

Performance Analysis with Periscope

M. Gerndt, V. Petkov, Y. Oleynik
Technische Universität München

periscope@lrr.in.tum.de

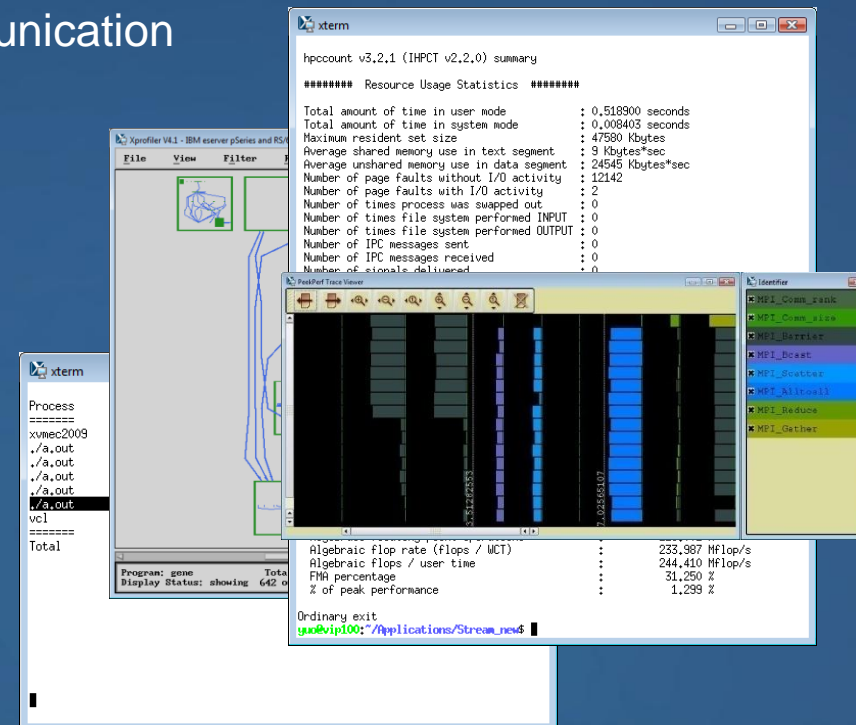
September 2011

Outline

- Motivation
- Periscope overview
- Periscope performance analysis model
- Performance analysis automation
- Periscope GUI

Motivation

- Performance analysis procedure on POWER6 as a example:
 - Use Tprof to pinpoint time consuming subroutines
 - Use Xprofiler (GUI for gprof) to understand call graph
 - Use hpmcount (libhpm) to measure Hardware Counters
 - Use mpitrace to investigate mpi communication
- Problems:
 - Time consuming
 - Error prone
 - Not scalable
 - Requires deep hardware knowledge
- Solution:
 - Performance analysis automation



Periscope

- **Iterative online analysis**
 - Measurements are configured, obtained and evaluated on the fly
 - no tracing!
- **Distributed architecture**
 - Analysis performed by multiple distributed hierarchical agents
- **Automatic bottlenecks search**
 - Based on performance optimization experts' knowledge
- **Enhanced GUI**
 - Eclipse based integrated development and performance analysis environment
- **Instrumentation**
 - Fortran, C/C++

Distributed Architecture

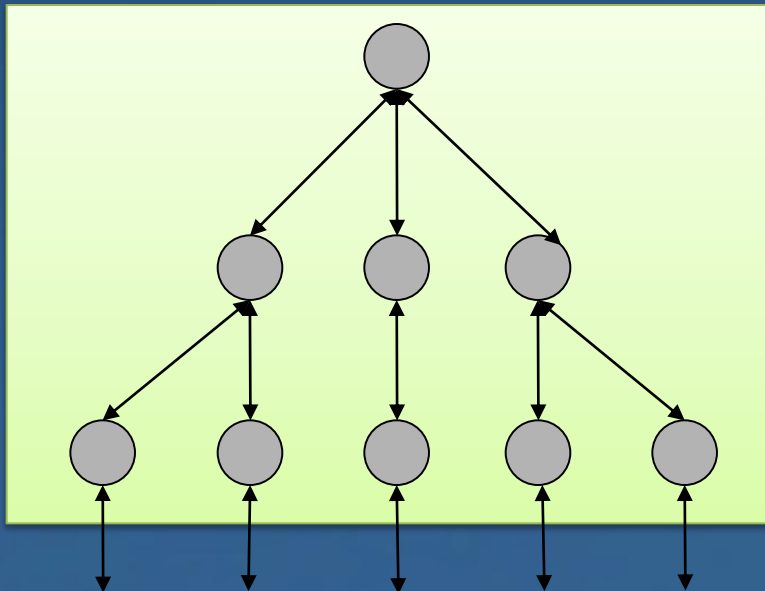
Graphical User Interface

Interactive frontend

Eclipse-based GUI

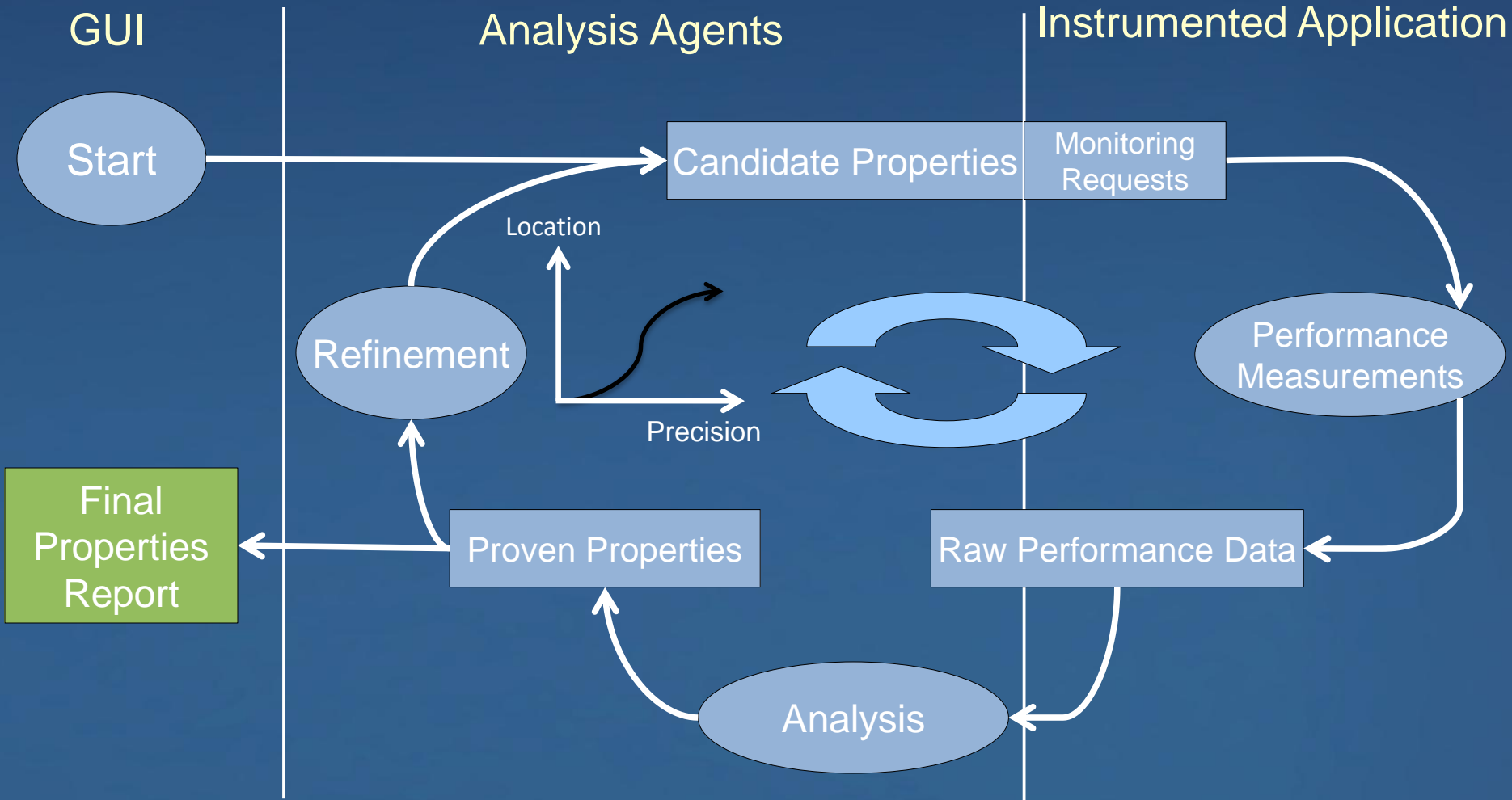
Analysis control

Agents network



Monitoring Request Interface

Application



Periscope Phases

- 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

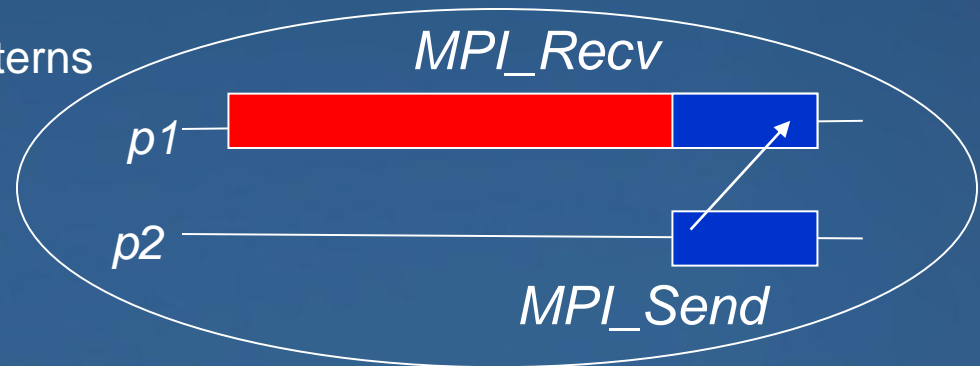


Automatic search for bottlenecks

- Automation based on formalized expert knowledge
 - Potential performance problems → properties
 - Efficient search algorithm → search strategies
- Performance property
 - Condition
 - Confidence
 - Severity
- Performance analysis strategies
 - Itanium2 Stall Cycle Analysis
 - IBM POWER6 Single Core Performance Analysis
 - MPI Communication Pattern Analysis
 - Generic Memory Strategy
 - OpenMP-based Performance Analysis
 - Scalability Analysis – OpenMP codes

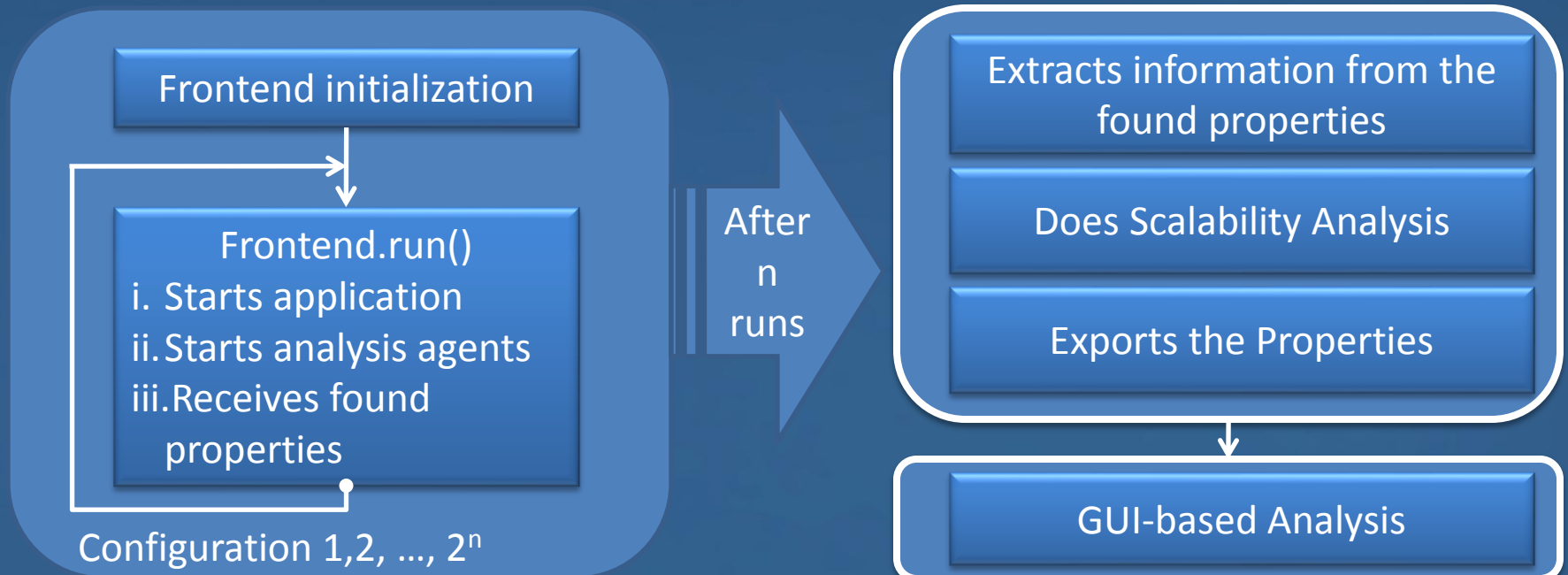
Example Properties

- **StallCycles (Region, Rank, Thread, Metric, Phase)**
 - Condition
 - Percentage of lost cycles >30%
 - Severity
 - Percentage of lost cycles
- **MPI Late Sender**
 - Automatic detection of wait patterns
 - Measurement on the fly
 - No tracing required!
- **OpenMP Synchronization properties**
 - Critical section overhead property
 - Frequent atomic property



Scalability Analysis – OpenMP codes

- Identifies the OpenMP code regions that do not scale well
- Scalability Analysis is done by the frontend / restarts the application /
- **No need to manually configure the runs and find the speedup!**



The screenshot shows the Eclipse IDE interface for a Fortran project. The main editor displays the source code of `time_scheme.F90`. The left sidebar shows the Project view with a tree structure of files and folders. The right sidebar shows the SIR outline view, which lists various subroutines and their call counts. The bottom panel shows the Properties view, which displays performance-related data for the selected code region.

```
100  endif
101
102  !-----
103  !ALL OTHER STAGES
104  !result is accumulated in k1
105  do stage=2,rkstages
106    if (stage.eq.2) then
107      call nextstage_g(a_rk(stage)*dt,k1,g_1,tg)
108    else
109      call nextstage_g(a_rk(stage)*dt,k2,g_1,tg)
110    endif
111
112    call calc_rest(tg,emfields,f_)
113
114    call CalcFullRhs(f_,tg,emfields,k2,.false.)
115
116    call add_ks(b_rk(stage)/b_rk(1),k2,k1)
117  enddo
118
119  !-----
120  !RESULT OF TIMESTEPPING, new g
121  call add_ks(b_rk(1)*dt,k1,g_1)
122
123  !all other quantities
124  call calc_rest(g_1,emfields,f_)
125
126  Call perffoff
127  DEBUG(1,"==== END  explicit time scheme ====")
128  End Subroutine calc_explicit_timestep
129  !===== End Runge-Kutta =====
```

Project view

- Fortran Projects
- Navigator
- time_scheme.F90
- gene.F90
- Obj
- PRE
- 128cpus.out
- appl
- appl.sir
- aux_fields.F90
- aux_func.F90
- boundary.F90
- calc_rhs_helper.F90
- calc_rhs.F90
- chease.F90
- checkpoint.F90
- circular.F90
- collisions.F90
- comm.F90
- debug_output.F90
- diag.F90
- dummyperf.F90
- eigenvalues.F90
- erf.f90
- erf.o
- field_solve_kxky.F90
- field_solve_xky.F90
- fourier_acml.F90
- fourier_essl.F90
- fourier_fftw.F90
- fourier_mkl.F90

SIR outline view

- SIR File
- program: GENE (0/1790) (./src/gene.
- userRegion: (608/1790) (./src/ger
- call: GET_SYSTIME (32/32) (./st
- call: CALC_EXPLICIT_TIMESTEP
- call: DIAG_NRG (32/32) (./src/g
- call: DIAG_OMEGA (32/32) (./st
- call: DIAG_FIELD (32/32) (./src/
- call: DIAG_MOM (32/32) (./src/
- call: DIAG_VSP (32/32) (./src/g
- call: DIAG_FIELD (32/32) (./src/
- call: DIAG_MOM (32/32) (./src/
- call: DIAG_ (32) (./src/g
- call: CHECK
- call: CH
- call: RE
- call: DI
- call: DI
- call: DIAG_OMEGA (32/32) (./st
- subroutine: CALC_EXPLICIT_TIMESTE
- loop: (517/517) (./src/time_schen

Properties view

Name	Filename	RFL	Severity	Region	Process	Thread
Stalls due to hardware page walker			9.03	Types Group		
Stalls due to L2DTLB to L1DTLB transfer			0.00	Types Group		
IA64 Pipeline Stall Cycles			24.10	Types Group		
Stalls due to full store buffer			0.20	Types Group		
Stalls due to full store buffer	./src/time_scheme.F90	105	0.24	LOOP_REGION	4	0
Stalls due to full store buffer	./src/time_scheme.F90	105	0.21	LOOP_REGION	13	0
Stalls due to full store buffer	./src/time_scheme.F90	105	0.18	LOOP_REGION	3	0

Thank you for your attention!

- Current version 1.4
 - Available under: <http://www.lrr.in.tum.de/periscope/Download>
- Supported architectures
 - SGI Altix 4700 Itanium2
 - IBM Power575 POWER6
 - IBM BlueGene/P
 - x86-based architectures
- Further information:
 - Periscope web page: <http://www.lrr.in.tum.de/periscope>
 - Contact us directly at: periscope@lrr.in.tum.de