Performance Analysis with Periscope

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

periscope@lrr.in.tum.de

April 2012
Outline

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

• Performance analysis procedure on POWER6 as an 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 mpirace to investigate mpi communication

• Problems:
  – Time consuming
  – Error prone
  – Not scalable
  – Requires deep hardware knowledge

• Solution:
  – Performance analysis automation
Periscope

- **Distributed architecture**
  - Analysis performed by multiple distributed hierarchical agents

- **Iterative online analysis**
  - Measurements are configured, obtained and evaluated on the fly
  - No tracing files needed

- **Automatic bottlenecks search**
  - Based on performance optimization experts' knowledge

- **Enhanced GUI**
  - Eclipse based integrated development and performance analysis environment

- **Instrumentation**
  - Fortran, C/C++
  - Automatic overhead control
Distributed Architecture

Graphical User Interface

Eclipse-based GUI

Interactive frontend

Analysis control

Agents network

Monitoring Request Interface

MRIMonitor/Score-P

Application
Iterative Online Analysis

GUI

Analysis Agents

Start

Refinement

Candidate Properties

Monitoring Requests

Performance Measurements

Final Properties Report

Proven Properties

Raw Performance Data

Analysis
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
  - Automatic restart might be necessary

- Region needs to me marked as an Online Access Phase to use the Score-P Online Access Interface
  - 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
Definition of Online Access Phases

```c
#include <scorep/SCOREP_User.h>
void foo()
{
    SCOREP_USER_REGION_DEFINE( my_region_handle )
    for(i=0; ... 
    {
        SCOREP_USER_OA_PHASE_BEGIN( my_region_handle, \ "foo",SCOREP_USER_REGION_TYPE_COMMON )
        // do something

        ...

        SCOREP_USER_OA_PHASE_END( my_region_handle )
    }
}
```
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
  - Westmere Single-node Analysis
  - Itanium2 Stall Cycle Analysis
  - IBM POWER6 Single Core Performance Analysis
  - MPI Communication Pattern Analysis
  - Generic Memory Analysis
  - 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!

Frontend initialization

- Frontend.run()
  - i. Starts application
  - ii. Starts analysis agents
  - iii. Receives found properties

Configuration 1, 2, ..., 2^n

After n runs

Extracts information from the found properties

Does Scalability Analysis

Exports the Properties

GUI-based Analysis
### Source code view

```fortran
333 call mpi_send(wn(1,1,1d0), LXDIM2, MPI_DOUBLE_PRECISION, 
334 ngh, 600, 
335 MPI_COMM_WORLD, error)
336 endif
337
338 if (NV.GE.0) then
339   call mpi_recv(wn(1,1,0), LXDIM2, MPI_DOUBLE_PRECISION, 
340                  MPI_ANY_SOURCE, 
341                  600, MPI_COMM_WORLD, status, error)
342 endif
343
344 nach_rechts_senden(vn)
345
346 if (NR.GE.0) then
347   call csendxs(77, LDIM3, vn(1, LDIM2, 1), 8*LX, 8*LX*(LDIM2-1), nh, 0)
348 endif
349
350 if (NL.GE.0) then
351   call crecvxs(77, LDIM3, vn(1, 0, 1), 8*LX, 8*LX*(LDIM2-1), nh, 0)
352 endif
353
```

### Project view

The project view displays a source code editor with highlighted lines indicating code changes and a call hierarchy for MPI communication routines.

### SIR outline view

The SIR outline view shows a call hierarchy with nodes labeled as subroutine names and their parameters.

### Properties view

The properties view displays a table with columns for Name, Filename, RFL, Severity, Region, and Process, indicating analysis results for MPI communication timing issues.
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/x64-based architectures

- **Further information:**
  - Periscope web page: http://www.lrr.in.tum.de/periscope
  - Contact us directly at: periscope@lrr.in.tum.de