Tools for Portable HPC Performance: Are We Addressing the Right Issues?

Cherri Pancake
pancake@nacse.org
Northwest Alliance for Computational Science & Engineering
Oregon State University

Portability Is Critical

- Programming to one platform isn’t practical
  - Hardware lifespan << software development cycle
- Increasing need to
  - run codes at multiple sites
  - use computational grids
  - re-use code to form more complex applications
- Increasing interest in heterogeneous applications
How Portability Affects Usability

- Portability extends useful lifetime of software
- Also affects application developer
  - Must deal with multiple, moving targets

"Each new version of each part of the development environment introduces some new, though usually justifiable, quirk"
(Mike Frese, NumerEx)

The Fundamental Nature of Portable Code

- Portable code is always under revision

“I’ve got it, too, Omar... A strange feeling, like we’ve just been going in circles.”
1. Supporting Code Portability

**Observation**
Applications need to run on many platforms (sometimes heterogeneous or distributed)

**Tool Tradeoffs #1**
Tools run on - but not across - multiple platforms

**Portability: What Today’s Tools Do**

Increasingly, tool back ends are targeted to multiple platforms, e.g.
- Vampir (Pallas)
- DEEP (Pacific Sierra)
- Paradyn (Univ Wisconsin)
- Jumpshot (Argonne Nat’l Lab)
Portability: What Users Really Need

- GUI needs to run on all desktop platforms
  - UNIX/LINUX
  - Wintel
  - Perhaps also Mac, Web browser
  - Example of a tool that does: Jumpshot (Argonne Nat’l Lab)
- Back end needs to support heterogeneous applications
  - Simply running on different platforms isn’t enough for Grid applications

NACSE - Oregon State University

2. Supporting Performance Variability

Observation
Portable code tends to “shake out” performance quirks

Tool Tradeoffs #2
Tools present (raw) data, but not (assimilated) information

NACSE - Oregon State University
Frese’s “Rule of Thumb” for Portable Code

- A portable code works
  - until the new processor boards are installed
  - until the parallel environment changes
  - until the shared library changes
  - until the next system upgrade
  - until the system reconfiguration
  - until the next reboot
  - until the system load changes
  - until the next phase of the moon

- And ... correcting for platform-specific quirks lessens portability!

Variability: What Today’s Tools Do

- Lists of “top resource-users” are primary way to identify where to concentrate improvement efforts
Variability: What Users Really Need

- Tools should be able to perform more sophisticated analyses

- S-check uses sensitivity analysis to show where user efforts are most likely to pay off

S-check (NIST)

Variability: What Users Really Need

- Tools should be able to point the user to a solution

- Merlin maps out what an “expert” would try if he/she had those results

Merlin (Purdue Univ.)
3. Supporting Code Complexity

**Observation**

Portable applications are typically complex, involving semi-independent components

**Tool Tradeoffs**

- Tools support only limited code scale/complexity

---

**Example: A Typical Portable Application**

- **Mach3** (magneto-hydrodynamics)
- Professionally ported
  - by NumerEx, for Dept. of Defense HPC Program
- Composition
  - A few tens of thousands of lines of source
  - A few tens of include files
  - A few hundred subroutines (one per source file, each dependent on a few include files)
  - A few custom component libraries
  - Multilanguage (Fortran, C)
**Complexity: What Today's Tools Do**

- The user must sift through massive amounts of data to extrapolate useful information

![xprofiler (IBM) AIMS (NASA Ames)](image)

**Complexity: What Users Really Need**

- Tools should give high-level abstractions & summaries
- DEEP's “whole program” view offers an overview -- and facilitates navigation

![DEEP (Pacific Sierra Research)](image)
Complexity: What Users Really Need

- Tools should present data at multiple levels of detail
- Jumpshot’s “preview” lets user navigate quickly through time

Jumpshot (Argonne Nat’l Lab)

Complexity: What Users Really Need

- Tools should be able to identify meaningful patterns from the raw data
- Paradyn recognizes some patterns of memory access

Paradyn (Univ. Wisconsin)
Complexity: What Users Really Need

- Tools should support high-level operations on user-defined states
  - Ability to structure and view states hierarchically
  - Ability to filter the timeline -- so only selected states show
  - Ability to search timeline for certain conditions (e.g., stateA>10 ms)

4. Supporting Suite Analysis

**Observation**

Testing has to be more comprehensive for portable applications

**Tool Tradeoffs** #4

Tools only present data on one program run at a time
Suite Analysis: What Today's Tools Do

- User must invoke multiple copies of the tool -- manipulating them independently

Jumpshot (Argonne Nat'l Lab)

Suite Analysis: What Users Really Need

- Tools should be able to compare and cross-analyze multiple executions
- Guard shows differences in 2 value sets (filters allow for precision differences, boundary thresholds, etc.)

Guard (Griffith Univ.)
Suite Analysis: What Users Really Need

- Tools should make data available for analysis by other tools -- or user's own analysis.

- Ursa Minor exports its data to statistical tools, spreadsheets, databases, etc.

Ursa Minor (Purdue Univ.)

Conclusions

- HP²C is what today's programmers need.

- Current tools have made tradeoffs that assume platform-specificity and stable codes.

Tool tradeoffs

Portability

Performance
Conclusions

- Not enough research projects are addressing the real challenges for users
  - Increase in platform heterogeneity
  - Need to minimize platform-specific improvements
  - Increasingly hierarchical complexity of applications
  - Need to regression-test improvements across suites of machines