneous, geographically distributed platforms
    - Changes in performance of nodes and links during execution
Context

• Explosive Growth of Information Technology
  — Now represents 20 percent of economy, 35 percent of GDP growth
  — Essential to operation of most organizations, especially government

• Enormous Demand for Software

• Shortage of IT Professionals
  — Challenge: double the number of CS graduates

• Complex Computer Architectures
  — Deep memory hierarchies, high degrees of parallelism
  — Heterogeneous, geographically distributed platforms
    - Changes in performance of nodes and links during execution

• Complex Applications
  — Many diverse components, dynamic, adaptive, unstructured
Philosophy

• **Compiler Technology = Off-Line Processing**
  
  — **Goals: improved performance and language usability**
  
  - Making it practical to use the full power of the language
Philosophy

- **Compiler Technology = Off-Line Processing**
  - **Goals:** improved performance and language usability
    - Making it practical to use the full power of the language
  - **Trade-off:** preprocessing time versus execution time
  - **Rule:** performance of both compiler and application must be acceptable to the end user
Philosophy

• **Compiler Technology = Off-Line Processing**
  - **Goals:** improved performance and language usability
    - Making it practical to use the full power of the language
  - **Trade-off:** preprocessing time versus execution time
  - **Rule:** performance of both compiler and application must be acceptable to the end user

• **Examples**
  - **Macro expansion**
    - PL/I macro facility — 10x improvement with compilation
Philosophy

• **Compiler Technology = Off-Line Processing**
  - Goals: improved performance and language usability
    - Making it practical to use the full power of the language
  - Trade-off: preprocessing time versus execution time
  - Rule: performance of both compiler and application must be acceptable to the end user

• **Examples**
  - Macro expansion
    - PL/I macro facility — 10x improvement with compilation
  - Query processing
    - Dramatic improvement in speed through planning
Philosophy

• **Compiler Technology = Off-Line Processing**
  
  - Goals: improved performance and language usability
  - Making it practical to use the full power of the language
  - Trade-off: preprocessing time versus execution time
  - Rule: performance of both compiler and application must be acceptable to the end user

• **Examples**
  
  - Macro expansion
    - PL/I macro facility — 10x improvement with compilation
  
  - Query processing
    - Dramatic improvement in speed through planning
  
  - Communication planning in dynamic applications
    - Develop efficient communication schedules at run time
It was our belief that if FORTRAN, during its first months, were to translate any reasonable “scientific” source program into an object program only half as fast as its hand-coded counterpart, then acceptance of our system would be in serious danger... I believe that had we failed to produce efficient programs, the widespread use of languages like FORTRAN would have been seriously delayed.

— John Backus
Preliminary Conclusions

• Definition of Application Will Become Fuzzy
  — Knowledge of the computation will be revealed in stages
  — Examples:
    - Compilation with input data,
    - Compiler-generated run-time preprocessing
    - Optimization with late binding of target platform
    - Compilation based on predefined component libraries
Preliminary Conclusions

• Definition of Application Will Become Fuzzy
  — Knowledge of the computation will be revealed in stages
  — Examples:
    - Compilation with input data,
    - Compiler-generated run-time preprocessing
    - Optimization with late binding of target platform
    - Compilation based on predefined component libraries

• Performance Will Be More Elusive
  — Even reliable performance will be hard to achieve
  — Compiler will need to be even more heroic,
    - Yet programmer will continue to want control
Preliminary Conclusions

• **Definition of Application Will Become Fuzzy**
  - Knowledge of the computation will be revealed in stages
  - **Examples:**
    - Compilation with input data,
    - Compiler-generated run-time preprocessing
    - Optimization with late binding of target platform
    - Compilation based on predefined component libraries

• **Performance Will Be More Elusive**
  - Even reliable performance will be hard to achieve
  - Compiler will need to be even more heroic,
    - Yet programmer will continue to want control

• **Compilers Structure Will Be More Flexible**
  - Compilation will be carried out in stages
Compiling with Data

Program -> Compiler -> Application
Compiling with Data

- Program
- Slowly-Changing Data
- Compiler
- Reduced Application
Compiling with Data

Program

Slowly-Changing Data

Rapidly-Changing Data

Compiler

Reduced Application

Answers
Run-Time Compilation

Program → Compiler → Application
Run-Time Compilation

Program

Compiler

Pre-Optimizer

Application

Slowly-Changing Data
Run-Time Compilation

Program

Compiler

Pre-Optimizer

Application

Rapidly-Changing Data

Slowly-Changing Data

Answers
Bandwidth as Limiting Factor

• Program and Machine Balance
  – Program Balance: Average number of bytes that must be transferred in memory per floating point operation
  – Machine Balance: Average number of bytes the machine can transfer from memory per floating point operation
**Bandwidth as Limiting Factor**

- **Program and Machine Balance**
  - **Program Balance:** Average number of bytes that must be transferred in memory per floating point operation
  - **Machine Balance:** Average number of bytes the machine can transfer from memory per floating point operation

<table>
<thead>
<tr>
<th>Applications</th>
<th>Flops</th>
<th>L1–Reg</th>
<th>L2–L1</th>
<th>Mem–L2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Convolution</strong></td>
<td>1</td>
<td>6.4</td>
<td>5.1</td>
<td>5.2</td>
</tr>
<tr>
<td><strong>Dmxpy</strong></td>
<td>1</td>
<td>8.3</td>
<td>8.3</td>
<td>8.4</td>
</tr>
<tr>
<td><strong>Mmjki (o2)</strong></td>
<td>1</td>
<td>24.0</td>
<td>8.2</td>
<td>5.9</td>
</tr>
<tr>
<td><strong>FFT</strong></td>
<td>1</td>
<td>8.3</td>
<td>3.0</td>
<td>2.7</td>
</tr>
<tr>
<td><strong>SP</strong></td>
<td>1</td>
<td>10.8</td>
<td>6.4</td>
<td>4.9</td>
</tr>
<tr>
<td><strong>Sweep3D</strong></td>
<td>1</td>
<td>15.0</td>
<td>9.1</td>
<td>7.8</td>
</tr>
<tr>
<td><strong>SGI Origin</strong></td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>0.8</td>
</tr>
</tbody>
</table>
Cache and Bandwidth
Cache and Bandwidth

L2 Cache 128 Bytes

6.25 % Utilization

Memory

Center for High Performance Software
Cache and Bandwidth

L1 Cache 32 Bytes
25% Utilization

L2 Cache 128 Bytes
6.25% Utilization

Memory
Cache and Bandwidth

- Register 8 Bytes: 100% Utilization
- L1 Cache 32 Bytes: 25% Utilization
- L2 Cache 128 Bytes: 6.25% Utilization

Center for High Performance Software
Dynamic Data Packing

• Suppose the Calculation is Irregular
  — Example: Molecular Dynamics
    - Force calculations (pairs of forces)
    - Updating locations (single force per update)
Dynamic Data Packing

• Suppose the Calculation is Irregular
  — Example: Molecular Dynamics
    - Force calculations (pairs of forces)
    - Updating locations (single force per update)

• Strategy
  — Dynamically reorganize data
    - So locations used together are updated together
  — Dynamically reorganize interactions
    - So indirect accesses are not needed
  — Example: “first touch”
    - Assign elements to cache lines in order of first touch by pairs calculation
First-Touch Ordering

Original Ordering

P_5  P_1  P_4  P_3  P_2
First-Touch Ordering

Original Ordering

Interaction Pairs
First-Touch Ordering

Original Ordering

P5
P1
P4
P3
P2

Interaction Pairs

P1
P1
P1
P2
P2

P2
P3
P4
P3
P5

First-Touch Ordering

P1
P2
P3
P4
P5
Performance Results 2

Magi

- Exe. time
- L1 misses
- L2 misses
- TLB misses

- original
- data regrouping
- base packing
- opt packing
Irregular Multilevel Blocking

• Associate a tuple of block numbers with each particle
  – One block number per level of the memory hierarchy
  – Block number = selected bits of particle address

Particle address

A

B

C

L2 block number

TLB block number

L1 block number
Irregular Multilevel Blocking

- Associate a tuple of block numbers with each particle
  - One block number per level of the memory hierarchy
    - Block number = selected bits of particle address

<table>
<thead>
<tr>
<th>Particle address</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
</table>

- L2 block number
- TLB block number
- L1 block number

- For an interaction pair, interleave block numbers for particles

| A | A | B | B | C | C |

- Sorting by composite block number $\rightarrow$ multi-level blocking
Dynamic Optimization

Program → Compiler → Application
Dynamic Optimization

Program

Compiler

Configuration And Data

Dynamic Optimizer (Optimizing Loader)

Application
Dynamic Optimization

Program → Compiler → Dynamic Optimizer (Optimizing Loader) → Application → Answers

Configuration And Data → Dynamic Optimizer (Optimizing Loader)

Rapidly- Changing Data → Application
A Software Grand Challenge

• Application Development and Performance Management for Grids
  — Problems:
    - Reliable performance on heterogeneous platforms
    - Varying load
    
    On computation nodes and on communications links
  
  • Challenges:
    — Presenting a high-level programming interface
      - If programming is hard, its useless
    — Designing applications for adaptability
    — Mapping applications to dynamically changing architectures
    — Determining when to interrupt execution and remap
      - Application monitors
      - Performance estimators
National Distributed Computing
National Distributed Computing
National Distributed Computing

Supercomputer

Database
National Distributed Computing
National Distributed Computing

Supercomputer

Database

Supercomputer
What Is a Grid?

• Collection of computing resources
  — Varying in power or architecture
  — Potentially dynamically varying in load
    - Unreliable?
  — No hardware shared memory

• Interconnected by network
  — Links may vary in bandwidth
  — Load may vary dynamically

• Distribution
  — Across room, campus, state, nation, globe

• Inclusiveness
  — Distributed-memory parallel computer is a degenerate case
Globus

- Developed by Ian Foster and Carl Kesselman
  - Originally to support the I-Way (SC-96)

- Basic Services for distributed computing
  - Accounting
  - Resource directory
  - User authentication
  - Job initiation
  - Communication services (Nexus and MPI)

- Applications are programmed by hand
  - User responsible for resource mapping and all communication
  - Many applications, most developed with Globus team
    - Even Globus developers acknowledge how hard this is
What is Needed

- Compiler and language support for **reliable performance**
  - dynamic reconfiguration, optimization for distributed targets
What is Needed

• Compiler and language support for reliable performance
  —dynamic reconfiguration, optimization for distributed targets

• Abstract Grid programming models
  —design of an implementation strategy for those models
What is Needed

• Compiler and language support for reliable performance
  — dynamic reconfiguration, optimization for distributed targets

• Abstract Grid programming models
  — design of an implementation strategy for those models

• Easy-to-use programming interfaces
  — problem-solving environments
What is Needed

• Compiler and language support for **reliable performance**
  — dynamic reconfiguration, optimization for distributed targets

• Abstract Grid programming models
  — design of an implementation strategy for those models

• Easy-to-use programming interfaces
  — problem-solving environments

• Robust reliable numerical and data-structure libraries
  — predictability and robustness of accuracy and performance
  — reproducibility, fault tolerance, and auditability
What is Needed

• Compiler and language support for **reliable performance**
  — dynamic reconfiguration, optimization for distributed targets

• Abstract Grid programming models
  — design of an implementation strategy for those models

• Easy-to-use programming interfaces
  — problem-solving environments

• Robust reliable numerical and data-structure libraries
  — predictability and robustness of accuracy and performance
  — reproducibility, fault tolerance, and auditability

• Performance monitoring and control strategies
  — deep integration across compilers, tools, and runtime systems
  — performance contracts and dynamic reconfiguration
Programming Models

• Distributed Collection of Objects (for serious experts)
  — message passing for communication

• Problem-Solving Environment (for non-experts)
  — packaged components
  — graphical or scripting language for glue

• Distribution of Shared-Memory Programs (for experts)
  — language-based decomposition specification from programmer
  — parametrizable for reconfiguration
    - example: reconfigurable distributed arrays (DAGH)
  — implemented as distributed object collection
  — implicit or explicit communications
Grid Compilation Architecture

- **Goal:** reliable performance under varying load

GrADS Project (NSF NGS): Berman, Chien, Cooper, Dongarra, Foster, Gannon, Johnsson, Kennedy, Kesselman, Reed, Torczon, Wolski

Center for High Performance Software
Grid Compilation Architecture

Execution Environment

Source Application
Whole-Program Compiler
Configurable Object Program
Service Negotiator
Scheduler
Real-time Performance Monitor
Dynamic Optimizer
Grid Runtime System

Software Components
Performance Feedback
Performance Problem
Negotiation
Configurable Object Program
Libraries
PSE

Center for High Performance Software
Performance Contracts

• At the Heart of the GrADS Model
  – Fundamental mechanism for managing mapping and execution

• What are they?
  – Mappings from resources to performance
  – Mechanisms for determining when to interrupt and reschedule

• Abstract Definition
  – Random Variable: \( \rho(A,I,C,t_0) \) with a probability distribution
    - \( A = \text{app}, I = \text{input}, C = \text{configuration}, t_0 = \text{time of initiation} \)
    - Important statistics: lower and upper bounds (95% confidence)
  – Issue:
    - Is \( \rho \) a derivative at \( t_0 \)? (Wolski)
Grid Compilation Architecture

Program Preparation System

- Source Application
- Whole-Program Compiler
- Configurable Object Program
- Performance Feedback
- Software Components

Execution Environment

- Real-time Performance Monitor
- Grid Runtime System
- Negotiation
- Dynamic Optimizer

- Service Negotiator
- Scheduler

Performance Problem

Source Application

Whole-Program Compiler

Configurable Object Program

Software Components

Performance Feedback

Dynamic Optimizer

Grid Runtime System

Real-time Performance Monitor

Negotiation

Service Negotiator

Scheduler
• Representation of the Application
  — Supporting dynamic reconfiguration and optimization for distributed targets
  — Includes
    - Program intermediate code
    - Annotations from the compiler

• Reconfiguration strategies
  — Aggregation of data regions (submeshes)
  — Aggregation of tasks
  — Definition of parameters
    - Used for algorithm selection
GrADS Testbeds

• MicroGrid (Andrew Chien)
  — Cluster of Intel PCs
  — Runs standard Grid software (Globus, Nexus)
  — Permits simulation of varying loads
    - Network and processor
  — Extensive performance modeling

• MacroGrid (Carl Kesselman)
  — Collection of processors running Globus
  — At all 8 GrADS sites
  — Permits experimentation with real applications
    - Cactus (Ed Seidel)
Research Strategy

• **Begin Modestly**
  - application experience to identify opportunities
  - prototype reconfiguration system
    - with performance monitoring, without dynamic optimization
  - prototype reconfigurable library

• **Move from Simple to Complex Systems**
  - begin with heterogeneous clusters
  - refinements of reconfiguration system and performance contract mechanism
  - use an artificial testbed to test performance under varying conditions

• **Experiment with Real Applications**
  - someone cares about the answers
Programming Productivity

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

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

• One Strategy: Make the End User a Programmer
  — professional programmers develop components
  — users integrate components using:
    - problem-solving environments (PSEs)
    - scripting languages (possibly graphical)
      examples: Visual Basic, Tcl/Tk, AVS, Khoros
Programming Productivity

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

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

• **Compilation for High Performance**
  - translate scripts and components to common intermediate language
  - optimize the resulting program using interprocedural methods
Telescoping Languages

$L_1$ Class Library
Telescoping Languages

- L₁ Class Library
- Compiler Generator
- L₁ Compiler

Could run for hours
Telescoping Languages

L₁ Class Library → Compiler Generator

Script → Script Translator → L₁ Compiler

Could run for hours
understands library calls as primitives

Vendor Compiler → Optimized Application

Optimized Application

Center for High Performance Software
Telescoping Languages: Advantages

• Compile times can be reasonable
  — More compilation time can be spent on libraries
    - Amortized over many uses
  — Script compilations can be fast
    - Components reused from scripts may be included in libraries

• High-level optimizations can be included
  — Based on specifications of the library designer
    - Properties often cannot be determined by compilers
    - Properties may be hidden after low-level code generation

• User retains substantive control over language performance
  — Mature code can be built into a library and incorporated into language
Example: HPF Revisited

- Fortran 90 Program
- HPF Translator
- Global Optimizer
- MPI Code Generator
Example: HPF Revisited

Distribution Library → Distribution Precompiler → HPF Translator → Global Optimizer → MPI Code Generator

Fortran 90 Program → HPF Translator → Global Optimizer
Distribute (HilbertLib): A, B
Do i = 1, 100
   A(i) = B(i) + C
Enddo

A.putBlock(1, 100, B.getBlock(1, 100) + C)
Flexible Compiler Architecture

• Flexible Definition of Computation
  — Parameters
    - program scheme
    - base library sequence (l1, l2, ..., lp)
    - subprogram source files (s1, s2, ..., sn)
    - run history (r1, r2, ..., rk)
    - data sets (d1, d2, ..., dm)
    - target configuration
Flexible Compiler Architecture

• Flexible Definition of Computation
  — Parameters
    - program scheme
    - base library sequence \((l_1, l_2, \ldots, l_p)\)
    - 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
  — several compilation steps as information becomes available
Flexible Compiler Architecture

- Flexible Definition of Computation
  - Parameters
    - program scheme
    - base library sequence (l1, l2, ..., lp)
    - subprogram source files (s1, s2, ..., sn)
    - run history (r1, r2, ..., rk)
    - data sets (d1, d2, ..., dm)
    - target configuration

- Compilation = Partial Evaluation
  - several compilation steps as information becomes available

- Program Management
  - When to back out of previous compilation decisions due to change
  - When to invalidate certain inputs
    - Examples: change in library or run history
Summary

• Target Platforms, Languages, and Apps Becoming More Complex
  — Platforms: Parallel, heterogeneous, deep memory hierarchies
  — Applications: dynamic, irregular, extensive use of domain libraries
  — Programming: component development, system composition

• Example: The Grid as a Problem-Solving System
  — Seamless integration of access to remote resources
  — Programming support infrastructure is a challenge
    - Execution of applications must be adaptive
      Must manage execution to reliable completion
    - Ideally, should support high-level domain-specific programming

• Compiler Structure Will Be Correspondingly Complex
  — Partial evaluation in stages with incremental information

Center for High Performance Software