Achieving the P in HPCS

After Dinner Entertainment (the cheap kind)

Greg Watson

Objectives

- Get a few laughs
  - We’re meant to be having a good time!
- Serious stuff
  - This is *free* entertainment after all
- Contentious stuff
  - So you’ll have some questions to ask
  - Something to talk about at the bar (you’re buying)

State of productivity

- Google “productivity”
  - 370,000,000 hits
- Google “programming productivity”
  - 40,300,000 hits
- Google “productivity” and "high end computing”
  - 46,800 hits
- Googling “productivty”
  - 26,700 hits!
- Serious problem

What do *we* mean by productivity?

- Definition (economic theory): output per unit time
- *Not* lines of code
- *Not* number of bugs fixed
- *Not* how fast a code runs

- The ability to solve customer problems quickly

"solve" => correct, validated solution
"quickly" => minimizing time to solution

Challenges to Productivity

- System complexity
  - 100's of thousands of processes, millions of threads
  - Mixture of clusters/SMPs/multi-cores
  - Co-processors/accelerators
  - Hardware customized to each job
  - Lots of asymmetry
  - Yet we haven't solved the problems of the current generation of machines
- Application complexity
  - Parallel is hard
  - Current programming models about the level of macro assemblers
  - Finer resolutions
  - Massive date sets
  - Increasing model complexity

How to achieve productivity?

- Reduce the apparent complexity of the systems
- Provide tools that streamline the development cycle
- Improve or replace existing programming models

LCTs vs IDEs

Alien

- Lean and mean
- Robust, hard to kill
- (Relatively) limited range of functions
- Breeders
  - Proliferation hard to control
  - Lots available when needed
- Communication between Aliens questionable
- Work well in large collections
  - Relies on a Queen to tell them what to do

Predator

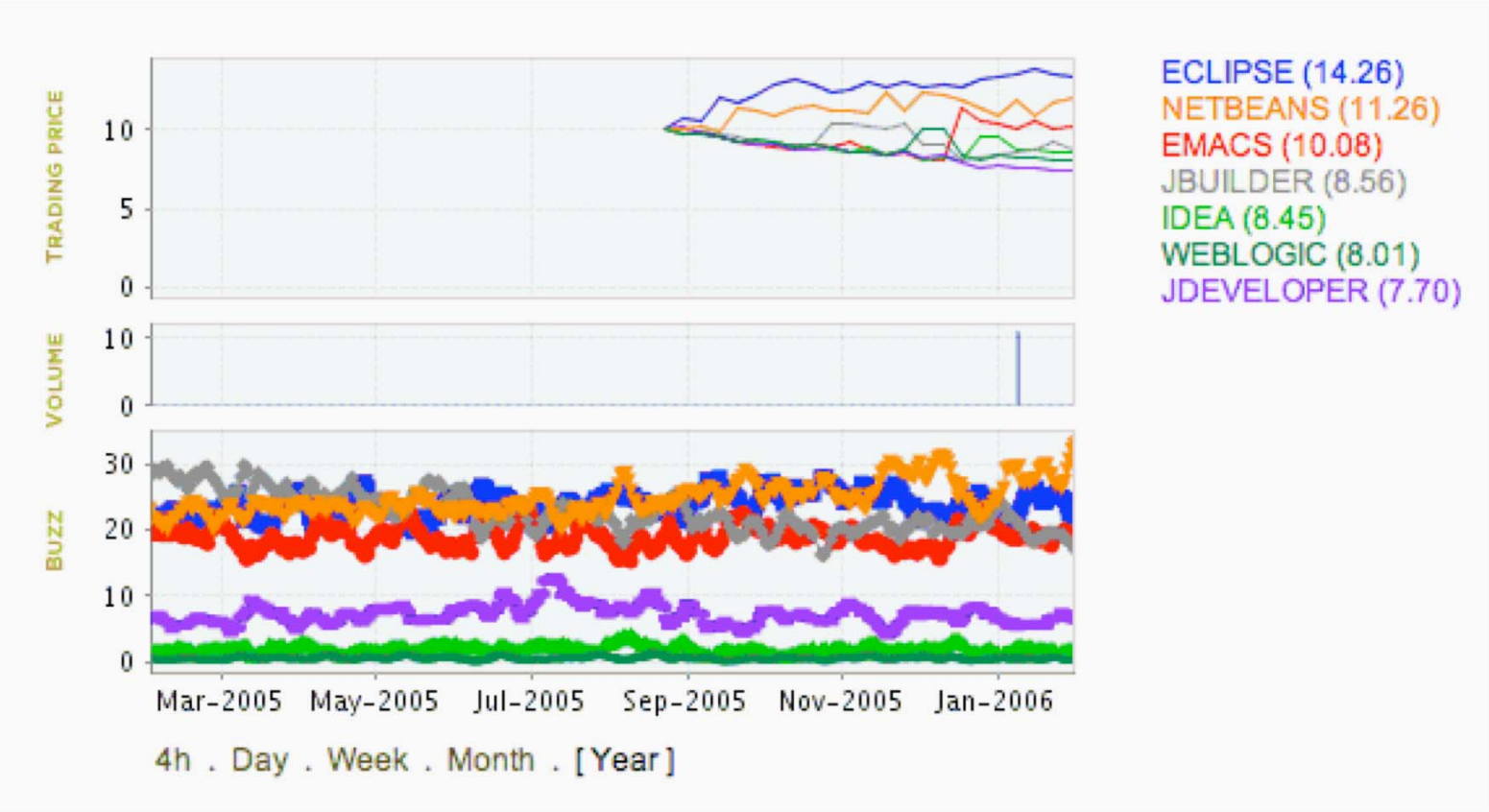
- Heavyweight
  - Latest technology needed to move around
- Packs sophisticated weaponry
- Able to switch weapons quickly & efficiently, without losing target
- Shares tactics with other predators
- Can become invisible when needed

So who wins?

- Best practice in most of the software development industry (but not HPC)
- Significant body of research points to improvements in productivity resulting from using IDEs
- Current tools will do nothing to reduce complexity, adding more will only compound the problem
- IDEs can provide functionality that is difficult or impossible to provide with separate tools
- IDEs can take advantage of new language paradigms

Drat!

Even so, there is a Java development environment for Emacs



What is possible now?

- Build support
  - IDE manages build process
  - Dependencies are calculated from project and source analysis
  - Automatic makefile generation (to allow non-IDE based builds)
  - Multiple multi-language tool chains (allows switching between different compilers, linkers, etc)
- Source editing
  - Visual editor with syntax highlighting
  - Yes, Emacs (or vi) mode is supported
  - Content assistance
  - Context-sensitive help

What is possible now? Continued…

- Refactoring
  - Behavior-preserving source code transformation
  - Why?
    - Because coding conventions, style, design, structure, architecture, and clarity all matter to productivity
- Examples of legacy language refactoring (e.g. Fortran)
  - Introduce local variables
  - Replace manifest constants with variables
  - Change global variables to function parameters
  - Change code to use new language features
    - Fixed to free format
    - New style “DO” loops
    - IMPLICIT NONE
  - Extract procedures/functions
  - Renaming (more difficult than it seems!)

What is possible now? Continued…

- Examples of modern language refactoring (e.g. Java)
  - Change method signature
    - Changes parameter names, parameter types, parameter order and updates all references to the corresponding method.
  - Extract interface/method/local variable/constant
    - Creates a new “thing” containing the statements or expressions currently selected and replaces the selection with a reference to the new “thing”.
  - Use supertype where possible
    - Replaces occurrences of a type with one of its supertypes after identifying all places where this replacement is possible.
  - Infer generic type arguments
    - Replaces raw type occurrences of generic types by parameterized types after identifying all places where this replacement is possible.
- Create refactoring “scripts” that can be applied to all or a group of files in the project.

What is possible now? Continued…

- MPI programming assistance
  - Content assist
    - Key sequence to show list of MPI completions
    - Provides argument type and return value details
    - Provides description of the MPI object
  - Context sensitive help
    - Help information available on demand during code writing process
  - Artifact navigation
    - Lists all MPI objects in code
    - Uses editor/view to navigate source code
- Debugging
  - Integrated debugger
  - Visual display of program state, current execution location, etc.
  - Point and click operation rather than line numbers

What is possible now? Continued…

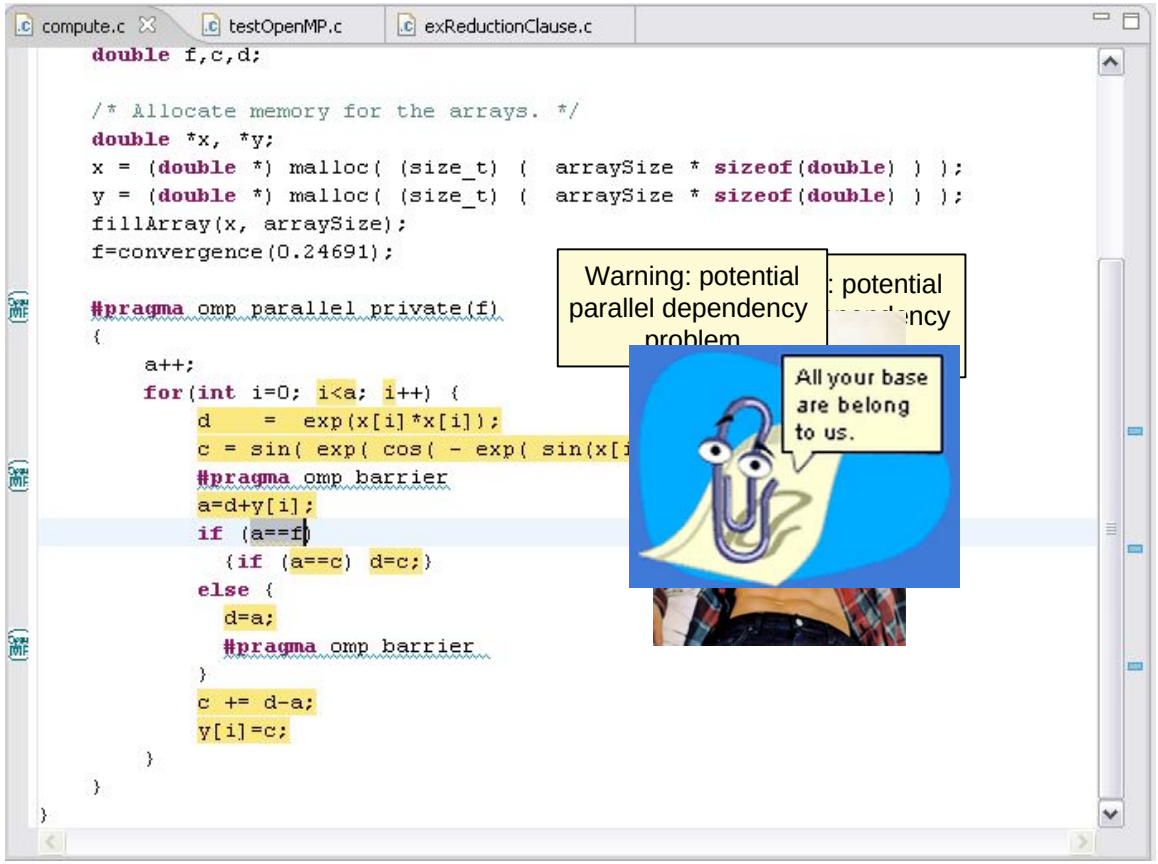
- Performance analysis
  - Integrated performance tools
  - Monitoring and logging of performance data
  - Profiling and tracing of program execution
  - Statistical and performance data viewers
- Testing
  - Integrated test tools (can also be run headless for automatic/nightly regression testing)
  - Test creation
  - Deployment and execution of tests
  - Execution history analysis and reporting
- Version control
  - Manage multiple repositories
  - Synchronize changes
  - Simplified branching/merging

What is possible in the future?

- Manage system complexity for the user
  - Provide a uniform view of complex, widely different architectures
  - Alleviate the need to learn system-specific details (e.g. job scheduler)
  - Ensure relevant information about the system is continuously available to the user
- Exploit new language features
  - Automated code generation
  - Blur division between editing/compilation
  - Improve language presentation
  - Visual programming
- Refactorings specific to scientific computing
  - Help transition from legacy codes

What is possible in the future? Continued…

- Sophisticated programming assistance for legacy languages



What is possible in the future? Continued…

- Address the debugging issues
  - How to debug a code that may run partly on a conventional processor and partly on an accelerator
  - How to debug a code that has been compiled to hardware
  - How to deal with millions of processes/threads
  - How to simplify debugging in general
- Harness tools to improve performance
  - Simplify the use of parallel performance tools
  - Automate much of the instrumentation, analysis and optimization processes
  - Take advantage of improved interaction with user during the development cycle

And much more…

What IDE to use?

	Visual Studio	Netbeans	Eclipse
Multi-OS	✗	✓	✓
Multi-language	✓	✗ [1]	✓
Open-source	✗	✓ [2]	✓ [3]
Parallel support	✗ [4]	✗	✓
Primary support base	1	1	15
Maturity	2001	2000	2001
Developer community	O(10M)	O(100K)	O(1M)
Third party plug-ins	~500	~50	~1200
Requirements	.NET framework	Java	Java

1. Planned for a future release
2. Common Development and Distribution License (CDDL)
3. Eclipse Public License (EPL)
4. Ready for Windows 2015 HPCS Edition

Conclusion

- Aliens are not going to meet productivity challenge
- Predator offers improvements and opportunities, but more work is needed
- If either loses, we lose
- Emacs, like Fortran, will be around forever
- The bar is still open!