Software and High Performance Computing: Challenges for Research

The Implications of PITAC

Ken Kennedy
Center for High Performance Software
Rice University

http://www.cs.rice.edu/~ken/Presentations/IBMAustinChallenges.pdf
Part I: The Role of the Federal Government in Information Technology

The PITAC Experience
Question

• Should the Federal Government Have Any Role?
  — Other than ensuring that commerce flows freely

• Answer:
  — The Federal Government is the only effective source of funding for long-term fundamental research in high-technology areas.
  — Federal government must also fund research in short-term applied research in areas of importance to the government but marginal commercial value

• Is the Federal Government Doing This Well?
  — This is the question that the President’s Information Technology Advisory Committee (PITAC) was asked to address.
PITAC Charter

• The Committee shall provide an independent assessment of:
  
  — Progress made in implementing the High-Performance Computing and Communications (HPCC) Program;
  
  — Progress in designing and implementing the Next Generation Internet initiative;
  
  — The need to revise the HPCC Program;
  
  — Balance among components of the HPCC Program;
  
  — Whether the research and development undertaken pursuant to the HPCC Program is helping to maintain United States leadership in advanced computing and communications technologies and their applications;
  
  — Other issues as specified by the Director of the Office of Science and Technology.
  
  - Review of the entire IT investment strategy — is it meeting the nation's needs
PITAC Membership 97–99

• Co-Chairs:
  — Bill Joy, Sun Microsystems
  — Ken Kennedy, Rice

• Members:
  — Eric Benhamou, 3Com
  — Ching-chih Chen, Simmons
  — Steve Dorfman, Hughes
  — Bob Ewald, SGI
  — Sherri Fuller, U of Washington
  — Susan Graham, UC Berkeley
  — Danny Hillis, Disney, Inc
  — John Miller, Montana State
  — Raj Reddy, Carnegie Mellon
  — Larry Smarr, UIUC
  — Les Vadasz, Intel
  — Steve Wallach, Centerpoint
  — Vinton Cerf, MCI
  — David Cooper, LLNL
  — David Dorman, AT&T
  — David Farber, Penn
  — Hector Garcia-Molina, Stanford
  — Jim Gray, Microsoft
  — Robert Kahn, CNRI
  — David Nagel, AT&T
  — Ted Shortliffe, Columbia
  — Joe Thompson, Miss. State
  — Andy Viterbi, Qualcomm
  — Irving Wladawsky-Berger, IBM
Methodology

• Evaluation of Federal Research Investment Portfolio
  — Plans reviewed for each of the major areas:
    - High End Computing and Computation
    - Large Scale Networking
    - Human Centered Computer Systems
    - High Confidence Systems
    - Education, Training, and Human Resources

• Review of Balance in Federal Research Portfolio
  — Fundamental versus Applied
    - Based on our own definition of these terms
  — High-Risk versus Low-Risk
  — Long-Term versus Short-Term
Principal Finding

• Drift Away from Long-Term Fundamental Research
  — Agencies pressed by the growth of IT needs
    - IT R&D budgets have grown steadily but not dramatically
    - IT industry has accounted for over 30 percent of the real GDP growth over the past five years, but gets only 1 out of 75 Federal R&D dollars
    - Problems solved by IT are critical to the nation—engineering design, health and medicine, defense
  — Most IT R&D agencies are mission-oriented
    - Natural and correct to favor the short-term needs of the mission

• This Trend Must Be Reversed
  — Continue the flow of ideas to fuel the information economy and society
Remedy

• Increase the Federal IT R&D Investment by 1.4 billion dollars per year
  — Ramp up over five years
  — Focus on increasing fundamental research

• Invest in Key Areas Needing Attention
  — Software
  — Scalable Information Infrastructure
  — High-End Computing
  — Social, Economic, and Workforce Issues

• Develop a Coherent Management Strategy
  — Establish clear organizational responsibilities
  — Diversify modes of support
Software

• Recommendations
  — Make fundamental software research an absolute priority
  — Invest in key area needing attention
    - Improving programmer productivity
      Ameliorate the shortage of IT professionals
    - Improving reliability and robustness of software
    - Improving usability through human interface innovations
    - Improving capabilities for information management
  — Make software research a substantive component of every major information technology research initiative.
Scalable Information Infrastructure

- Research Needed:
  - Understanding the behavior of the global-scale network.
  - Physics of the network, including optical and wireless technologies such as satellites, and bandwidth issues.
  - Scalability of the Internet.
  - Information management, Information and services survivability
  - Large-scale applications and the scalable services they require.
    - National digital library, Next-generation world-wide web
  - Fund a balanced set of testbeds that serve the needs of networking research, research in enabling information technologies and advanced applications, and Internet research.
High-End Computing

• Findings:
  – High-end computing is essential for science and engineering research
  – High-end computing is an enabling element of the United States national security program
  – New applications of high-end computing are ripe for exploration
  – Suppliers of high-end systems suffer from difficult market pressures
    - High-end market not large
  – Advances in high-end computing eventually find their way to desktop

• Recommendation
  – Fund high-end computing research (architecture, software, and applications, and testbeds) because it is important to the government and the health, welfare, and security of the population
Social, Economic, Workforce Issues

- **Invest in Four Areas:**
  - IT-literate population
  - IT workforce
    - More workers, more underrepresented groups
  - Use of IT in education
  - Understanding economic and policy implications of technology

- **An Observation on IT Workforce**
  - Research investment in universities is critical
    - Without it, faculty leave
    - Without it, grad students do not go → no new faculty
    - Without faculty, we cannot produce more BS graduates
Good News

• Administration Budget
  – Proposed additional $366 million in FY 2000
    - Appropriated: $226 million
  – Proposed $605 million increase for FY 2001
  – Successive years unclear

• Congress
  – Sensenbrenner NITR&D Act from House Science Committee
    - 5 years of funding at PITAC-recommended levels
    - Permanent R&D investment tax credit
    - Passed with near-unanimous support
    - Only partially reflected in the Senate authorization bills
  – Appropriations are year-to-year
Questions

• *Can we increase long-term research by rebudgeting?*
  — No, because the short-term work addresses essential problems

• *Why doesn’t industry fund this?*
  — Industry research focused on product development
  — Enormously expensive
  — Thurow:
    — Private rate of return on research — 24%
    — Societal rate of return on research — 66%
  — Industry is not good at funding and developing disruptive technologies
  — Federal Government funding creates fuel for the venture capital system
IT Grand Challenges

• Software Reliability
  — Who will pay for bug free, feature-poor software?

• Internet Scalability and Security
  — What happens when we have 2 billion internet connections at DSL speed?

• Realistic Videoconferencing
  — Can we put airlines out of business?

• The Internet as Problem-Solving Engine*
  — GrADS Project

• Software Productivity*
  — Workforce shortage
  — Idea: make it possible for end users to be application developers
Conclusions

• U. S. leadership in Information Technology research provides an essential foundation for commerce, education, health care, environmental stewardship, and national security in the 21st century.
  — Dramatically transform the way we communicate, learn, deal with information and conduct research
  — Transform the nature of work, nature of commerce, product design cycle, practice of health care, and the government itself

• Increased investment in long-term research is needed to continue the flow of ideas and people from universities into the IT industry.
  — Revitalize university research (and education) in IT

• This is a unique responsibility of the Federal Government
  — Companies must focus on short term product development
Part II: Compiler Technology for Problem Solving on Computational Grids

The GrADS Project: Toward the Internet as a Problem-Solving System
National Distributed Computing

Database

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
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
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

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

• Development of 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
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

Execution Environment

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

Libraries
Configurable Object Program

• Representation of the Application
  – Supporting dynamic reconfiguration and optimization for distributed targets
  – Includes
    - Program intermediate code
    - Annotations from the compiler
      Reconfiguration strategy
    - Historical information (run profile to now)

• Reconfiguration strategies
  – Aggregation of data regions (submeshes)
  – Aggregation of tasks
  – Definition of parameters
    - Used for algorithm selection
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
Summary

• The scalable infrastructure should be a scalable problem-solver
  — Access to information is not enough
  — Linked computation is not enough

• Infrastructure is complex
  — Seamless integration of access to remote resources in response to need

• Execution of applications must be adaptive
  — In response to changing loads and resources

• Programming support is hard
  — Must manage execution to reliable completion
  — Must prepare a program for execution
  — Ideally, should support high-level domain-specific programming
    – Telescoping languages
Part III: Compiler Architecture for High-Performance Problem Solving

A Quest for High-Level Programming Systems
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
  - 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
  - Database query optimization
  - Emulation acceleration
    - TransMeta “code morphing”
  - Communication planning in dynamic applications (Inspector/Executor)
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
A Java Experiment

• **Scientific Programming In Java**
  
  — *Goal*: make it possible to use the full object-oriented power for scientific applications
  
  - Many scientific implementations mimic Fortran style

• **OwlPack Benchmark Suite**
  
  — Three versions of LinPACK in Java
    
    - Fortran style
    - Lite object-oriented style
    - Full polymorphism
  
  No differences for type

• **Experiment**
  
  — Compare running times for different styles on same Java VM
  
  — Evaluate potential for compiler optimization
Performance Results

Run Time in Secs

Results Using JDK 1.2 JIT on SUN Ultra 5

Fortran Style
Lite OO Style
OO Style
Optimized OO
Native F90

dgefa  dgesl  dgedi
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
- Slowly-Changing Data
- Rapidly-Changing Data
- Compiler
- Reduced Application
- Answers
Run-Time Compilation

- Program
- Slowly-Changing Data
- Rapidly-Changing Data
- Compiler
- Pre-Optimizer
- Application
- 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

<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>Convolution</td>
<td>1</td>
<td>6.4</td>
<td>5.1</td>
<td>5.2</td>
</tr>
<tr>
<td>Dmxpy</td>
<td>1</td>
<td>8.3</td>
<td>8.3</td>
<td>8.4</td>
</tr>
<tr>
<td>Mmjki (o2)</td>
<td>1</td>
<td>24.0</td>
<td>8.2</td>
<td>5.9</td>
</tr>
<tr>
<td>FFT</td>
<td>1</td>
<td>8.3</td>
<td>3.0</td>
<td>2.7</td>
</tr>
<tr>
<td>SP</td>
<td>1</td>
<td>10.8</td>
<td>6.4</td>
<td>4.9</td>
</tr>
<tr>
<td>Sweep3D</td>
<td>1</td>
<td>15.0</td>
<td>9.1</td>
<td>7.8</td>
</tr>
<tr>
<td>SGI Origin</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>0.8</td>
</tr>
</tbody>
</table>
Cache and Bandwidth

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

Memory
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

Interaction Pairs

First-Touch Ordering

Center for High Performance Software
Performance Results 1

Moldyn

- Original
- Data regrouping
- Base packing
- Opt packing

Bar chart showing:
- Execution time
- L1 misses
- L2 misses
- TLB misses
Performance Results 2

Magi

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

Bars represent:
- Original
- Data regrouping
- Base packing
- Opt packing
Dynamic Optimization

Program \(\rightarrow\) Compiler

Configuration And Data \(\rightarrow\) Dynamic Optimizer (Optimizing Loader)

Rapidly-Changing Data \(\rightarrow\) Application

Answers
Grid Compilation Architecture

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

---

**GrADS Project**: Berman, Chien, Cooper, Dongarra, Foster, Gannon, Johnsson, Kennedy, Kesselman, Reed, Torczon, Wolski

---

*Center for High Performance Software*
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
Script-Based Programming

Component Library

User Library

Script

Translator

Global Optimizer

Intermediate Code

Code Generator

Problem: long compilation times, even for short scripts!
Telescoping Languages

- L₁ Class Library
- Compiler Generator
- L₁ Compiler
- Script
- Script Translator
- Vendor Compiler
- Optimized Application

Could run for hours
understands library calls as primitives
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
Applications

• Matlab Compiler
  – Automatically generated from LAPACK or ScaLAPACK
    - With help via annotations from the designer

• Flexible Data Distributions
  – Failing of HPF: inflexible distributions
  – Data distribution == collection of interfaces that meet specs
  – Compiler applies standard transformations

• Automatic Generation of POOMA
  – Data structure library implemented via template expansion in C++
  – Long compile times, missed optimizations

• Generator for Grid Computations
  – GrADS: automatic generation of NetSolve
Requirements of Script Compilation

• Scripts must generate efficient programs
  — Comparable to those generated from standard interprocedural methods
  — Avoid need to recode in standard language

• Script compile times should be proportional to length of script
  — Not a function of the complexity of the library
  — Principle of “least astonishment”
Telescoping Languages

understands library calls as primitives
Script Compilation Algorithm

• Propagate variable property information throughout the program
  — Use jump functions to propagate through calls to library

• Apply high-level transformations
  — Driven by information about properties
  — Ensure that process applies to expanded code

• Perform low-level code specialization
  — At each call site, determine the best estimate to parameter properties that is reflected by a specialized fragment in the code database
    — Use a method similar to “unification”
  — Substitute fragment from database for call
    — This could contain a call to a lower-level library routine.
Telescoping Languages

L₁ Class Library → Compiler Generator

Could run for hours → L₁ Compiler
Library Analysis and Preparation

• Discovery of and Propagation Critical Properties

• Analysis of Transformation Specifications

• Code specialization for different sets of parameter properties
Library Analysis and Preparation

• Discovery and Propagation of Critical Properties
  — Which properties of parameters affect optimization
    - Examples: value, type, rank and size of matrix
Examining the Code

• Example from LAPACK

```fortran
subroutine VMP(C, A, B, m, n, s)
    integer m, n, s; real A(n), B(n), C(m)
    i = 1
    do j = 1, n
        C(i) = C(i) + A(j)*B(j)
        i = i + s
    enddo
end VMP
```

Vectorizable if s != 0
Library Analysis and Preparation

• Discovery and Propagation of Critical Properties
  — Which properties of parameters affect optimization
    - Examples: value, type, rank and size of matrix
  — Construction of jump functions for the library calls
    - With respect to critical properties
Propagation of Properties

• Jump Functions (Transfer Functions)
  — Tell the effect on properties of a call to a library routine
    - Whether it preserves the property or changes it
  — Computed during library preparation (compiler generation) phase
    - Can use lots of time
  — Can be designed to be fast
    - Tradeoff between accuracy and performance

• Advantage of Jump Functions
  — Avoid necessity for analysis of library source
    - Prevent blowup of compilation times
Library Analysis and Preparation

- Discovery and Propagation of Critical Properties
  - Which properties of parameters affect optimization
    - Examples: value, type, rank and size of matrix
  - Construction of jump functions for the library calls
    - With respect to critical properties

- Analysis of Transformation Specifications
  - Construction of a specification-driven translator for use in compiling scripts
High-level Identities

- Often library developer knows high-level identities
  - Difficult for the compiler to discern
  - Optimization should be performed on sequences of calls rather than code remaining after expansion
- Example: Push and Pop
  - Designer Push(x) followed by y = Pop() becomes y = x
    - Ignore possibility of overflow in Push
- Example: Trigonometric Functions
  - Sin and Cos used in same loop—both computed using expensive calls to the trig library
  - Recognize that cos(x) and sin(x) can be computed by a single call to sincos(x,s,c) in a little more than the time required for sin(x).
Contextual Expansions

- Out of Core Arrays
  - Operations Get(I,J) and GetRow(I,Lo,N)

- Get in a loop
  ```
  Do I
    Do J
      ... Get(I,J)
    Enddo
  Enddo
  ```

- When can we vectorize?
  - Turn into GetRow
  - Answer: if Get is not involved in a recurrence.
    - How can we know?
Library Analysis and Preparation

- Discovery and Propagation of Critical Properties
  - Which properties of parameters affect optimization
    - Examples: value, type, rank and size of matrix
  - Construction of jump functions for the library calls
    - With respect to critical properties

- Analysis of Transformation Specifications
  - Construction of a specification-driven translator for use in compiling scripts

- Code specialization for different sets of parameter properties
  - For each set, assume and optimize to produce specialized code
Code Selection Example

• Library compiler develops inlining tables

```fortran
subroutine VMP(C, A, B, m, n, s)
  integer m, n, s; real A(n), B(n), C(m)
  i = 1
  do j = 1, n
    C(i) = C(i) + A(j)*B(j)
    i = i + s
  enddo
end VMP
```

Inlining Table

```
case on s:
  ==0:  C(1) = C(1) + sum(A(1:n)*B(1:n))
  !=0:  C(1:n:s) = C(1:n:s) + A(1:n)*B(1:n)
  default: call ! VMP(C, A, B, m, n, s)
```

vector
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

- **Optimization Enables Language Power**
  - Principle: encourage rather than discourage use of powerful features
    - Good programming practice should be rewarded
- **Target Platforms, Languages, and Apps Becoming More Complex**
  - Platforms: Parallel, heterogeneous, deep memory hierarchies
  - Applications: dynamic, irregular, extensive use of domain libraries
- **Programming support is hard**
  - Ideally, should support high-level domain-specific programming
    - Telescoping languages
- **Compiler Structure Will Be Correspondingly Complex**
  - Partial evaluation in stages with incremental information

[http://www.cs.rice.edu/~ken/Presentations/IBMAustinChallenges.pdf](http://www.cs.rice.edu/~ken/Presentations/IBMAustinChallenges.pdf)