



SOFTWARE

+ ☐ 19.56 updatex
+ ☐ 399.70 updateien
+ ☐ 0.00 gene
+ ☐ 0.00 <<iteration loop>>
+ ☐ 447.52 genbc



FAST SOLUTIONS

- ☒ PAPI_L1_ICM
- ☐ PAPI_L2_DCM
- ☒ PAPI_L2_ICM
- ☐ PAPI_L1_TCM

Periscope Tutorial Exercise NPB-MPI/BT

M. Gerndt, V. Petkov, Y. Oleynik, S. Benedict

Technische Universität München

periscope@lrr.in.tum.de

October 2010

- Intermediate-level tutorial example
- Available in MPI, OpenMP, hybrid OpenMP/MPI variants
 - also MPI File I/O variants (collective & individual)
- Automatic performance properties search with Periscope:
 - Source code instrumentation
 - ▶ Loops, MPI & application function calls
 - Automatic search for slow MPI communication patterns
 - Results exploration with Eclipse based GUI
- Manual instrumentation optimization

- Configuration of Periscope
- Program instrumentation: `psc_instrument`
- Periscope analysis: `psc_frontend`
- Performance properties exploration: `Periscope GUI`
- Documentation:
 `DVD Index` → `Periscope User's Guide`

- Before first use of Periscope, one has to create the configuration file `.periscope` in the home directory. Configuration could be copied from `$PERISCOPE_ROOT`:

```
% cp $PERISCOPE_ROOT/etc/periscope.sample ~/.periscope
```

- It should look like:

```
MACHINE      = localhost      //hostname
SITE         = LiveDVD
REGSERVICE_HOST = localhost    //host of registry
REGSERVICE_PORT = 50001        //port of the registry
APPL_BASEPORT  = 51000          //first port for application
AGENT_BASEPORT = 50002          //first port agent hierarchy
```

- Registry enables the agents and the application to connect.
- The registry listens at the port specified in `.periscope`.
- Command line:
 - > `pvc_regrsv &`
- Check the registry:
 - > `telnet localhost port_number`
 - > `list`
 - > `quit`

- A program must be instrumented before it can be analyzed by Periscope.
- Easily done by prefixing the compiler command with our instrumentation script *psc_instrument*:

```
% psc_instrument --help
Periscope Source-to-Source Instrumentation Wrapper
Usage: psc_instrument [-t regions] [-n] [-s sir] [-v] [-d] compiler
      [options] file [libs]
-t Types of regions to instrument separated by commas
  (e.g. -t user,loop,call,mpi,omp)
-s File name for the resulting SIR file (default: appl.sir)
-f <fixed | free> Force a specific Fortran file format
-v Verbose output
-d Debug mode: keeps the instrumented source files
-n Prints each step of the compilation instead of executing them
```

- Edit NPB3.3/config/make.def
- remove `#` in line 39
 - #MPIF77 = psc_instrument ...
- Compile with
 - > `make bt CLASS=W NPROCS=16`

- Why?
 - To reduce the instrumentation overhead.
 - To circumvent problem with source level instrumentation.
- How?
 - `-t` switch of `psc_instrument`
 - Determines the region types to be instrumented.
 - All files are instrumented in the same way.
 - `psc_inst_config`
 - Generated from all source files in the directory
 - If `-t` switch is omitted, the file determines the instrumentation.
 - You can specify: none, module, sub, loop ...

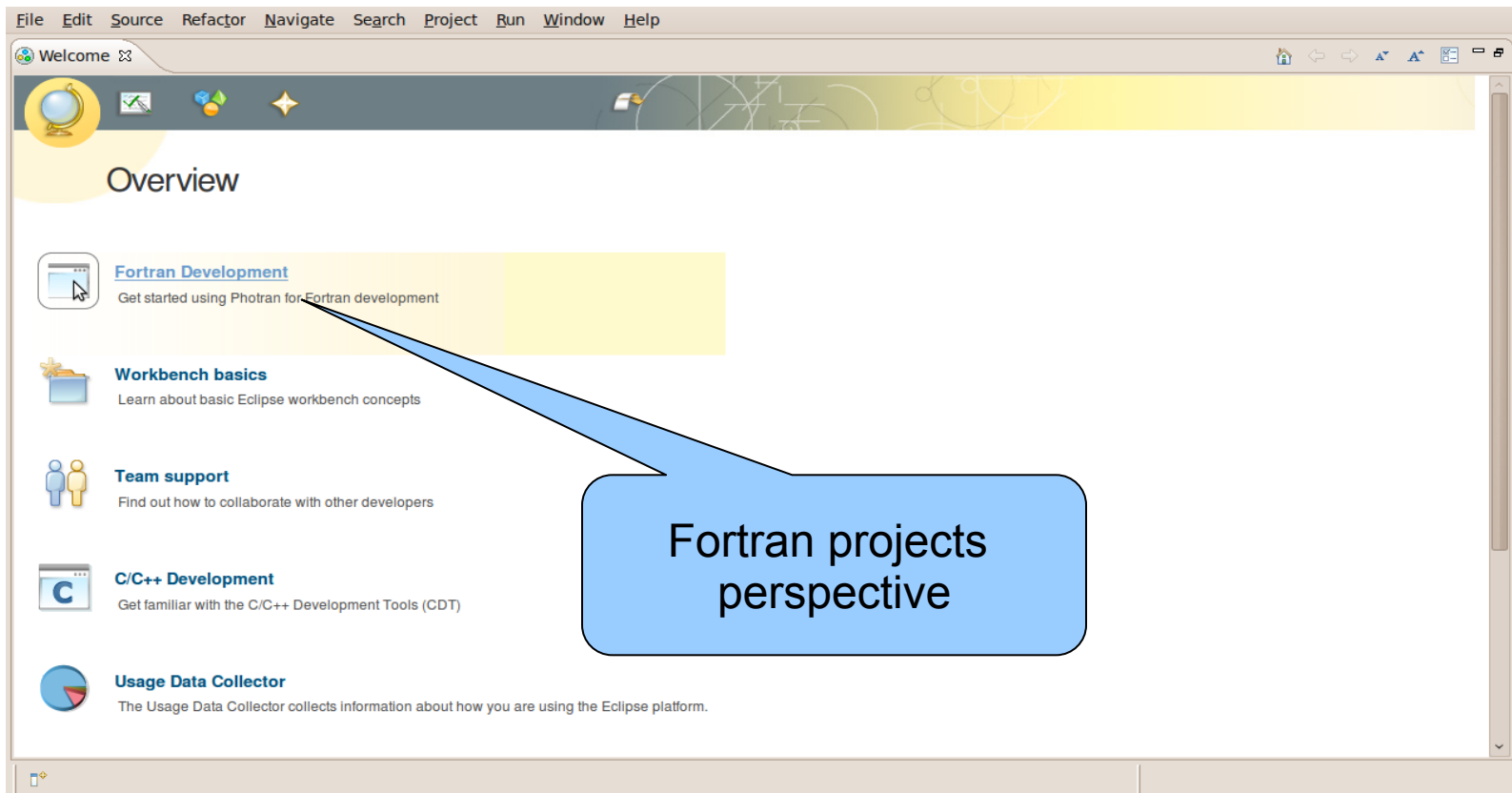
- Periscope is started via the frontend. It automatically starts application and hierarchy of analysis agents.
- Run `pvc_frontend --help` for brief usage information

```
% pvc_frontend --help
Usage: pvc_frontend <options>
  [--help]                (displays this help message)
  [--quiet]               (do not display debug messages)
  [--registry=host:port]  (address of the registry service, optional)
  [--port=n]              (local port number, optional)
  [--maxfan=n]            (max. number of child agents, default=4)
  [--timeout=secs]        (timeout for startup of agent hierarchy)
  [--dontcluster]         (Do not use online clustering)
  [--debug=level]         (search delay in phase executions)
  [--delay=n]             (search delay in phase executions)
  [--appname=name]        (search delay in phase executions)
  [--apprun=commandline]  (bt_W.16 or "your_app -F config.file")
  [--mpinumprocs=number of MPI processes]
  [--ompnumthreads=number of OpenMP threads]
  [--strategy=name]       (MPI, SCA, SCABF, P6, P6BF, P6BF_Memory,
                           SCPS_BF, scalability_OMP)
  [--sir=name]            (File containing the instrumentation outline)
```

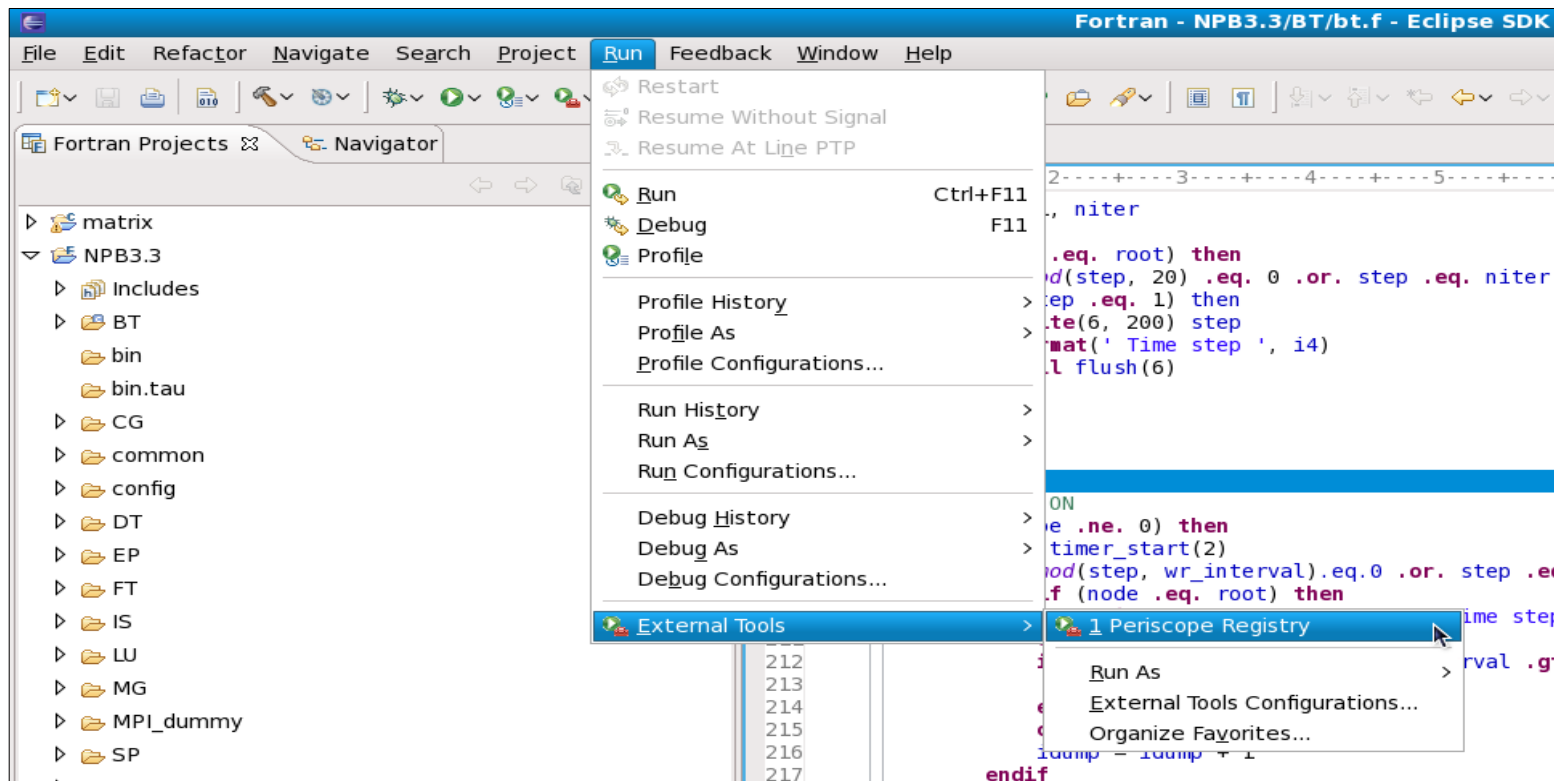
- Start the benchmark
 - > `cd bin.periscope`
 - > `psc_frontend --apprun=bt_W.16 --mpinumprocs=16`
 - `--strategy=MPI`
 - `--sir=bt_W.16.sir`
 - `--bg-mode=SMP, DUAL, VN`
 - `--maxfan=10`
- Steps
 1. Frontend starts the application.
 2. FE starts the agent hierarchy.
 3. Analysis agents execute one or more analysis steps.
 4. AA report the properties back to the frontend.
 5. FE outputs the found properties into `properties.psc`

- Double-click on **Eclipse** icon on the Desktop
- Or start it from console

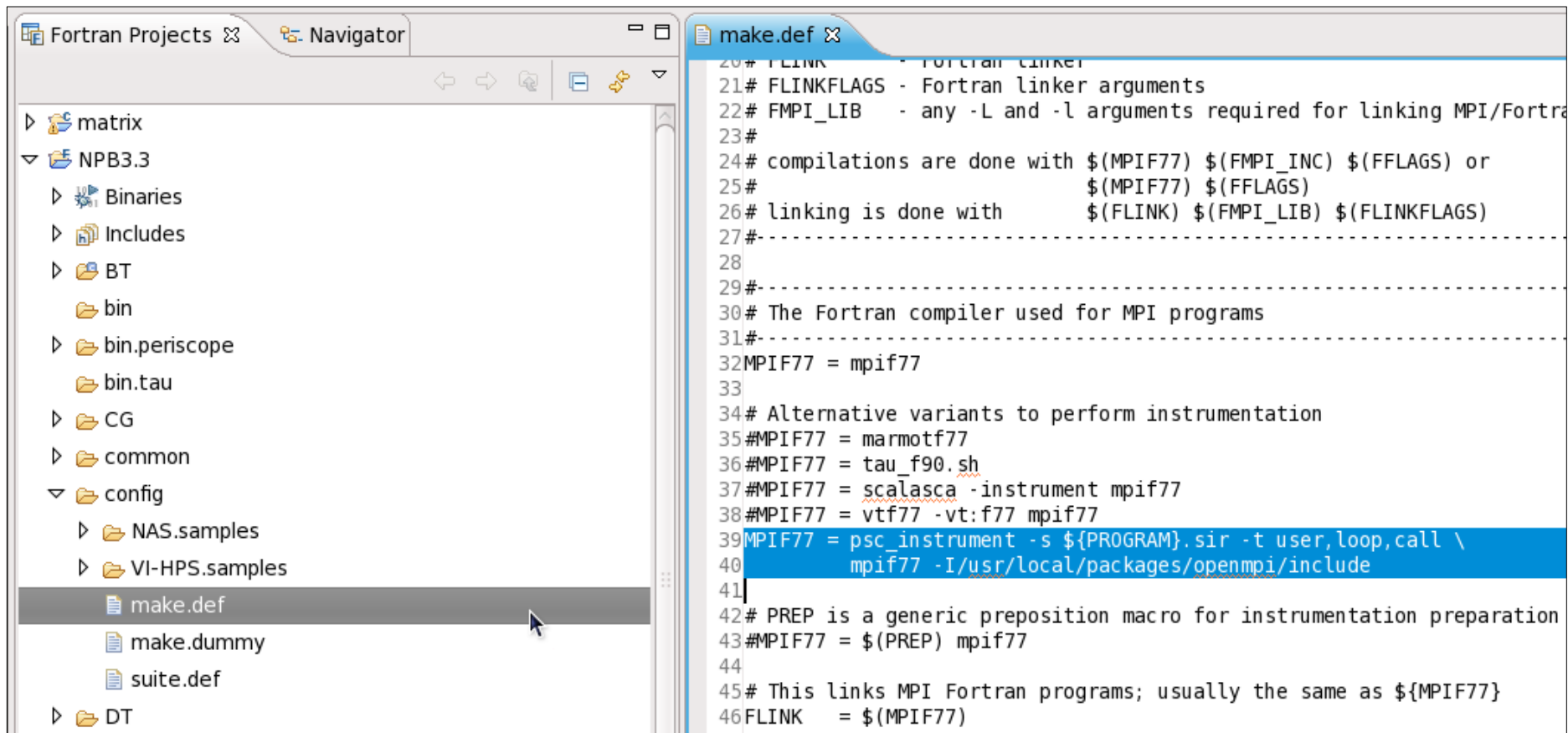
```
% eclipse
```



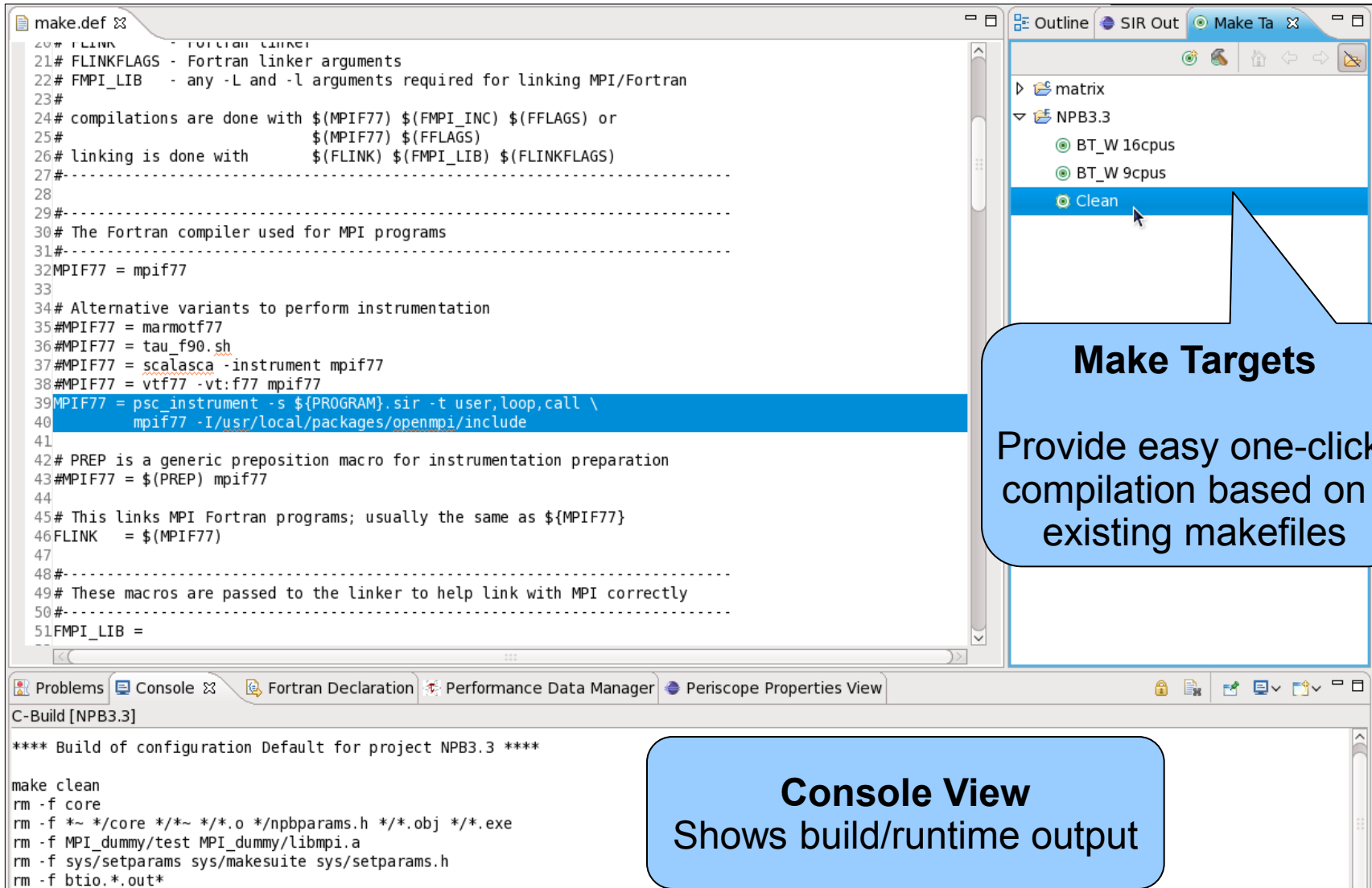
- Start it directly from Eclipse:
Run → External Tools → Periscope Registry



- Uncomment the compiler command for Periscope in `config/make.def` (the one with `psc_instrument`)



- Clean old compilation files using the **Clean** make target.



The screenshot displays a development environment with two main panels. The top panel shows a file named `make.def` with Fortran compiler settings. The bottom panel shows the `Console` view with build output for project `NPB3.3`.

make.def

```
20# FLINK - Fortran linker
21# FLINKFLAGS - Fortran linker arguments
22# FMPI_LIB - any -L and -l arguments required for linking MPI/Fortran
23#
24# compilations are done with $(MPIF77) $(FMPI_INC) $(FFLAGS) or
25#                               $(MPIF77) $(FFLAGS)
26# linking is done with          $(FLINK) $(FMPI_LIB) $(FLINKFLAGS)
27#-----
28
29#-----
30# The Fortran compiler used for MPI programs
31#-----
32MPIF77 = mpif77
33
34# Alternative variants to perform instrumentation
35#MPIF77 = marmotf77
36#MPIF77 = tau_f90.sh
37#MPIF77 = scalasca -instrument mpif77
38#MPIF77 = vtf77 -vt:f77 mpif77
39MPIF77 = psc_instrument -s ${PROGRAM}.sir -t user,loop,call \
40    mpif77 -I/usr/local/packages/openmpi/include
41
42# PREP is a generic preposition macro for instrumentation preparation
43#MPIF77 = $(PREP) mpif77
44
45# This links MPI Fortran programs; usually the same as ${MPIF77}
46FLINK = $(MPIF77)
47
48#-----
49# These macros are passed to the linker to help link with MPI correctly
50#-----
51FMPI_LIB =
--
```

Make Targets

Provide easy one-click compilation based on existing makefiles

Console View

Shows build/runtime output

Console Output:

```
**** Build of configuration Default for project NPB3.3 ****

make clean
rm -f core
rm -f *~ */core */*~ */*.o */npbparams.h */*.obj */*.exe
rm -f MPI_dummy/test MPI_dummy/libmpi.a
rm -f sys/setparams sys/makesuite sys/setparams.h
rm -f btio.*.out*
```

- Re-build BT using the *BT_W 16cpus* make target
- Check the compilation output in the *Console* view

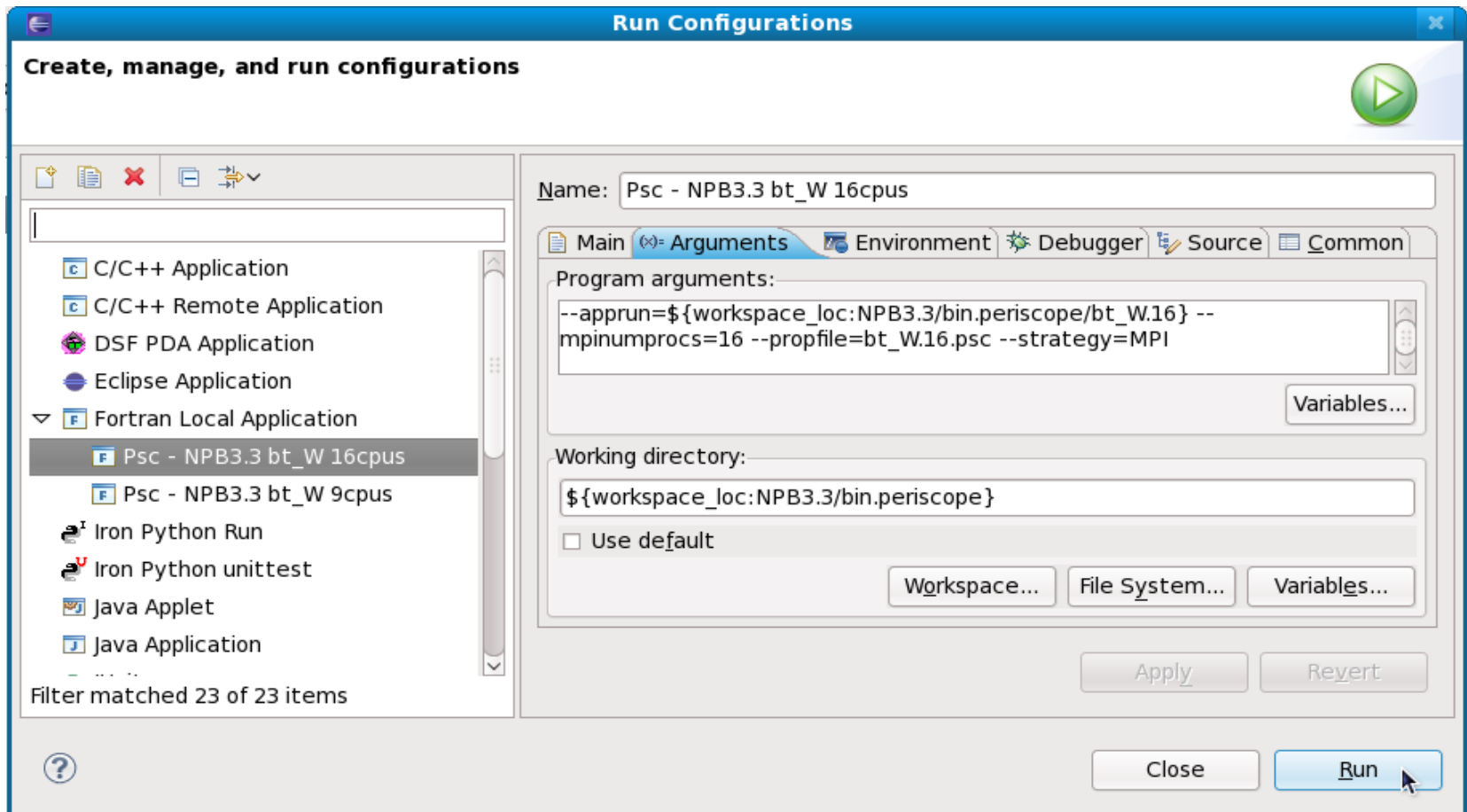
```
**** Build of configuration Default for project NPB3.3 ****

make bt CLASS=W NPROCS=16
=====
=      NAS Parallel Benchmarks 3.3      =
=      MPI/F77/C                        =
=====
cd BT; make NPROCS=16 CLASS=W SUBTYPE= VERSION=
make[1]: Entering directory `BT'
[...]
```

psc_instrument -s bt.sir -t user,loop,call mpif77 -c -O -g bt.f
psc_instrument -s bt.sir -t user,loop,call mpif77 -c -O -g make_set.f
[...]

psc_instrument -s bt.sir -t user,loop,call mpif77 -O \
-o ../bin.periscope/bt_W.16 bt.o
Built executable ../bin.periscope/bt_W.16
make[1]: Leaving directory `BT'

- Open the Eclipse Run Configurations:
Run → Run Configurations...
- Start the analysis using the profile for *BT_W* on *16 cpus*




- Check the analysis output in the **Console** view:

```
Periscope Performance Analysis Tool
[psc_frontend][INFO:fe] Preparing to start the performance analysis...
[psc_frontend][INFO:fe] Starting application ./bt_W.16 using 16 MPI procs
[psc_frontend][INFO:fe] Starting agents network...
[psc_frontend][DBG0:fe] Agent network UP and RUNNING. Starting search.

NAS Parallel Benchmarks 3.3 -- BT Benchmark
No input file inputbt.data. Using compiled defaults
Size:   24x  24x  24
Iterations: 200      dt:  0.0008000
Number of active processes:    16
[...]
Time step 200
BT Benchmark Completed.

[psc_frontend][INFO:fe] Exporting results to properties.psc
-----
End Periscope run! Search took 60.5 seconds (33.1 seconds for startup)
```

Keep in mind this number!

A blue arrow points from the text "Keep in mind this number!" to the text "Time step 200" in the console output.

- Refresh the `bin.periscope/` folder to check for the results: Select the folder → Right-click → *Refresh*

Loading the detected bottlenecks



The screenshot shows the Fortran Pro IDE interface. The left sidebar displays a project tree with the following structure:

- matrix
- NPB3.3
 - Binaries
 - Includes
 - BT
 - bin
 - bin.periscope
 - bt_W.16 - [x86/le]
 - bt_W.9 - [x86/le]
 - bt_W.16.psc** (selected)
 - bt_W.16.sir
 - bt_W.9.sir
- MG
- MPI_dummy
- SP

The main editor window shows the content of `bt_W.16.psc`, which is an XML file. The text in the editor includes:

```
1<?xml version="1.0" encoding="UTF-8"?>
2<Experiment xmlns:x
3
4 <date>2010-08-11
5 <time>15:45:58</
```

A context menu is open over the selected `bt_W.16.psc` file. The menu options are:

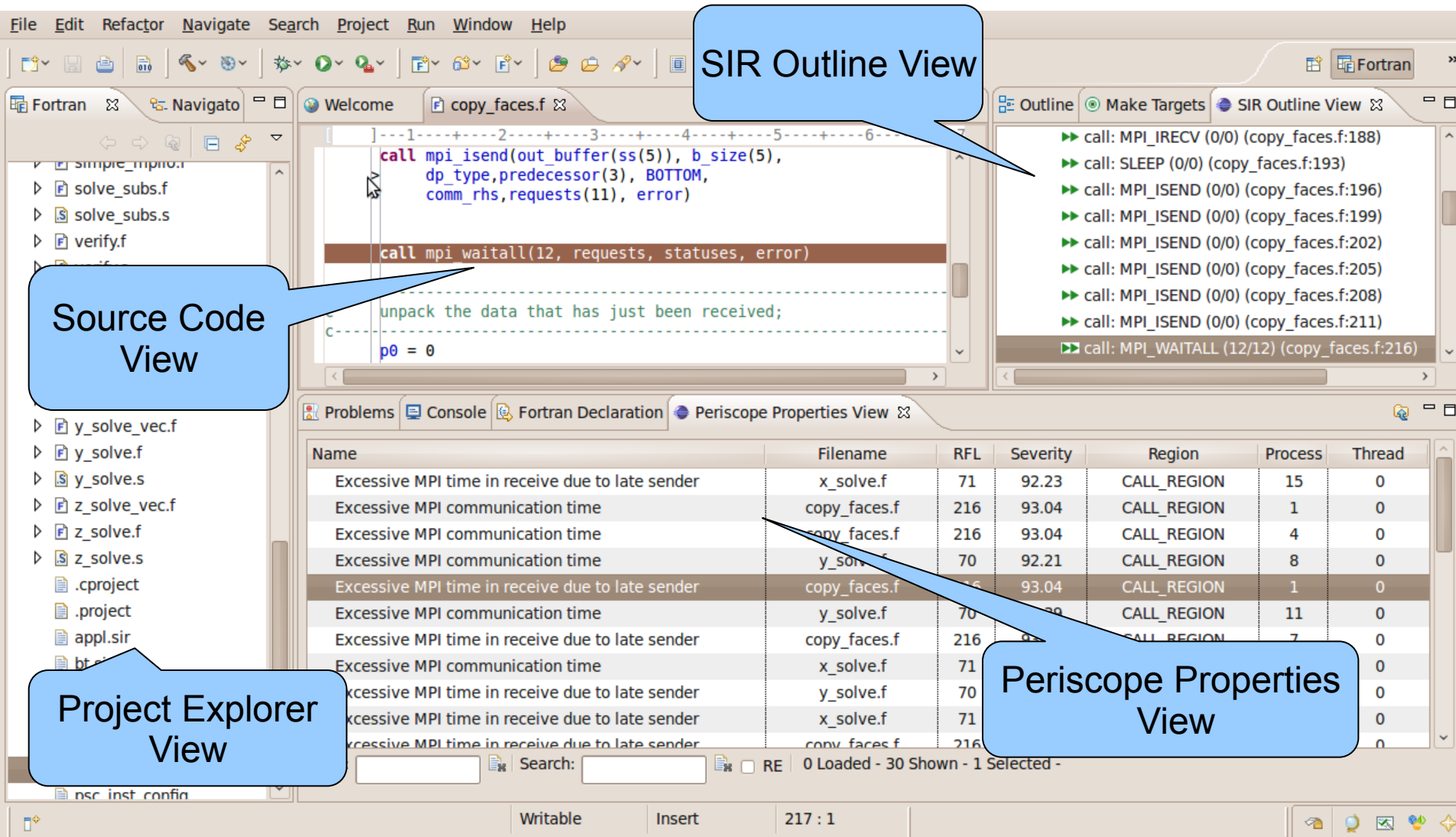
- New
- Open
- Open With
- Exclude from build...
- Build Configurations
- Make Targets
- Remove from Context (Shift+Ctrl+Alt+Down)
- Mark as Landmark (Shift+Ctrl+Alt+Up)
- Refactor
- Profile As
- Team
- Compare With
- Replace With
- Periscope** (highlighted)
- Properties (Alt+Enter)

A blue callout box with white text provides instructions:

Go to `bin.periscope/`,
search for and select `bt_W.16.psc`.
Then, right-click and choose
Periscope->Load all properties

At the bottom of the IDE, a status bar shows the current path: `NPB3.3/bin.periscope`. Below this, a list of actions is displayed:

- Load all properties (highlighted with a mouse cursor)
- Load properties above a severity
- Load and cluster properties
- Load SIR file



The screenshot displays the Periscope GUI with four main views highlighted by callouts:

- Source Code View:** Shows the Fortran source code for `copy_faces.f`. The code includes MPI-related functions like `mpi_isend` and `mpi_waitall`.
- SIR Outline View:** Displays a list of MPI calls with their respective line numbers in `copy_faces.f`, such as `call: MPI_Irecv (0/0) (copy_faces.f:188)`.
- Project Explorer View:** Shows the project structure on the left, including files like `y_solve_vec.f`, `y_solve.f`, `y_solve.s`, `z_solve_vec.f`, `z_solve.f`, `z_solve.s`, `.cproject`, `.project`, `appl.sir`, and `bt.c`.
- Periscope Properties View:** A table showing performance metrics for various MPI calls.

Name	Filename	RFL	Severity	Region	Process	Thread
Excessive MPI time in receive due to late sender	x_solve.f	71	92.23	CALL_REGION	15	0
Excessive MPI communication time	copy_faces.f	216	93.04	CALL_REGION	1	0
Excessive MPI communication time	copy_faces.f	216	93.04	CALL_REGION	4	0
Excessive MPI communication time	y_solve.f	70	92.21	CALL_REGION	8	0
Excessive MPI time in receive due to late sender	copy_faces.f	216	93.04	CALL_REGION	1	0
Excessive MPI communication time	y_solve.f	70	92.21	CALL_REGION	11	0
Excessive MPI time in receive due to late sender	copy_faces.f	216	93.04	CALL_REGION	7	0
Excessive MPI communication time	x_solve.f	71	92.23	CALL_REGION	0	0
Excessive MPI time in receive due to late sender	y_solve.f	70	92.21	CALL_REGION	0	0
Excessive MPI time in receive due to late sender	x_solve.f	71	92.23	CALL_REGION	0	0
Excessive MPI time in receive due to late sender	copy_faces.f	216	93.04	CALL_REGION	0	0

At the bottom of the Periscope Properties View, there is a search bar and a status bar indicating "0 Loaded - 30 Shown - 1 Selected".

- **Properties View:** Multi-functional table for the visualization of bottlenecks
 - Multiple criteria sorting algorithm
 - Complex categorization utility
 - Searching engine using Regular Expressions
 - Filtering operations
 - Direct navigation from the bottlenecks to their precise source location using the default IDE editor for that source file type (e.g. CDT/Photran editor).
- **SIR Outline View:** Shows a combination of the standard intermediate representation (SIR) of the analysed application and the distribution of its bottlenecks. The main goals of this view are to assist the navigation in the source code and attract developer's attention to the most problematic code areas.

- Properties view
 - Sort by severity
 - Double click on the property name takes you to the source code
- Outline view
 - Hide empty regions
 - Double click on the region takes you to the region and filters the properties for this region in the properties view
- Properties view
 - Clear all filters
 - Group by regions
 - Cluster

- Periscope performs multiple iterative performance measurement experiments on the basis of *Phases*:
 - All measurements are performed inside phase
 - Begin and end of phase are global synchronization points
- By default phase is the whole program
 - Needs restart if multiple experiments required (single core performance analysis strategies require multiple experiments)
 - Unnecessary code parts also measured
- User specified region in Fortran files that is marked with `!$MON USER REGION` and `!$MON END USER REGION` will be used as phase:
 - Typically main loop of application → no need for restart, faster analysis
 - Unnecessary code parts are not measured → less measurements overhead
 - Severity value is normalized on the main loop iteration time → more precise performance impact estimation



Defining a User Region in BT



The screenshot shows the Fortran IDE with the file `bt.f` open. The code editor displays the following Fortran code:

```
194
195
196
197      write(6, 200) step
198      format(' Time step ', i4)
199      call flush(6)
200
201      endif
202
203  !$MON USER REGION
204      call adi
205  !$MON END USER REGION
206
207      if (iotype .ne. 0) then
208      call timer_start(2)
209      if (mod(step, wr_interv)
210      if (node .eq. root)
211      print *, 'Wri
212
213      step .eq. nit
214      d_interval =
215      output times
```

Three callouts provide instructions for defining the user region:

1. Search for "bt.f" and double-click
2. Go to line 203 (CTRL+L) and surround "call adi" with
`!$MON USER REGION`
`!$MON END USER REGION`
3. Save file (CTRL+S)

The IDE interface includes a menu bar (File, Edit, Refactor, Navigate, Search, Project, Run, Window, Help), a toolbar, a project explorer on the left, a main code editor, a right-hand pane showing the project structure, and a bottom status bar with a filter, search, and process/thread information.

- Clean and rebuild the BT executable after saving your modifications
- Re-run Periscope analysis as before using the Run Configuration

```
Periscope Performance Analysis Tool
```

```
NAS Parallel Benchmarks 3.3 -- BT Benchmark
```

```
[...]
```

```
Time step 1
```

```
BT Benchmark Completed.
```

Do you
remember this?

```
[psc_frontend][INFO:fe] Exporting results to properties.psc
```

```
-----
```

```
End Periscope run! Search took 35.2 seconds (33.3 seconds for startup)
```

- Only **1** iteration of BT is now required instead of the 200 in the previous run!
- The analysis time is decreased but the quality of the results stays the same!

- Periscope components
 - Frontend and high-level agents execute on the Bluegene frontend.
 - Analysis agents run on the IO nodes. The tool startup support in mpirun is used.
- Eclipse can be used on the frontend.
- No automatic restart for multi-phase strategies.



Feedback / Questions?

Contact us:
periscope@lrr.in.tum.de