Integrating Compilers to Support Application Development & Optimization in Eclipse /PTP

University of Houston

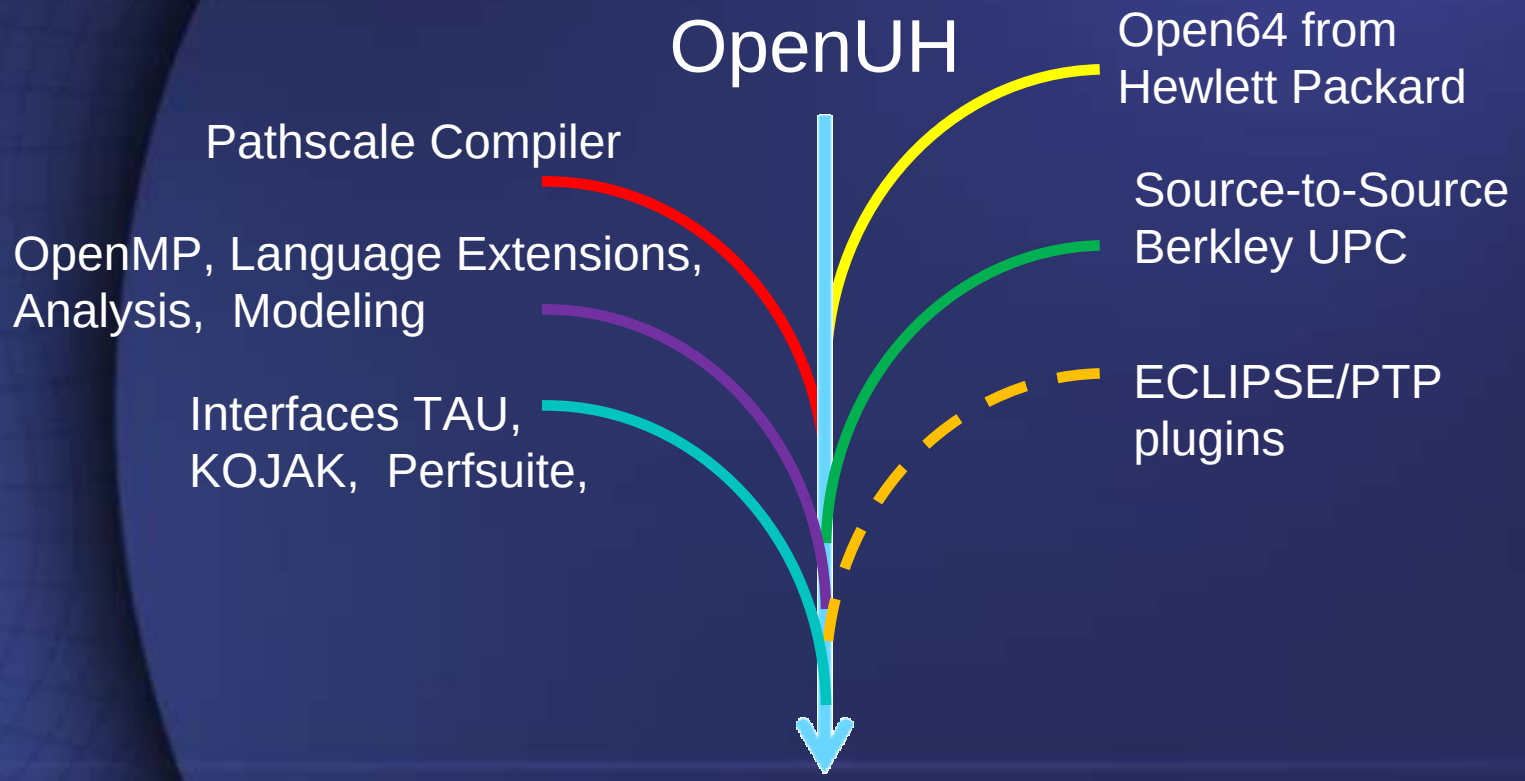
Oscar Hernandez

Barbara Chapman

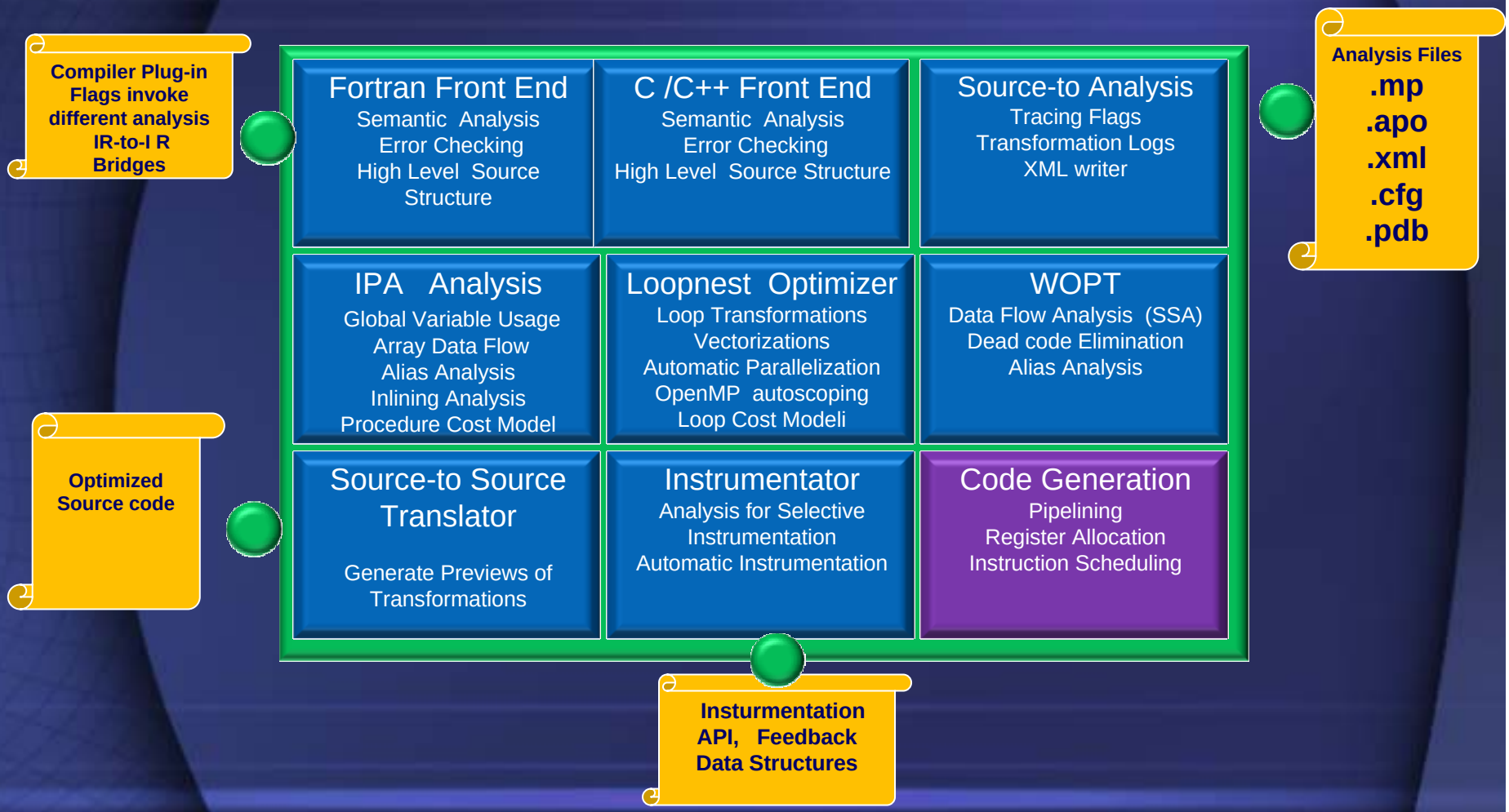
The Open64 Compiler

- A robust suite of open source optimizing compiler tools for Linux/Intel IA-64 systems. It's in public domain.
  - Originally developed by SGI, and currently maintained by Hewlett-Packard. Other companies have Open64-based products.
  - Full support for F95/F90/F77, C, C++.
  - State-of-the-art analysis and optimizations.
- Our branch of Open64 is called OpenUH:
  - Supports OpenMP 2.0 and optimizations,
  - Improved source-to-source capabilities
  - Tools interfaces, export static analysis, performs instrumentation, runtime library include performance monitoring.
  - Supports automatic parallelization.

Integration Efforts



Open64 Analysis Infrastructure

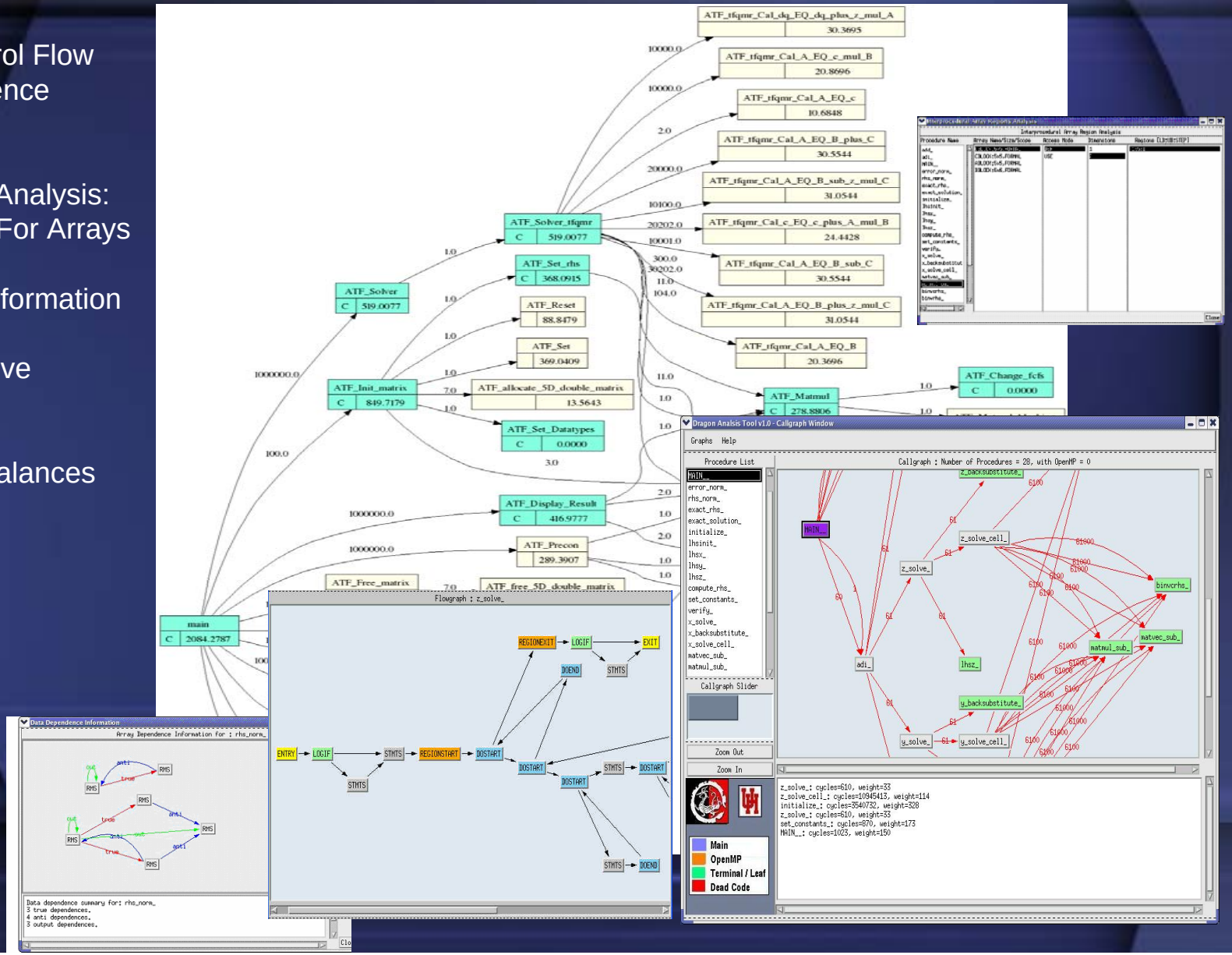


How compiler analysis can help:

- Be a source of information for the application developer.
- Provide the infrastructure to support development/optimization environments:
  - Semantic Error/Checkers
  - Refactoring Tools
  - Modeling Tools
  - Assist the creation of parallel code (e.g. OpenMP, Hybrid MPI/OpenMP code)
- Support work of other tools.

Example: Dragon Analysis Tools.

- Callgraph, Control Flow Graph, Dependence Analysis.
- Interprocedural Analysis: Variable Usage For Arrays
- Cost Modeling Information
- Detects Expensive Procedures
- Detect Load Imbalances (MPI/OpenMP)

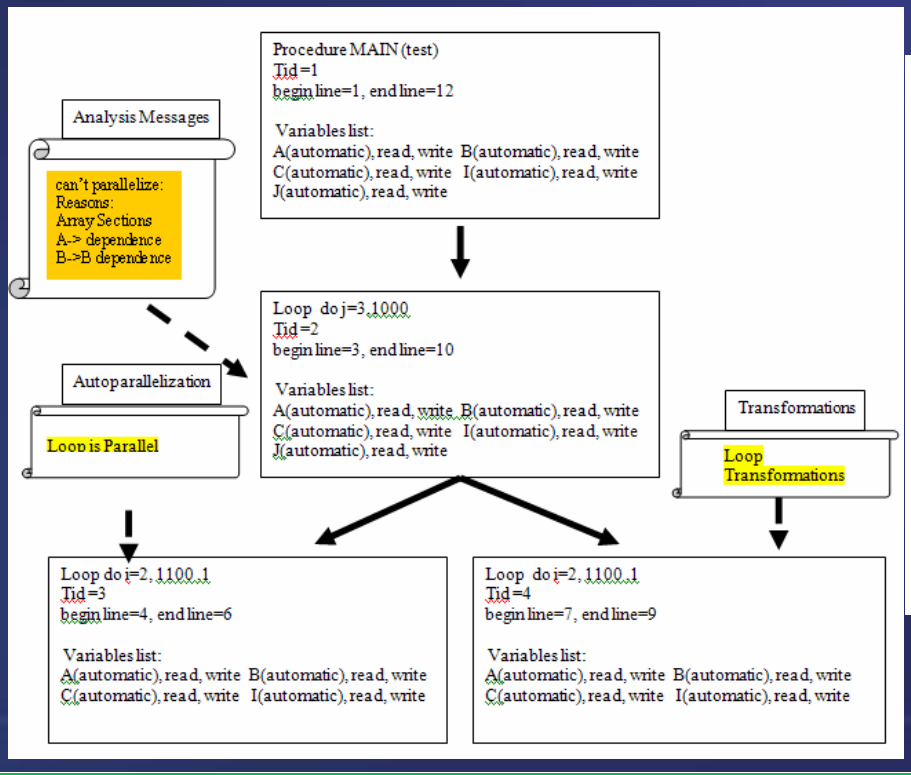


Program Transformations

Source Code

```
1:    program test
2:    real a(1100),b(1100),c(1100)
3:    do j=3,1000
4:    do i=2, 1100 ,1
5:      a(i) = b(i) + c(i)
6:    enddo
7:    do i=2, 1100 ,1
8:      b(i) = a(i) + c(i)
9:    enddo
10:   enddo
11:  write(*,*) a(500)
12:  end program
```

Hierarchical Representation of Program



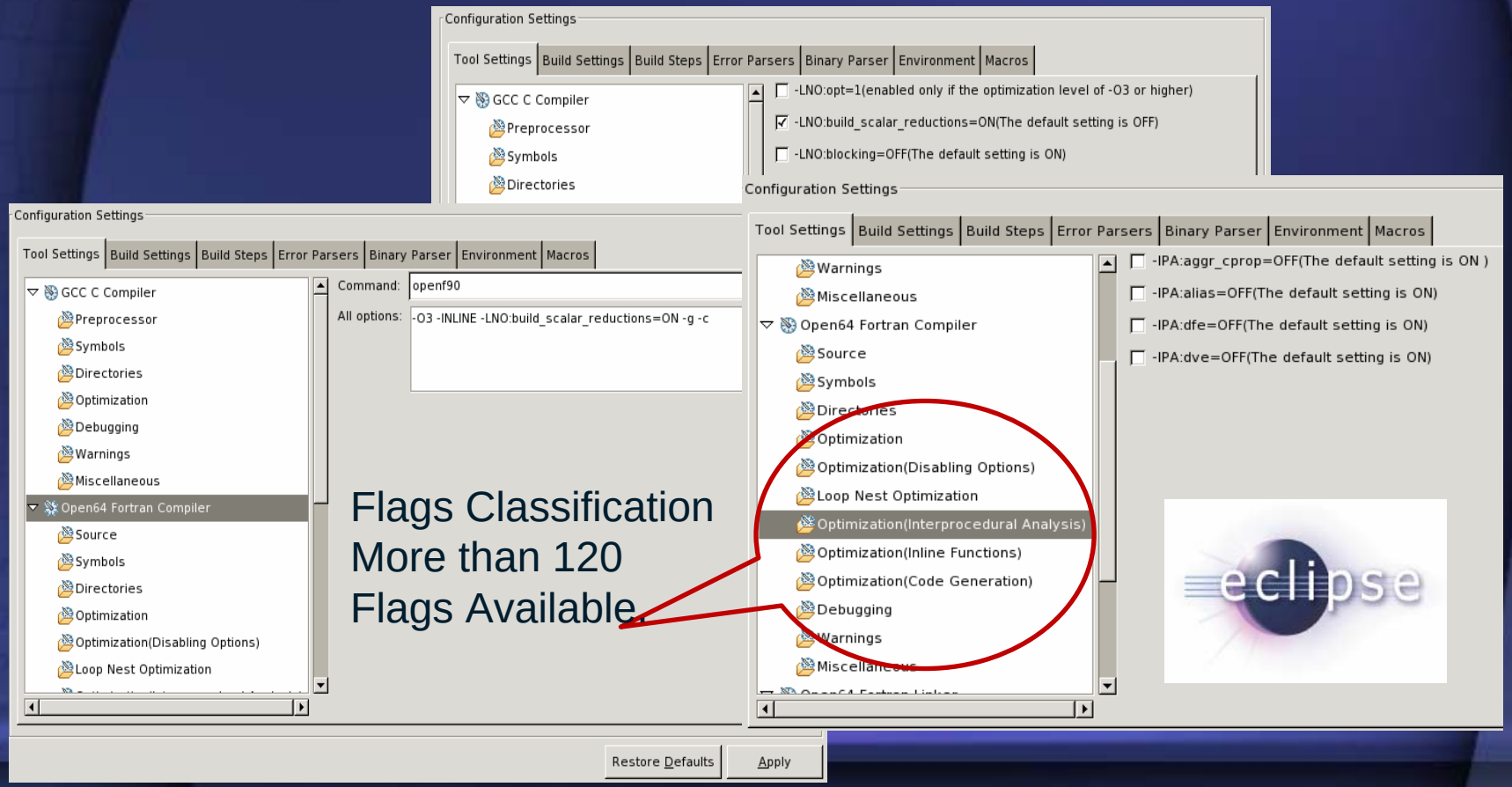
Optimization Logs

```
product "Mongoose"
version "3.1"
language "f90"
whirl2f "test.m"

function 1 "MAIN" range [1 1 0]-[1 12 0]
varlist 1 "A"(A):rw, "B"(A):rw, "C"(A):r,
"I"(A):rw, "J"(A):rw

doloop 2 "do J" range [1 3 0]-[1 10 0]
index "J"
varlist 2 "A"(A):rw, "B"(A):rw, "C"(A):r,
"I"(A):rw, "J"(A):rw
doloop 3 "do I" range [1 4 0]-[1 6 0]
index "I"
varlist 3 "A"(A):w, "B"(A):r, "C"(A):r, "I"(A):rw
end_doloop 3
doloop 4 "do I" range [1 7 0]-[1 9 0]
index "I"
varlist 4 "A"(A):r, "B"(A):w, "C"(A):r, "I"(A):rw
end_doloop 4
end_doloop 2
end_function 1

TRANSFORMATION_LOG_BEGIN
INTERCHANGE 2 , 3 3 , 2
TRANSFORMATION_LOG_END

PARALLELIZATION_LOG_BEGIN
tid 2 ARRAY_DEP highlight "A" lines 1
highlight "A" lines 1
tid 2 ARRAY_DEP highlight "B" lines 1
highlight "B" lines 1
tid 2 PARTIAL_ARRAY_SEC highlight "A"
tid 3 PARALLELIZED
tid 4 PARALLELIZED
PARALLELIZATION_LOG_END

DOACROSS_LOG_BEGIN
tid 3 "__mpdo_MAIN__1"
tid 4 "__mpdo_MAIN__2"
DOACROSS_LOG_END

NEST_LOG_BEGIN
tid 2 1
tid 3 2
tid 4 2
NEST_LOG_END

POST_DOACROSS_LOG_BEGIN
tid 3 "__mpdo_MAIN__1"
tid 4 "__mpdo_MAIN__2"
POST_DOACROSS_LOG_END

POST_NEST_LOG_BEGIN
tid 2 1
tid 3 2
tid 4 2
POST_NEST_LOG_END

msg loop lines [1 7 0]
msg loop lines [1 1 0]
msg loop lines [1 4 0]

msg loop lines [1 4 0]

msg loop lines [1 7 0]
SRCPOS_MAP_BEGIN
SRCPOS_MAP_END
```

Optimized Source Code

Initial Plug-in for PTP 1.1

Tool Chain plug-in for OpenUH/Open64.



Major Challenges

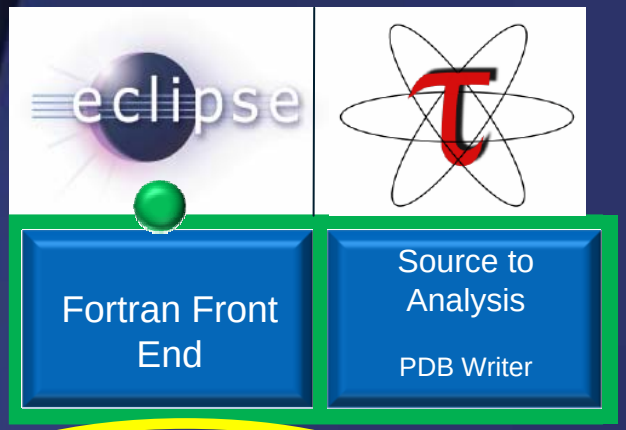
- Portability to support all Eclipse/PTP users.
- Interactions with tools can be complicated
  (e.g. IR-to-IR mappings, two-way interactions)
- No standard intermediate representation
- No standard format represents output of analyses
- Mappings of analyses to the source code has to be maintained.
- Provide analysis in an intuitive way to the user.
  (e.g. scalability is a problem)

Conclusions

- We have began to move our compiler to Eclipse/PTP
- Focus so far:
  - Exporting analyses
  - Making it easy to invoke the compiler
  - Defining interfaces with PTP/other tools
- Many standard interfaces seem to be needed.

Questions?

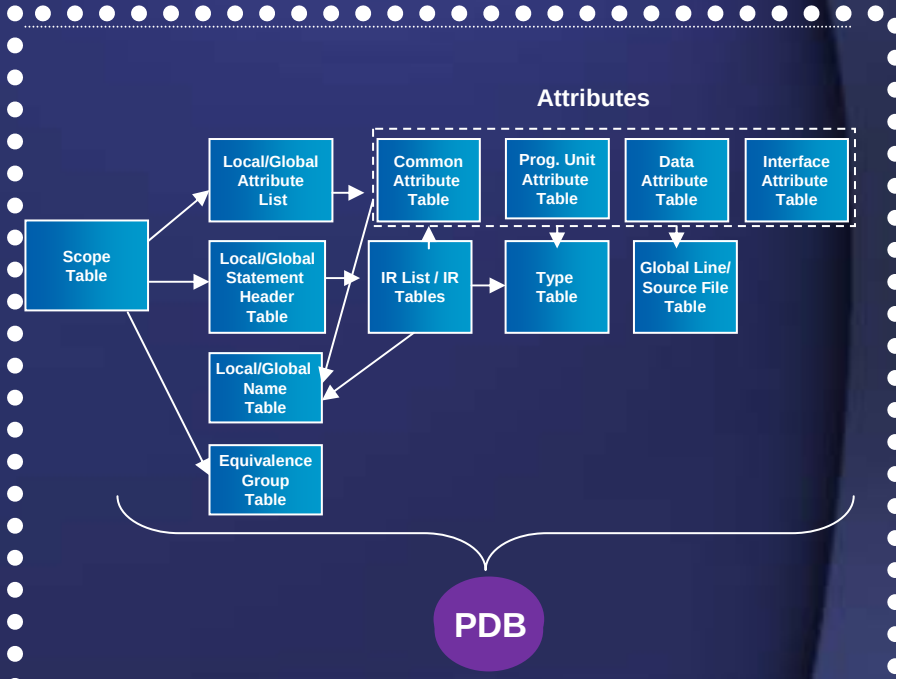
Example: Conversion to Program Database Representation



Source Code

PDB Format

Intermediate Representation



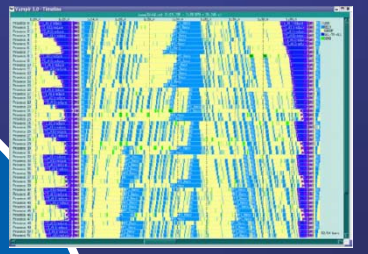
- High level representation of Programs
- Conversion to Other Representations

Integrated Tunning Enviroment

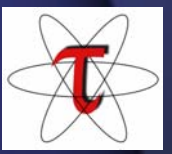
PerfSuite Runtime Monitoring

KOJAK

Low-Level Trace Data



TAU



Executing Application

Runtime Information / Sampling

Selective Instrumented Executable

Static/Feedback Optimizations

OpenUH/ORC Compiler

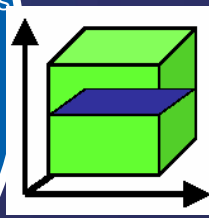
Program Analyses High Level Representation

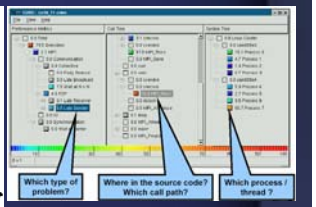
Performance Feedback

Common Program Database Interface

Performance Analysis Results

High Level Profile/ Performance Problem Analyzer

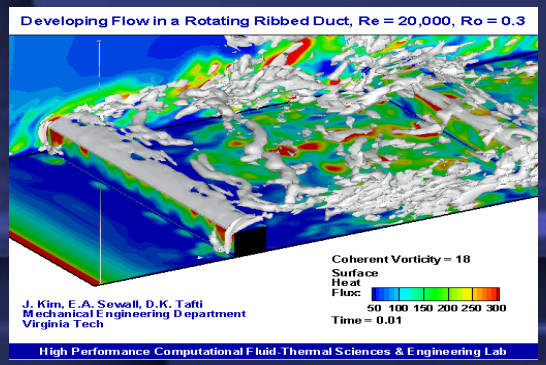




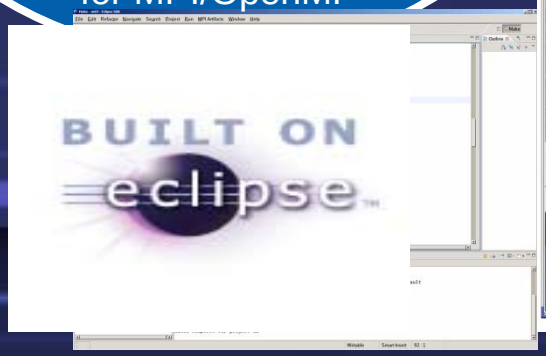
Queries for Application Information

Application Source code

Fluid Dynamics Application



Development Environment for MPI/OpenMP



Dragon Program Analysis Results