The Eclipse Parallel Tools Platform: A Framework and Community for Integrating Parallel Tools

Greg Watson

Craig Rasmussen

Los Alamos NATIONAL LABORATORY EST.1943

The World's Greatest Science Protecting America

LAUR-05-0574

NNSA

Eclipse Platform?

- Eclipse provides a platform on which to build Eclipse-based applications.
- The platform is extended with plugins.
- Examples
  - JDT (Java Development Tools)
  - CDT (C/C++ Development Tools)
  - Photran/FDT (Fortran Development Tools)
- Leverage commercial development

Eclipse Foundation History

- Originally developed by Object Technology International (OTI) and purchased by IBM for use by internal developers
- Released to open-source community in 2001, managed by consortium
  - Eclipse Public License (EPL)
  - Based on IBM Common Public License (CPL)
- Consortium reorganized into independent not-for-profit corporation, the Eclipse Foundation, in early 2004
  - Participants from over 85 companies

Eclipse Foundation Developer Members

Eclipse Foundation
Commercial Tools Based on Eclipse

- Exadel Struts Studio and JSF Studio
- Genuitec MyEclipse
- IBM WebSphere Studio
- Intel C++ Compiler 8.1 for Linux
- Kinzan Studio
- M7 NitroX
- Mentor Graphics Nucleus Edge
- Monta Vista Dev Rocket
- Novell/SuSE SDK
- PalmOS Dev Suite
- Parasoft Jtest
- PureEdge Designer
- QNX Momentics
- Red Hat Developer Suite
- SAP NetWeaver Studio
- Tensilica Xtensa Xplorer IDE
- TimeSys TimeStorm IDE
- Wind River Workbench

Parallel Development Tools
State of the Art

- Command-line compilers for Fortran and C/C++
  - Sometimes wrapped in a GUI
- Editors are vi, emacs and FRED (vintage 1960's)
- Dominant debugger is TotalView (proprietary)
  - Some use DDT (but guess what? - it's proprietary)
  - No widely used open-source parallel debugger
- Plethora of stand-alone tools
  - Platform/vendor specific, e.g. DCPI
  - Open source, e.g. TAU, HPCToolkit
  - Proprietary, e.g. Vampir, Assure

Parallel Development Tools Limitations

- Many tools are specific to only one platform or vendor
- They do not interoperate, and never will
  - No integrated UI
  - No ability to share data
  - Functionality limited to that provided by tool
- They do not scale
  - Fine for 1990's machines
  - New machines will have 10,000+ processors
- Few high quality open-source tools in wide use
  - Increases difficulty of adopting new architectures
  - Can lead to vendor lock-in

Parallel Development Tools
Why Change?

- Reinforce good software engineering practices
- Strengthen auditing
- Enhance work-flow
- Increase productivity
- Improve documentation
- Reduce time-to-delivery

= Reduced development costs

Parallel Development Tools
Industry Best Practice (for everyone else)

- Integrated development environment (IDE)
  - Combines editor, compiler, debugger and other tools into a single consistent user interface
- Integrated management
  - Change control, build management, software quality policies
- Integrated testing
  - Automated unit testing, verification and validation activities
- Integrated documentation
  - On-the-fly documentation generation

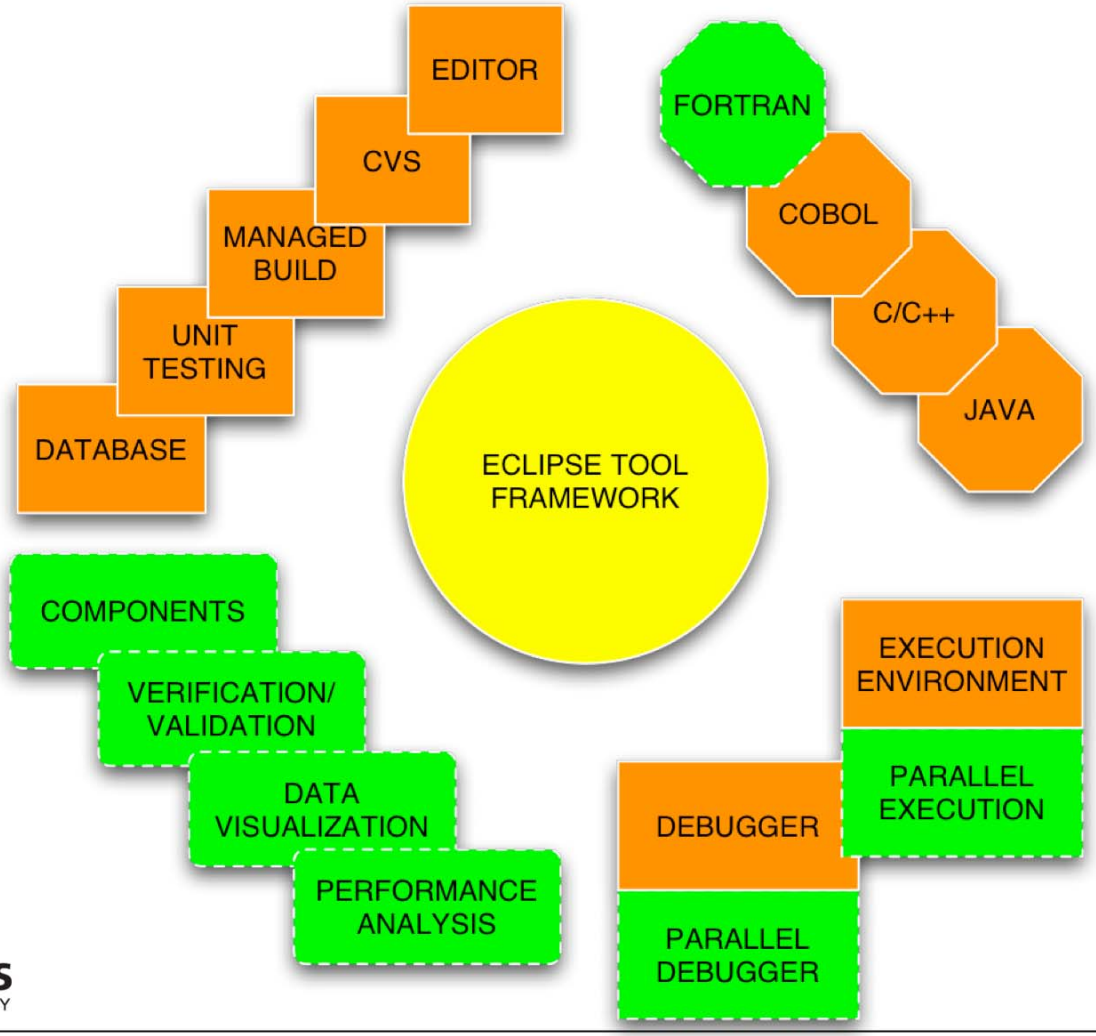
Goals of Eclipse Parallel Tools Platform (PTP)

- Build on top of tools already available
  - IDE, Editor, CVS, Debug, Launch
- Provide platform for new tool development
  - Petascale computing
  - Tools can be integrated to be aware of each other
    - Whole is greater than the sum of parts
- Goal is to integrate tools not develop new tools
  - There is already a large number of tools available

Parallel Tools Platform
Project Objectives

1. Extend Eclipse to support parallel development tools
2. Equip Eclipse with key tools needed to start developing parallel codes
3. Encourage existing parallel tool projects to support Eclipse
4. Exploit enhanced capabilities to develop a new generation of parallel tools

Parallel Tools Platform Components

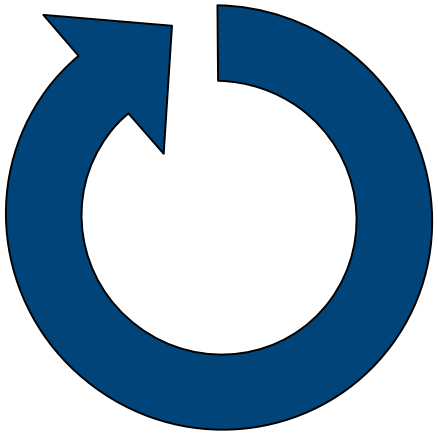


Potential Partners

- IBM (language tools and debugging)
- Cray (debugging)
- Intel (build, debugging and deploy)
- University of Illinois (Fortran development tools)
- University of Oregon (Performance analysis tools)
- University of Houston (Program optimization)
- University of Tennessee (Performance tuning and deployment)
- Rice University (Co-Array Fortran)
- Livermore (Refactoring and program optimization)
- Monash University (debugging)

Scientific Application Life Cycle

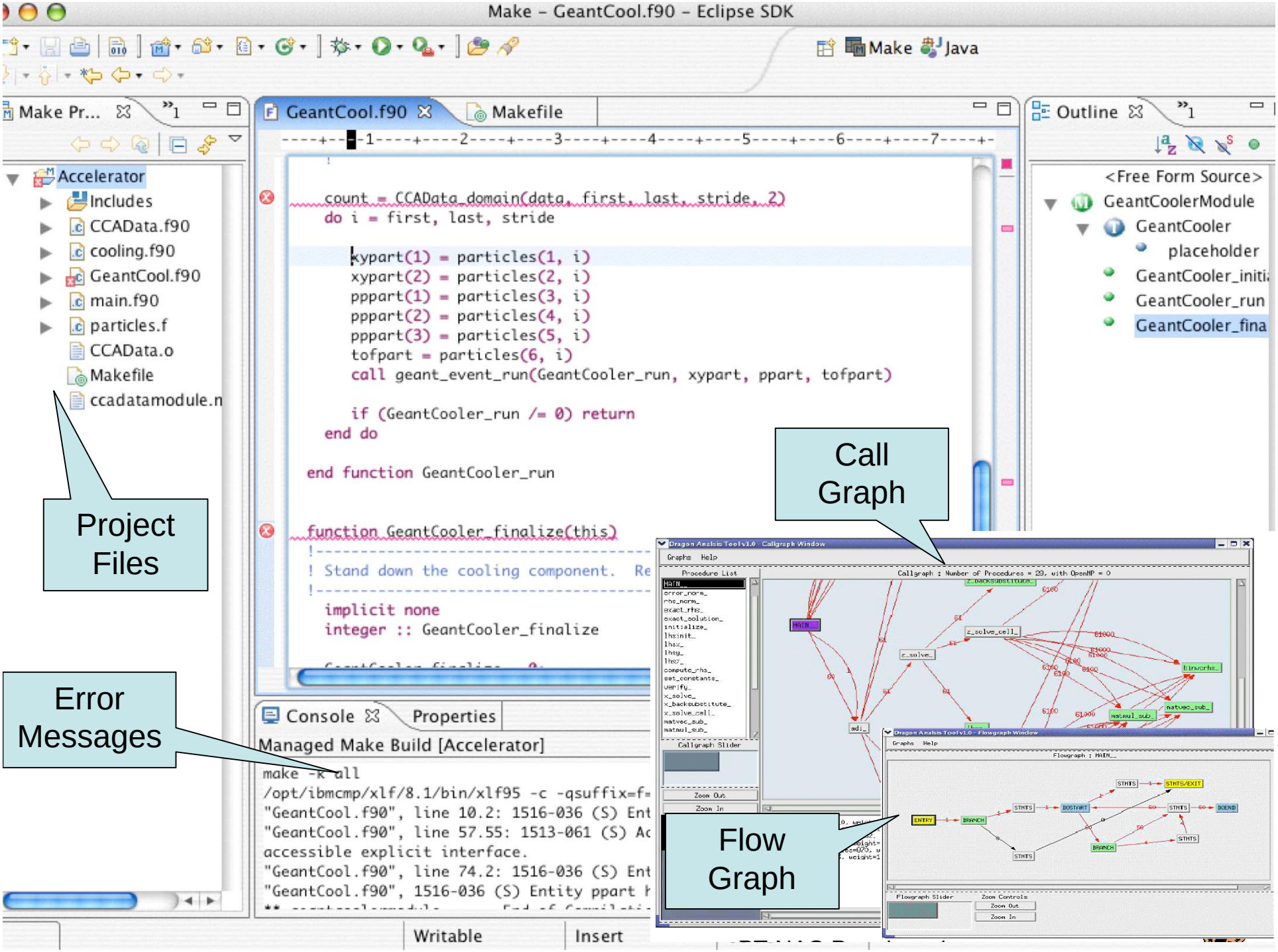
Design & Develop



Log & Deploy

Debug & Test

Performance Analysis & Tuning


Make – GeantCool.f90 – Eclipse SDK
Make
Java
Make Pr...
Accelerator
Includes
CCAData.f90
cooling.f90
GeantCool.f90
main.f90
particles.f
CCAData.o
Makefile
ccadatamodule.n
GeantCool.f90
Makefile
count = CCAData_domain(data, first, last, stride, 2)
do i = first, last, stride
xypart(1) = particles(1, i)
xypart(2) = particles(2, i)
pppart(1) = particles(3, i)
pppart(2) = particles(4, i)
pppart(3) = particles(5, i)
tofpart = particles(6, i)
call geant_event_run(GeantCooler_run, xypart, ppart, tofpart)
if (GeantCooler_run /= 0) return
end do
end function GeantCooler_run
function GeantCooler_finalize(this)
! Stand down the cooling component.
implicit none
integer :: GeantCooler_finalize
Outline
<Free Form Source>
GeantCoolerModule
GeantCooler
placeholder
GeantCooler_initi
GeantCooler_run
GeantCooler_fina
Project Files
Call Graph
Error Messages
Flow Graph
Console
Properties
Managed Make Build [Accelerator]
make -k all
/opt/ibmcmp/xlf/8.1/bin/xlf95 -c -qsuffix=f=
"GeantCool.f90", line 10.2: 1516-036 (S) Ent
"GeantCool.f90", line 57.55: 1513-061 (S) Ac
accessible explicit interface.
"GeantCool.f90", line 74.2: 1516-036 (S) Ent
"GeantCool.f90", 1516-036 (S) Entity ppart h
Writable
Insert

Existing Design & Development Facilities

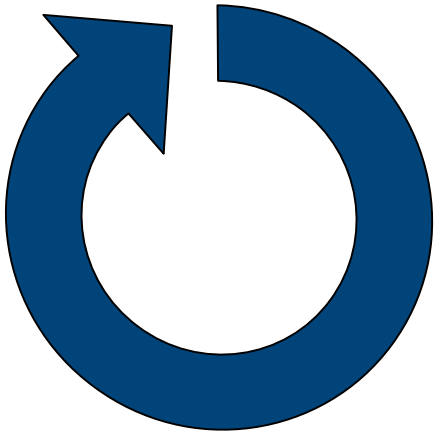
- Integrated development environment (IDE)
  - Combines editor, compiler, debugger and other tools into a single consistent user interface
- Integrated management
  - Build management, version control, software quality
- Integrated testing (TPTP)
  - Automated unit testing, performance monitoring and analysis
- Integrated documentation (JavaDoc)
  - On-the-fly documentation generation

Future Development Capabilities

- Static analysis infrastruture
  - Embedded C++ and Fortran parsers
  - Rose
- Refactoring for scientific computing
  - Language interoperability
  - Fortran: constant replacement and type promotion
  - Semi-automatic parallelization (OpenMP)
- Program graphs (e.g. call tree graphs)
- Error checking
- Assisted documentation generation (Doxygen)
- Support for new parallel languages
  - Co-array Fortran, UPC

Scientific Application Life Cycle

Design & Develop



Log & Deploy

Debug & Test

Performance Analysis & Tuning

Existing Debug & Test Facilities

- Sequential debugger
  - Visual representation of program state, current execution location
  - Point and click rather than line numbers
- Parallel debugger
  - Gang control of arbitrary process sets
  - Visual display of multiple program counter locations
  - Scalable architecture
- Testing
  - Test creation
  - Deployment and execution of tests
  - Execution history analysis and reporting

Future Debug Capabilities

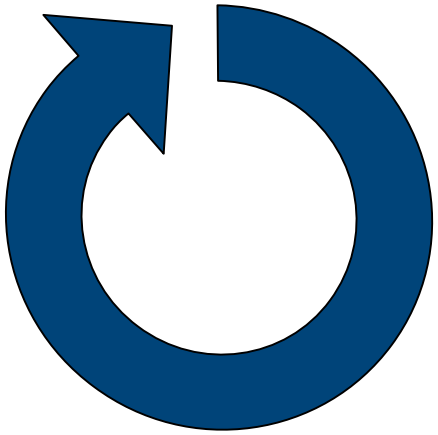
- Advanced Parallel Debugging
  - Data-centric debugging
  - Look at data rather than just instruction stream
    - Provide data viewers
  - Relative debugging (data centric)
    - Define assertions graphically
    - Run on new and existing platforms and compare
    - Regression testing
- MPI specific debugging
  - Integrate debugger with performance traces
    - XMPI

Future Test Capabilities

- Automated unit test stub generation
  - Fortran and C/C++
  - Output results to spreadsheet (for example)
- Integrate Verification and Validation
  - Logging
- Collaborate with other V&V initiatives

Scientific Application Life Cycle

Design & Develop

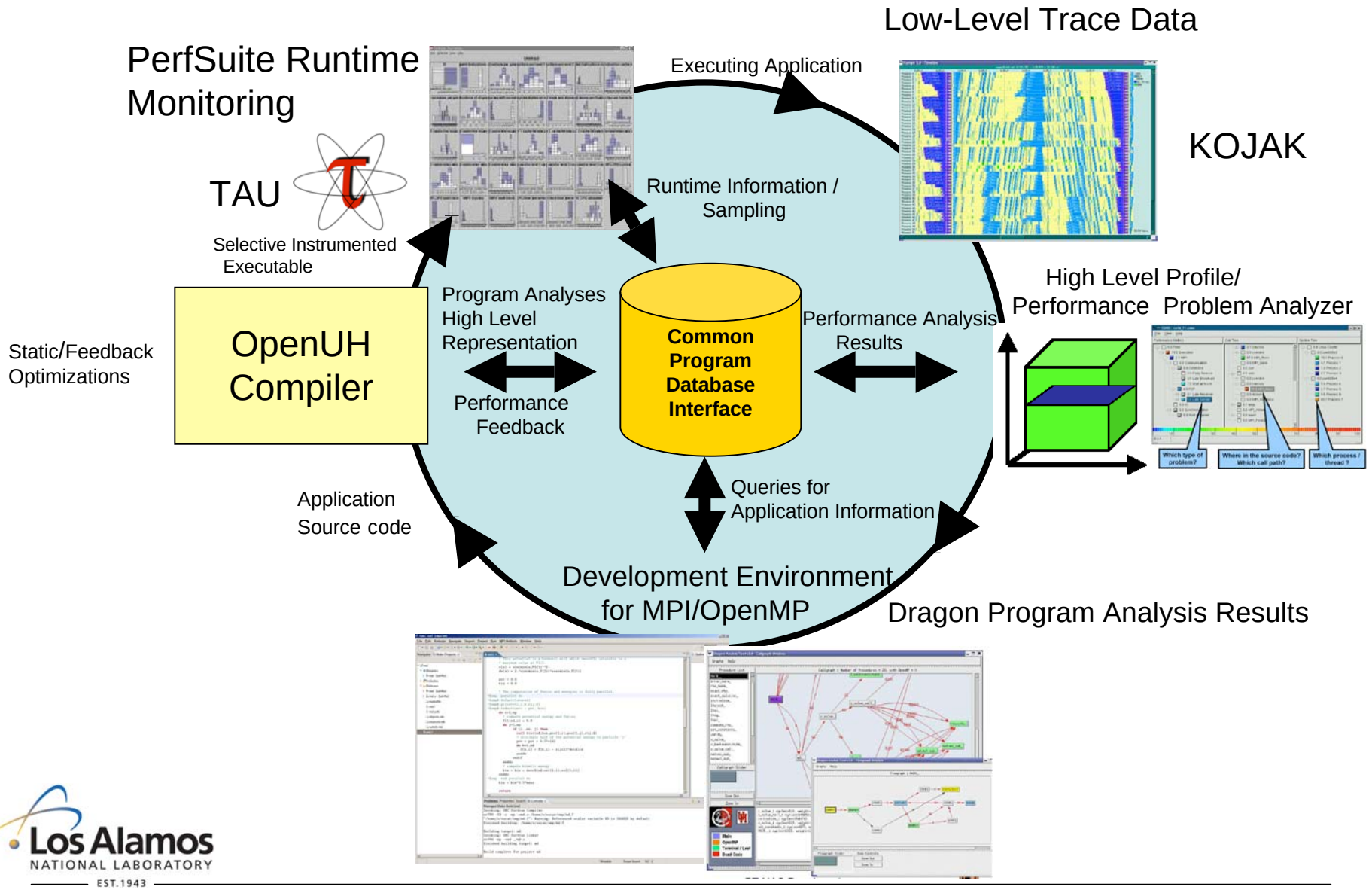


Log & Deploy

Debug & Test

Performance Analysis & Tuning

Compiler Learning



Existing Performance Analysis and Tuning

- Test and Performance Tools Platform (TPTP)
  - for sequential applications
- TAU stage 1 integration
  - Ability to instrument and launch TAU instrumented code
  - Launch paraprof viewer within Eclipse

Future Performance Analysis Capabilities

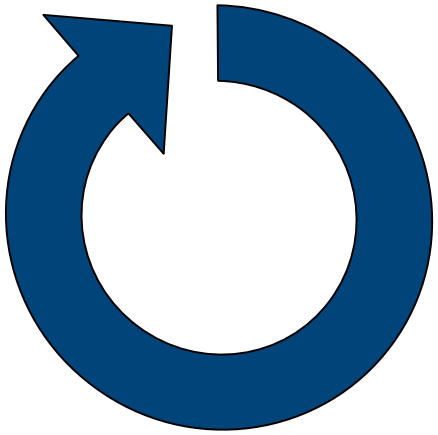
- Provide framework for performance analysis tools
  - Automatic instrumentation
  - Data collection
  - Data analysis
  - Visualization
- TAU
- Open|SpeedShop
- HPC Toolkit
- PMPI

Future Tuning Capabilities

- Framework for performance tuning using empirical methods
- Requires
  - Launch code
  - Gather performance data
  - Analyze performance data
  - Code/Compiler optimization
  - Repeat (search space)
- MPI modeling tools can utilize same framework

Scientific Application Life Cycle

Design & Develop



Debug & Test

Log & Deploy

Performance Analysis & Tuning

Existing Deployment & Logging Capabilities

- Plugin architecture
  - Explicit listing of dependencies
  - Exporting of interfaces
  - Versioning
- Packaging wizards
- Web-based deployment
  - Automatic build/install on download

Future Deployment & Logging Capabilities

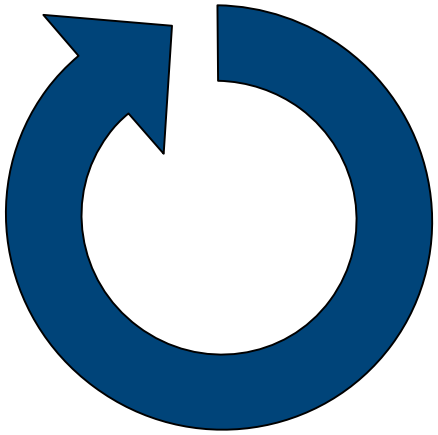
- Deployable packages
  - Autoconf based
  - Explicit listing of dependencies on other tools
  - Package version
  - List of interfaces (for components)
- Web-based deployment
  - Automatic configure/build/install on download
- Provide history through logging
  - Build tools (compilers and options, libraries, versions, …)
  - Test history
  - Run history (traces, profiles, …)

Conclusion

- PTP is about tool integration
- Interfaces to tools for platform and tool independence
- Step 1: Tool is runnable from eclipse environment
- Step 2: Tool integration with eclipse environment
- Step 3: Tool to tool integration
- PTP integrated tools
  - Greater than the sum of the parts

Questions?

Design & Develop



Log & Deploy

Debug & Test

Performance Analysis & Tuning