Tool Futures:
A Look Back and a Look Forward

Barton P. Miller
University of Wisconsin
bart@cs.wisc.edu

(with thanks to John Mellor-Crummey and Jeff Vetter

Background

- "Recent survey of distributed memory system users
  - "80% have never even tried to use the parallel debugger
  - "90% still rely primarily on hand-coded instrumentation."

How recent?

October 1992,

Supercomputing Debugging Workshop

Dallas, TX

Outline

- Technical Challenges:
  - Old
  - New
  - Other
- Enablers
- Recommendations from me
- Recommendation from CScADS workshop

NB: These comments are my own and likely biased. I look forward to your input.

Approaching the Technical Challenges

We are here because:

- We have important and difficult problems to solve.
- We are motivated to produce effective solutions: we have pride in what we do.

We will have to work differently than we have in the past to make progress that keeps up with changes in technology.

We will have to leave our egos at the door.

Technical Challenges: Old

- Scale: We have been fighting size for many years:
  - Number: nodes/processors/cores.
  - Code size: e.g. 250MB executables
  - Run length: short runs may be representative, but often are not.
  - Other resources: communicators, file descriptors, etc.
- Detail: Sometimes there is no substitute for precise information, which is expensive and can be hard to obtain.
- Multiple platforms: Systems differ in OS, processor, compilers, interconnect, I/O system, scheduler, MPI, OpenMP, and libraries.
- Parallel languages: We fought with HPF and now UPC (and it's cousins). The greater the semantic gap between the language and its machine code, the greater the tool challenge.
- Automation: I was working on this in my dissertation (1984) and I'm still work on it…sigh. (As are many of you.)

Technical Challenges: New(er)

- Deepening memory hierarchies:
  Difference between having something local vs. not local is getting much worse. Scale exacerbates it.

- Heterogeneity:
  - Mixed mode in one platform: SMP (CMP) nodes, threads, and message-passing between nodes. CMP's are getting bigger. Roadrunner is a cluster with Cells; Tokyo has cluster with Clearspeed processors.
  - Mixed nodes: Cascade will have scalar, CMP, vector nodes.

  We can't solve the problem well with one mode, so let's add more!?

- Extreme scale: Machine sizes are growing at a bewildering rate. Nobody bats an eye when they say 1 million processors. Eek!

Technical Challenges: Other Things

- New programming languages: Matlab, Excel, other specialized descriptive languages.
- And more automation: Apply advanced statistical, learning, and data mining to the problem.
  - What was a luxury at 1000 nodes becomes a survival necessity at 100,000 nodes (1 million cores).

The Basics

We still need to …

- Instrument
- Collect
- Analyze
- Reduce
- Store
- Visualize

… statically, dynamically, for the source, binary, application, OS, runtime, interconnect, I/O system, etc.

But I'm not going to talk further about the technical solutions or areas, but rather the path to bringing these solutions to the people that need them.

Some Concerns

There has been some extremely good work done in this research community, but we're not keeping up.

- *Point solutions are only short-term victories (however they <u>are</u> victories ☺)*

Avoid a "5-year plan" mentality (declaring victory and moving on):

- *Many of the old problems are still here. Even though it's not sexy, we can't pretend they've all been solved.*
- *Real solutions to real problems are sexy.*

<u>Bottom line</u>: Five years from now, will we still be complaining about the same things (only point solutions)? Or can we effect a paradigm shift?

Enablers:

To keep the pipeline of tools flowing, we need:

1. A steady supply of new technical ideas
2. A means for testing them in the context of real applications.
3. A means for "hardening" them such that they are robust enough for others to use.
4. An infrastructure for distributing, proselytizing, training, and supporting the tools.

The flow through the pipe will be interrupted if you're missing <u>any</u> of these steps.

1. A Steady Flow of New Ideas

The problem space is too big for any one group to succeed alone

⇒ Big groups should stop trying for end-to-end solutions.

⇒ Collaboration is crucial.

- Reduces redundant efforts
- Enables us to agree upon interfaces so that we experiment with technical variations.

⇒ Basic functionality must be encapsulated in clean, concise components.

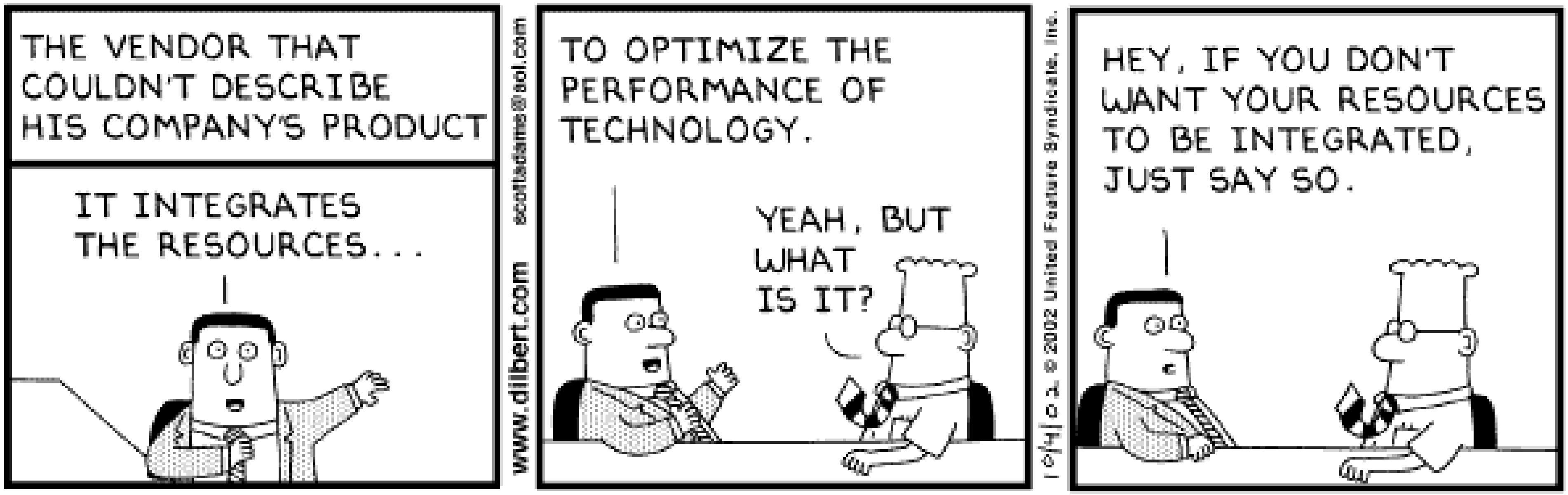
- Requires well-defined interfaces.
- Allows other groups to leverage our work.
- Allows other groups to improve upon our work.
- Provides a more equal opportunity for small groups.

1. A Steady Flow of New Ideas

If we can plug-n-play software in the same way that we plug-n-play hardware, we have some hope of keeping up, and <u>maybe</u> even supporting heterogeneous systems.

Groups need to span national boundaries.

- Teams and technologies are international, so research funding should reflect this.


THE VENDOR THAT COULDN'T DESCRIBE HIS COMPANY'S PRODUCT
IT INTEGRATES THE RESOURCES...
scottadams@aol.com
www.dilbert.com
TO OPTIMIZE THE PERFORMANCE OF TECHNOLOGY.
YEAH, BUT WHAT IS IT?
10/4/02 © 2002 United Feature Syndicate, Inc.
HEY, IF YOU DON'T WANT YOUR RESOURCES TO BE INTEGRATED, JUST SAY SO.

2. A Means for Testing on Real Programs

Testing on real programs requires:

⇒ A supply of real applications.

- This is getting easier as big groups work with real application groups. Need a clearinghouse for making these more generally available.

⇒ More general access to large systems.

- We need to create the expectation the your research results are not interesting unless they are demonstrated on a significantly sized problem (open question as to how big is "significant").
- Funding programs should <u>include</u> machine access; not as a separate (e.g., INCITE) proposal.

3. A Means for Hardening our Tools

This step is difficult for most groups:

⇒ Requires software engineering skills for which most researchers do not have the training or experience.

⇒ Significant time overhead in the progression:

- Fragile demo prototype
- Prototype that runs on a couple of real applications
- Tool that you can run reliably on many applications
- Tools that also runs on a variety of systems
- A significant and ongoing test regime.

⇒ Incompatible with academic promotion

"Nice software … good luck in your new job!"

⇒ Most grants don't provide funding for such activities.

4. Infrastructure for Distribution and Support

If the previous step is difficult for most groups, then this one is even harder:

⇒ Not academia: these skills are even further from those of our students and staff.

⇒ Not industry: ideally, each vendor should want to do this . . .

. . . but tools don't sell systems because the procurement specs don't include significant tool requirements.

⇒ Not the labs:

- Limited funding to support this activity and each lab usually has only a subset of systems.

A Few Recommendations

1. Team projects are crucial.

2. We need to identify the fundamental components:
   - Functionality boundaries and interfaces
   - Divide the work so that sharing becomes the default, not extra-effort special case.

A Few Recommendations

3. Funding should provide opportunity for a spectrum of proposals:
   - Basic concepts
   - Basic concepts + significant demonstration
   - Basic concepts + significant demonstration + distribution of software

   The further along you go, the larger the team and funding needed.
4. Application group engagement continues to be important (both technically and politically).

A Few Recommendations

5. Producing stable, usable software will require a new vision:
   - An organization that transcends individual labs, research groups, and companies.

     Each of these has its own agenda and limitations.
   - Stable long-term funding (at least a 5-year initial window)
   - A virtual center with:
     - A core team with key expertise
     - Incremental support (½ FTE?) to each participating software group.
     - Commitment in the form of a ¼ to ½ FTE from each vendor.

Funding Priorities

New techniques

- Anomaly detection
- Mining large performance database archives
- Analysis techniques for new architectures – GPUs, multi-core
- Integration of analysis into visualization
- Integration of system level data into application level analysis
- Data reduction - for both analysis and visualization

Funding Priorities

- Support for augmentation & componentization of existing infrastructure
  - rewarding developers whose software is reusable/reused
- Support for standardization
- Support for international collaborations
  - joint programs with other countries?
    - supercomputing infrastructure is a priority in the EU
  - travel and coordination represent a minimum investment which should yield good ROI
    - APART model
- Training support
  - fund engagement with application teams
  - workshops to help application teams learn to use tools productively
    - benefit for tool developers: having users adopt tools provides ROI to DOE

Funding Issues

Model for long term maintenance of software?

- Tools built by academic groups: how to support them in the long term?
  - tension in academia: innovation vs. support
  - who owns the tool?
    - awkward if owner is not original developer
  - co-locating innovators with maintainers is essential
    - maintenance programmers will burn out if divorced from innovation
- Examples: measurement support
  - support PAPI group to port and maintain on leadership class machines?
    - pro: the UTK folks are best qualified
    - con: should be vendor’s responsibility (LLNL)
  - standard OS interface is important too, e.g. perfmon
- Model for government and industry partnership in funding?
  - government + HPC vendors
  - alternate HPC revitalization message
    - those who buy the machines own the problem