



SOFTWARE

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



PRODUCTIVITY

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  
Technische Universität München  
periscope@in.tum.de  
March 2012

- Intermediate-level tutorial example
- Available in MPI, OpenMP, hybrid OpenMP/MPI variants
- Automatic performance properties search with Periscope:
  - Source code instrumentation
    - ▶ MPI calls
  - Automatic search for slow MPI communication patterns
  - Results exploration with Eclipse based GUI
- Manual instrumentation optimization

0. Configuration of Periscope
1. Program instrumentation: `psc_instrument`
2. Periscope analysis: `psc_frontend`
3. Performance properties exploration: `Periscope GUI`

- 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      = Levque           //hostname
SITE         = CMM
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
```

- The Periscope agents and the application processes register with a **registry**. The registry is started via:

```
% psc_regsrv &
```

- To enable performance measurement, the program has to be instrumented. This is done with **psc\_instrument**:

```
% psc_instrument
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 spaces
      (e.g. -t "user loop call")
  -s Filename for the resulting SIR file (default: appl.sir)
  -v Verbose output
  -d Debug mode: keeps the instrumented source files
      after the compilation
  -n Prints each step of the compilation instead of executing them
```

- Substitute compile/link commands in Makefile definitions (config/make.def) with `psc_instrument`:

```
MPIF77 = psc_instrument -s ${PROGRAM}.sir -t user,mpi mpif77

FLINK = $(MPIF77)
FFLAGS = -O

mpi-bt: $(OBJECTS)
    $(FLINK) $(FFLAGS) -o mpi-bt $(OBJECTS)

.f.o:
    $(MPIF77) $(FFLAGS) -c $<
```

- Return to root directory and clean-up

```
% make clean
```

- Re-build BT with the original command

```
% 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 -0 -g bt.f
psc_instrument -s bt.sir -t "user loop call" mpif77 -c -0 -g make_set.f
...
psc_instrument -s bt.sir -t "user loop call" mpif77 -0 \
-o ../bin.periscope/bt_W.16 bt.o ...
Built executable ../bin.periscope/bt_W.16
make[1]: Leaving directory `BT'
```

- Change directory to bin.periscope

```
% cd bin.periscope
```

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

```
% psc_frontend --help
Usage: psc_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)
  [--delay=n]             (search delay in phase executions)
  [--appname=name]
  [--apprun=commandline]
  [--mpinumprocs=number of MPI processes]
  [--ompnumthreads=number of OpenMP threads]
...
  [--strategy=name]
  [--sir=name]
  [--phase=(FileID,RFL)]
  [--debug=level]
```



- Run Periscope analysis by executing `psc_frontend` with the following command

```
% psc_frontend --sir=bt_W.16.sir --apprun=./bt_W.16 --strategy=MPI
--mpinumprocs=16 --force-localhost
[psc_frontend][DBG0:fe] Agent network UP and RUNNING. Starting search.

NAS Parallel Benchmarks 3.3 -- BT Benchmark
[...]
Time step 200
BT Benchmark Completed.

-----
End Periscope run! Search took 60.5 seconds (33.3 seconds for startup)
```

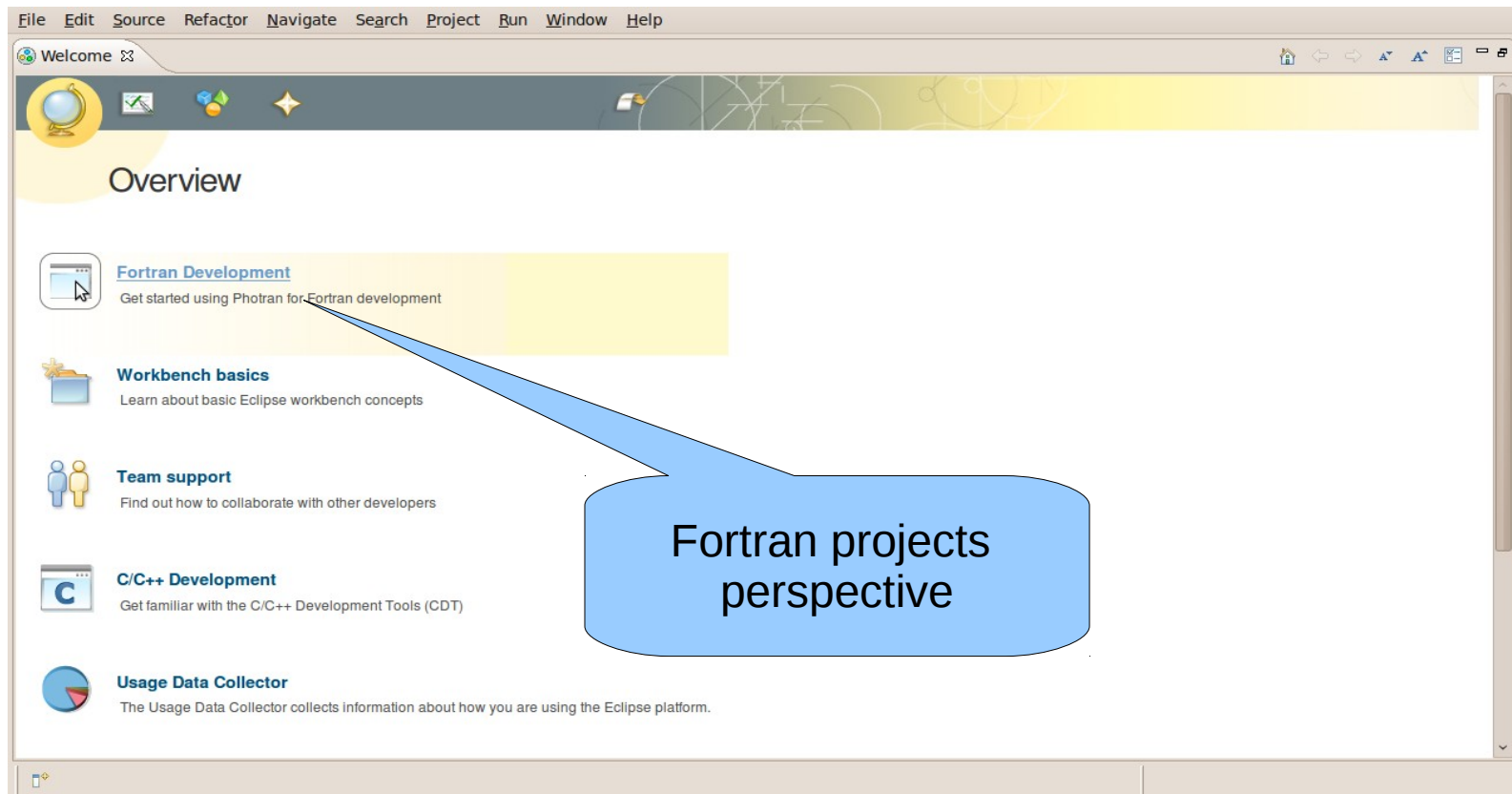
- Frontend will write the detected properties into the file `properties_MPI_*.psc` in the current directory. It should be copied into the BT source directory

```
% cp properties.psc ../BT
```

- Start **Eclipse** with Periscope GUI from console

```
% eclipse
```

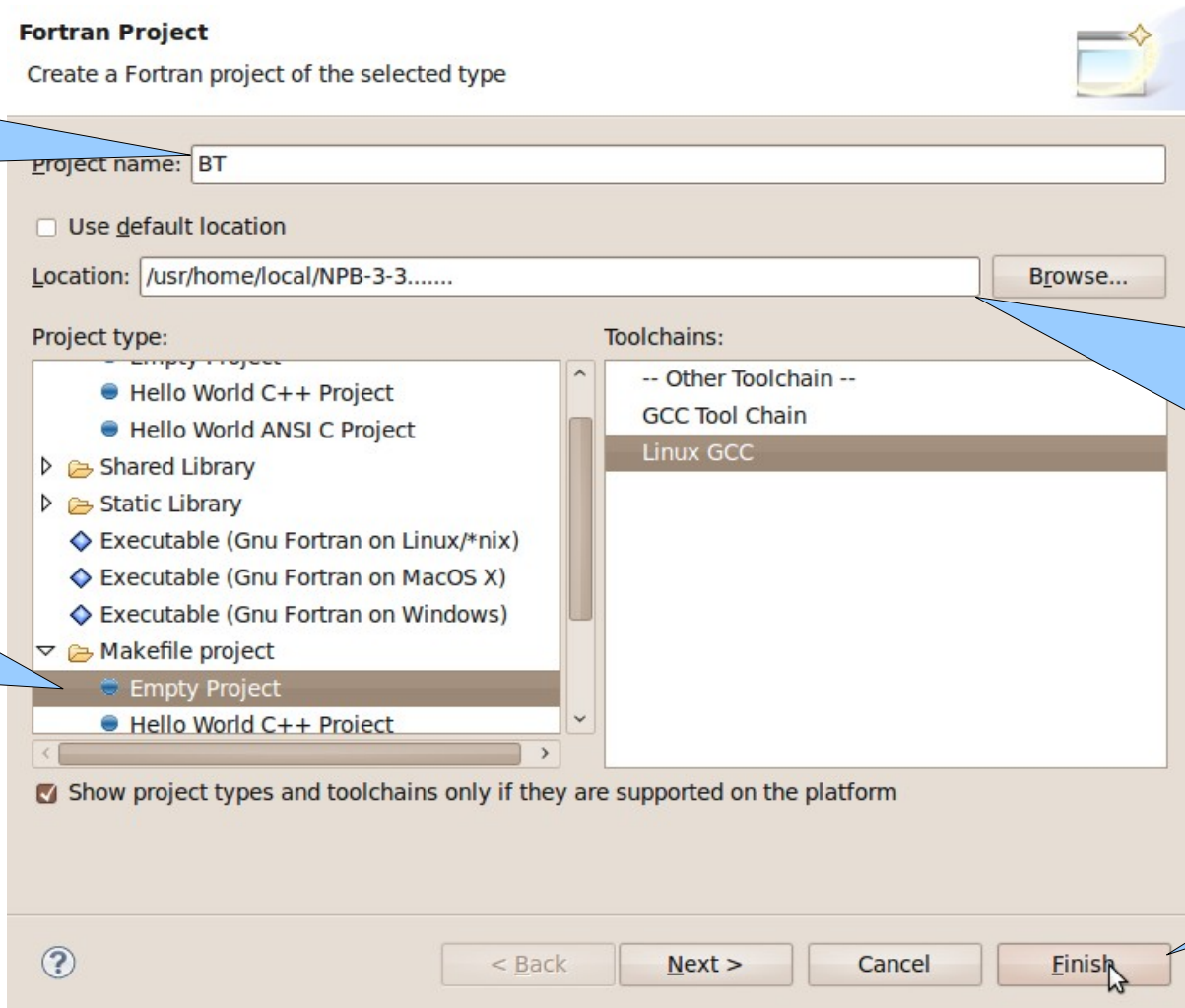
- Or by double-click on Eclipse pictogram on the Desktop



- File->New->Project... → Fortran->Fortran Project

Input project name

Project type



Unmark "Use default location" and provide path to BT folder

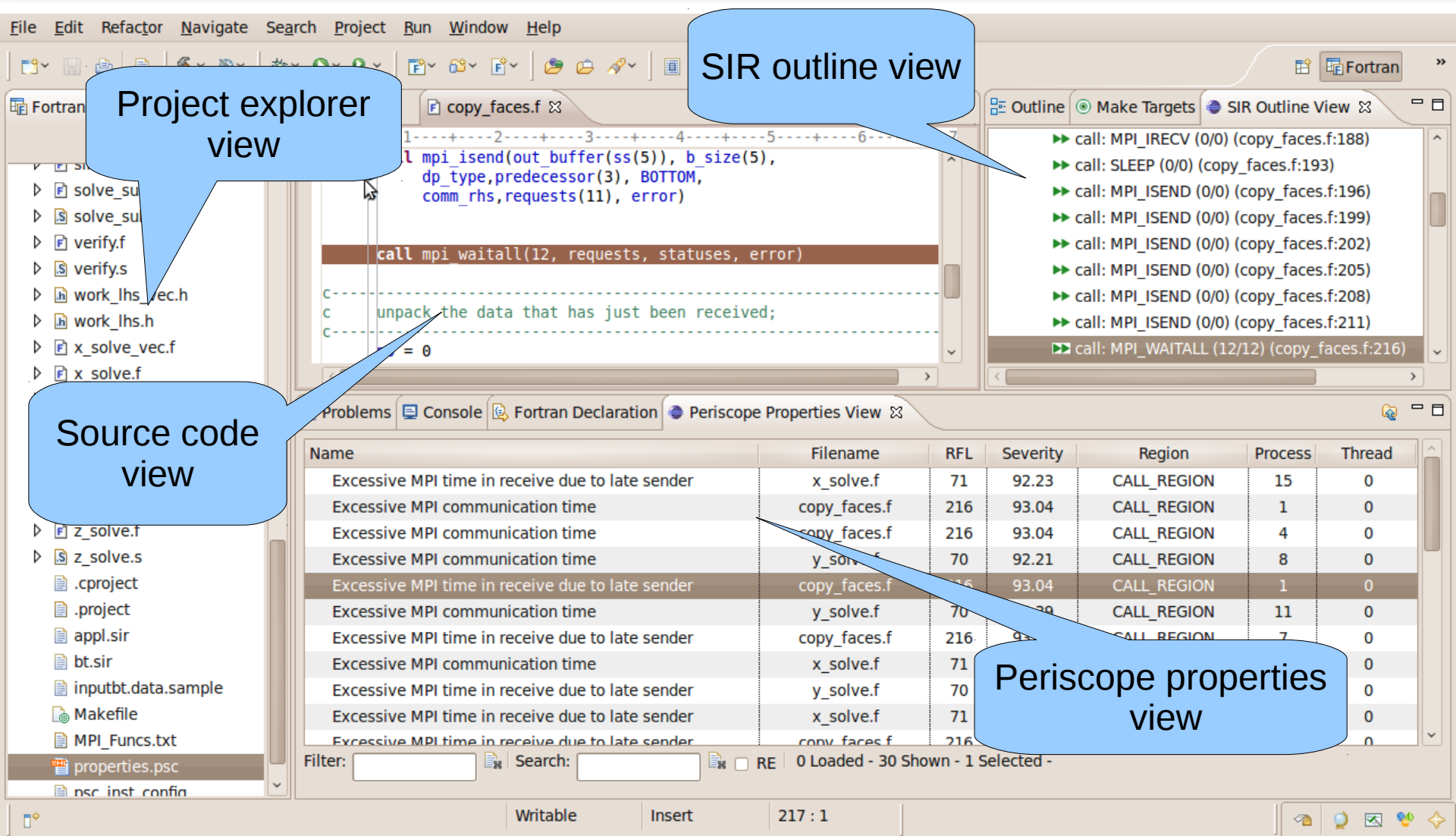
Press Finish

# Loading properties



A screenshot of the Eclipse IDE interface. The top menu bar includes File, Edit, Refactor, Navigate, Search, Project, Run, Window, and Help. The main window displays the Eclipse logo and the text "Welcome to Photran 5.0!". Below this, there are links for "User's Guide", "Photran mailing List", "ptp-announce mailing List", and "Frequently Asked Questions". The bottom of the page has the copyright notice "COPYRIGHT © 2009 THE ECLIPSE FOUNDATION. ALL RIGHTS RESERVED.". On the left, the "Navigator" view shows a project tree with files like solve\_subs.f, verify.f, work\_lhs\_vec.h, x\_solve\_vec.f, y\_solve\_vec.f, z\_solve\_vec.f, .project, .project, appl.sir, bt.sir, inputbt.data.sample, Makefile, MPI\_Funcs.txt, properties.psc, and nsc\_inst.conf. A blue speech bubble is overlaid on the bottom left, containing the text: "Expand BT project, search for properties.psc and Right click-&gt;Periscope-&gt; Load all properties".

Expand BT project,  
search for properties.psc  
and  
Right click->Periscope->  
Load all properties



Project explorer view

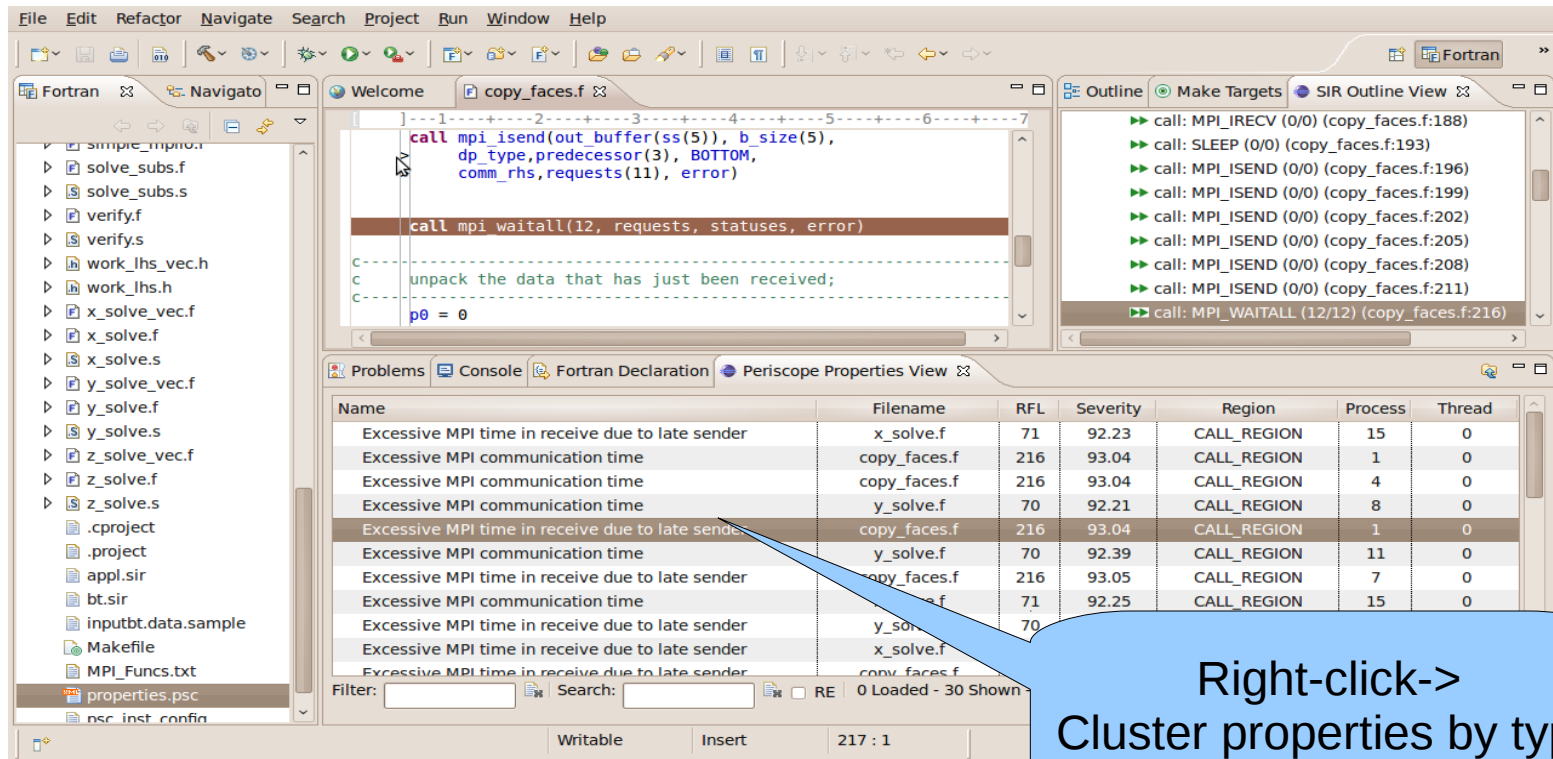
SIR outline view

Source code view

Periscope properties view

- Multi-functional table is used in the GUI for Eclipse 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.

- Clustering can effectively summarize displayed properties and identify a similar performance behavior possibly hidden in the large amount of data



The screenshot shows an IDE window with a table of performance properties. The table has columns for Name, Filename, RFL, Severity, Region, Process, and Thread. A callout box points to the table with the text: "Right-click-> Cluster properties by type".

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	92.39	CALL_REGION	11	0
Excessive MPI time in receive due to late sender	copy_faces.f	216	93.05	CALL_REGION	7	0
Excessive MPI communication time	copy_faces.f	71	92.25	CALL_REGION	15	0
Excessive MPI time in receive due to late sender	y_solve.f	70				
Excessive MPI time in receive due to late sender	x_solve.f					
Excessive MPI time in receive due to late sender	copy_faces.f					

# Properties clustering



File Edit Refactor Navigate Search Project Run Window Help

Problems Console Fortran Declaration Periscope Properties View Clustering Results View

Name	Filename	RFL	Severity	Confidence	Processes	Threads	Clustering Error
call: MPI_WAIT (8) (y_solve.f:70)	y_solve.f	70	92.35	1.00	Regions Group		
Excessive MPI time in receive due to late send					Types Group		Clustering squared error: 0.13/0.50
Cluster 1					8 9		
Cluster 2					10 11		
Excessive MPI communication time (4)					Types Group		Clustering squared error: 0.17/0.50
Cluster 1	y_solve.f		92.45		10 11		
Cluster 2	y_solve.f	70	92.28		8 9		
call: MPI_WAITALL (12) (copy_faces.f:216)	copy_faces.f	216	93.01	1.00	Regions Group		
Excessive MPI time in receive due to late send	copy_faces.f	216			Types Group		Clustering squared error: 0.11/0.50
Cluster 1	copy_faces.f	216	92.98		3 12 13		
Cluster 2	copy_faces.f	216	93.04		1 7		
Excessive MPI communication time (6)	copy_faces.f	216			Types Group		Clustering squared error: 0.11/0.50
Cluster 1	copy_faces.f	216	92.98		3 1		
Cluster 2	copy_faces.f	216	93.04		1 4		
call: MPI_WAIT (1) (x_solve.f:71)	x_solve.f	71	92.40	1.00	Regions		
Excessive MPI time in receive due to late send	x_solve.f	71			Types		Clustering squared error: 0.12/0.50
Cluster 1	x_solve.f	71	92.60		14		
Cluster 2	x_solve.f	71	92.34		2 5 6		
Excessive MPI communication time (6)	x_solve.f	71			Types		Clustering squared error: 0.13/0.50
Cluster 1	x_solve.f	71	92.36				
Cluster 2	x_solve.f	71	92.62				

Filter: Search: RE 0 Loaded - 21 Shown - 1 Selected -

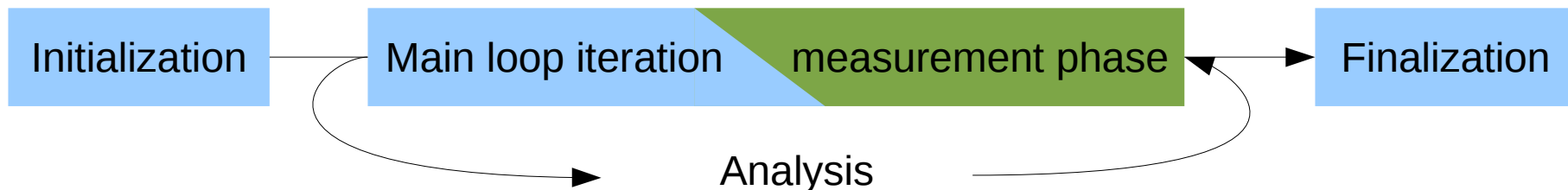
Severity value of the Cluster 1

Region and property where clustering performed

Processes belonging To the Cluster1



- 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 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



File Edit Refactor Navigate Search Project Run Window Help

Fortran Welcome copy\_faces.f \*bt.f

194  
195  
196  
197  
198 200  
199  
200  
201  
202  
203 !\$MON USER REGION  
204  
205 !\$MON END USER REGION  
206  
207  
208  
209  
210

0 .or. step .eq. niter .or.  
write(6, 200) step  
format(' Time step ', i4)  
call flush(6)  
endif  
endif  
call adi  
endif  
if (iotype .ne. 0) then  
call timer\_start(2)  
if (mod(step, wr\_interval) .eq. 0) then  
if (node .eq. root) then  
print \*, 'Wri  
if (step .eq. niter) then  
d\_interval =  
output times

3. Save file (^S)

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

Name  
call: MPI\_WAIT (8) (y\_solve.f:70) y\_solve.f 70 92.35 1.00 Regions Group

Filter: Search: RE 0 Loaded - 21 Shown - 1 Selected -

Writable Insert 205 : 22

- Return to root directory and clean-up

```
% make clean
```

- Re-build BT with the original command

```
% make bt CLASS=W NPROCS=16
```

- Change directory into location of executable

```
% cd bin.periscope
```

- Re-run Periscope analysis by executing `psc_frontend`

```
% psc_frontend --sir=bt_W.16.sir --apprun=./bt_W.16 --strategy=MPI
--mpinumprocs=16 --force-localhost
[psc_frontend][DBG0:fe] Agent network UP and RUNNING. Starting search.

NAS Parallel Benchmarks 3.3 -- BT Benchmark
[...]
Time step 1
BT Benchmark Completed.

-----
End Periscope run! Search took 37.2 seconds (33.3 seconds for startup)
```

- Only 1 iteration of BT required instead of 200 previous run!
- Frontend will overwrite the properties found into the file `properties.psc` in the current directory, which again need to be copied into the BT source directory

```
% cp properties.psc ../BT
```

- Re-load `properties.psc` in Periscope GUI. Now found properties should have more precise severities values