ompP: A Profiling Tool for OpenMP

Karl Fürlinger
Michael Gerndt

{fuerling, gerndt}@in.tum.de

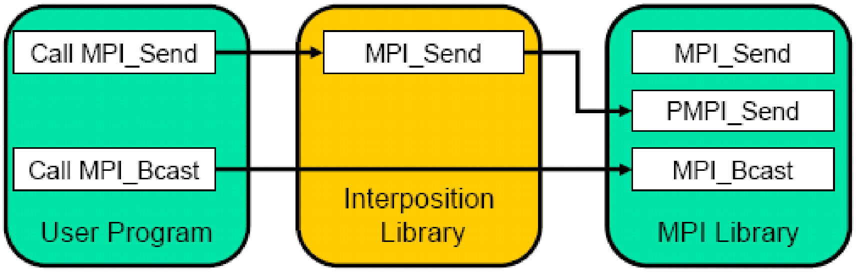
Technische Universität München

Performance Analysis of OpenMP Applications

- Platform specific tools
  - SUN Studio
  - Intel Thread Analyzer
  - ...
  - Make use of platform/compiler specific knowledge (naming conventions, outlining of parallel regions, ...)
- Platform independent tools
  - How can we obtain performance data in a portable way?
  - No standard performance measurement interface for OpenMP yet,
  - POMP proposal for such an inteface [Mohr02]
  - DMPL proposed as a debugging interface [Cownie03]

MPI Profiling Interface (MPIP)

- Wrapper interposition approach
  - Easy since MPI functionality is provided in a library
  - No recompilation necessary

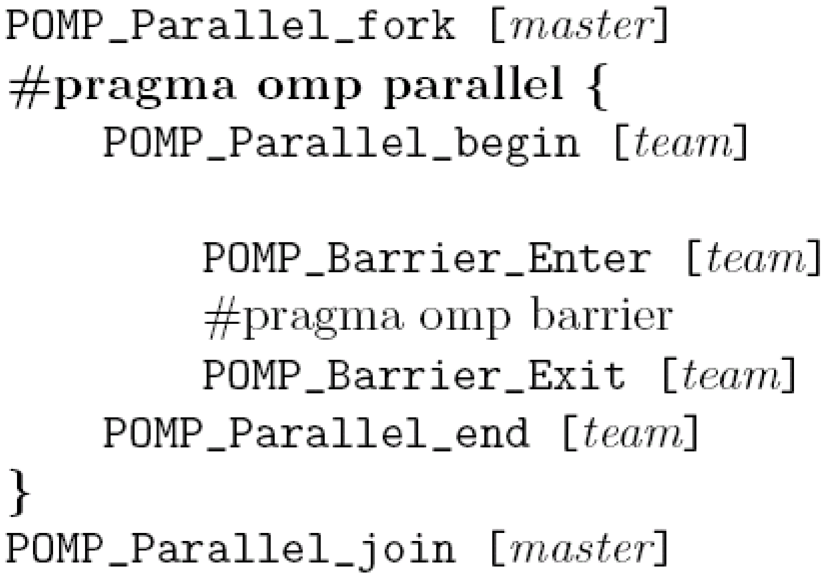


- Performance measurement libraries libraries
  - For tracing: Vampir / Intel Trace Analyzer, Paraver, ...
  - For profiling: mpiP

```
---------------------------------------------------------------------------
@--- Aggregate Sent Message Size (top twenty, descending, bytes) ----------
---------------------------------------------------------------------------
Call                 Site      Count       Total    Avrg      MPI%
Send                    7        320    1.92e+06   6e+03     99.96
Bcast                   1         12         336      28      0.02
```

OpenMP Profiling Interface (POMP)

- No standard yet, but POMP proposal by Bernd Mohr et al.
- Insert function calls in and around OpenMP constructs to expose exectution events.
- Implicit barriers added to expose load imbalances
- Example:

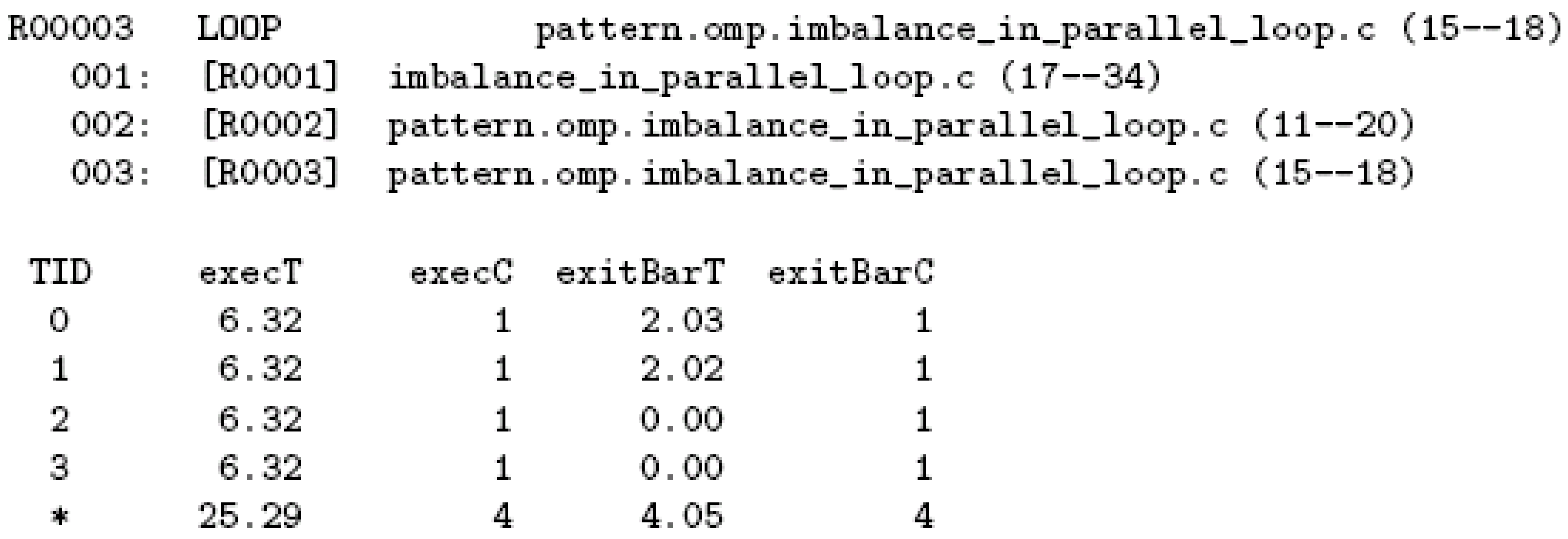


```
POMP_Parallel_fork [master]
#pragma omp parallel {
    POMP_Parallel_begin [team]

        POMP_Barrier_Enter [team]
        #pragma omp barrier
        POMP_Barrier_Exit [team]
    POMP_Parallel_end [team]
}
POMP_Parallel_join [master]
```

ompP: OpenMP Profiler

- ompP
  - Simple execution profiler for OpenMP, based on POMP instrumentation
  - Currently only *counts* and *times* are kept
  - Hardware performance counter support planned for future
  - Simple textual profiling report available immediately after execution of the target application



```
R00003   LOOP            pattern.omp.imbalance_in_parallel_loop.c (15--18)
  001:   [R0001]  imbalance_in_parallel_loop.c (17--34)
  002:   [R0002]  pattern.omp.imbalance_in_parallel_loop.c (11--20)
  003:   [R0003]  pattern.omp.imbalance_in_parallel_loop.c (15--18)

 TID      execT      execC  exitBarT  exitBarC
  0        6.32          1      2.03         1
  1        6.32          1      2.02         1
  2        6.32          1      0.00         1
  3        6.32          1      0.00         1
  *       25.29          4      4.05         4
```

ompP: Design / Implementation

- Opari creates a *region descriptor* for each identified OpenMP construct

  –
  ```
  struct ompregdescr omp_rd_1 = {
    "parallel", "", 0, "main.c", 8, 8, 11, 11
  };
  ```

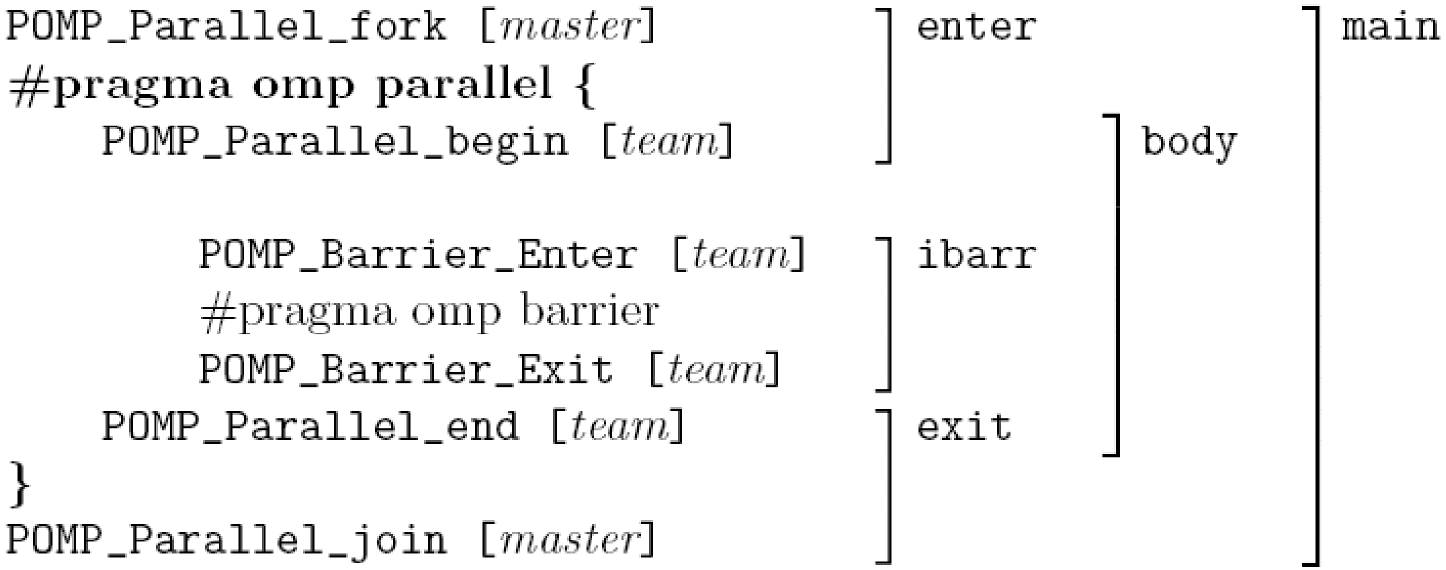
  – Descriptor passed in `POMP_*` calls, multiple different calls use same descriptor

  – Complicates performance data bookkeeping so we break down larger POMP regions into smaller „Pseudoregions“

Pseudoregions

- „Pseudoregions“
  - To simplify performance data book-keeping split POMP regions into smaller conceptual pseudo-regions: enter, exit, body, main,..
  - Exactly two „events“ for each pseudo-region: ENTER and EXIT
  - Times and counts are kept for each Pseudo-region

- Opari Instrumentation with pseudo-region nesting

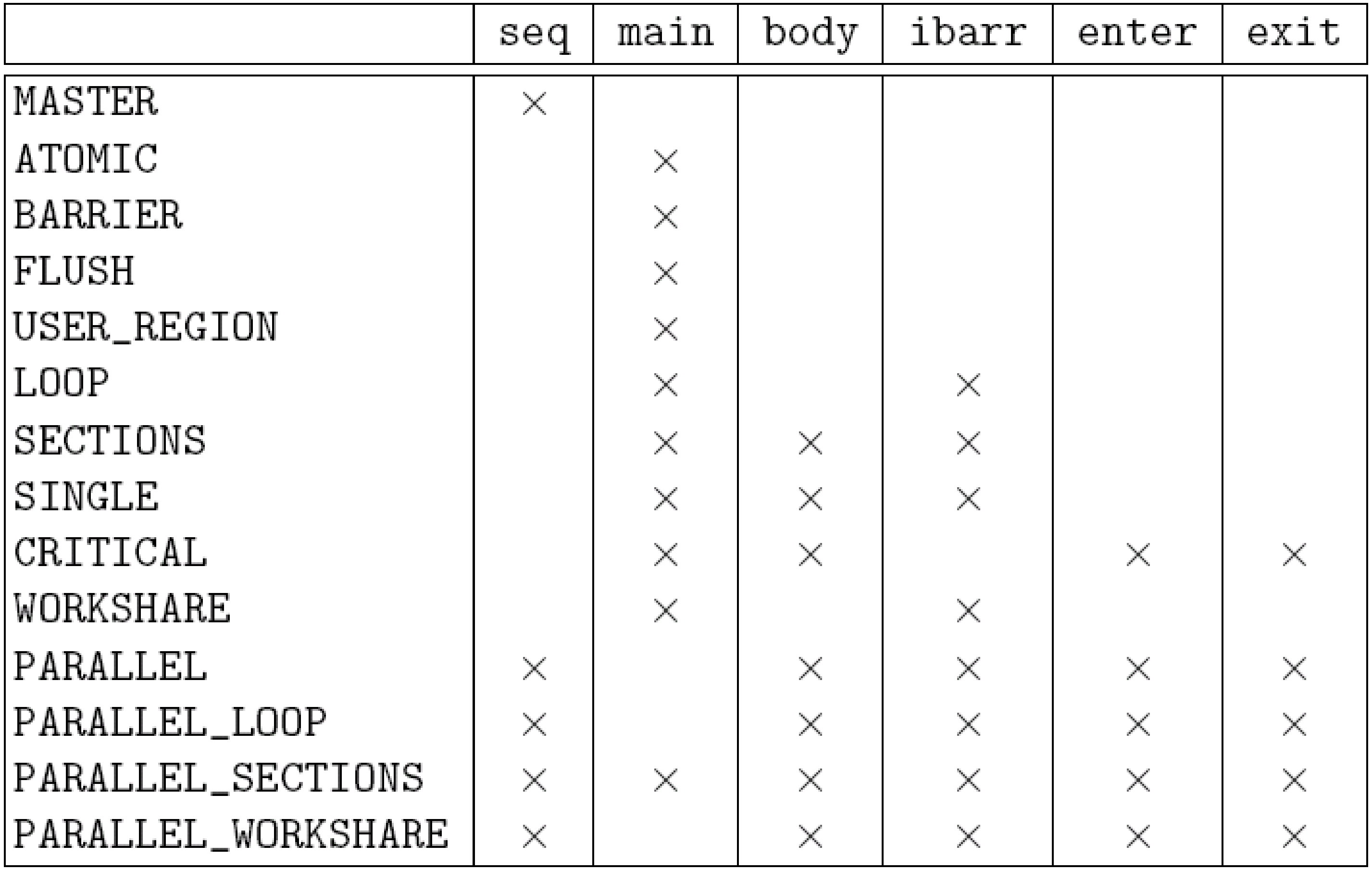


```
POMP_Parallel_fork [master]            enter                main
#pragma omp parallel {
    POMP_Parallel_begin [team]                     body

        POMP_Barrier_Enter [team]      ibarr
        #pragma omp barrier
        POMP_Barrier_Exit [team]
    POMP_Parallel_end [team]           exit
}
POMP_Parallel_join [master]
```

Pseudoregions (2)

- OpenMP constructs / POMP regions and Pseudoregions



	seq	main	body	ibarr	enter	exit
MASTER	×					
ATOMIC		×				
BARRIER		×				
FLUSH		×				
USER_REGION		×				
LOOP		×		×		
SECTIONS		×	×	×		
SINGLE		×	×	×		
CRITICAL		×	×		×	×
WORKSHARE		×		×		
PARALLEL	×		×	×	×	×
PARALLEL_LOOP	×		×	×	×	×
PARALLEL_SECTIONS	×	×	×	×	×	×
PARALLEL_WORKSHARE	×		×	×	×	×

Performance Data Reporting

- Regionstack
  - Stack of entered POMP regions is maintained
  - Performance data is attributed to stack, not to entered region itself (similar to callgraph profile vs. flat profile)

- Profiling report contains:
  - Header with general information: date and time of the program run, number of threads,...
  - List of all identified POMP regions with their type (PARALLEL, ATOMIC, BARRIER,...)
  - Region summary list: Performance data is summed over threads, list is sorted according to the summed execution time
  - Detailled region profile

Columns of the detailled region profile

- **execT**, **execC**: number of executions and total inclusive time, derived from main or body
- **exitBarT**, **exitBarC** derived from ibarr pseudo region and correspond to time spent in the implicit “exit barrier” in worksharing constructs or parallel regions.load for detecting load imbalances
- **startupT** and **startupC** derived from enter pseudo region, defined for parallel regions
- **shutdownT** and **shutdownC** defined for parallel regions, derived from exit
- **singleBodyT** and **singleBodyC** for single regions, time spent inside the single region
- **sectionT** and **sectionC**, defined for sections construct, time spent inside a section construct
- **enterT**, **enterC**, **exitT** and **exitC** for critical constructs,

Usage Examples

- Platform:
  - 4-way Itanium-2 SMP system
  - 1.3 GHz, 3 MB third level cache and 8 GB main memory
  - Intel compiler version 8.0
  - Suse Linux 2.4.21 kernel

- Test Applications:
  - APART Test Suite
  - Quicksort code from the OpenMP source code repository

APART Test Suite

- ATS:
  - Framework for testing automated and manual performance analysis tools
  - Work functions that specify a certain amount of (sequential) work for a single thread / process
  - Distribution functions specify distribution of work among threads / processes
  - Individual programs demonstrate certain inefficiencies (imbalances, etc.)
  - ompP output of „imbalance in parallel loop“ property:

```
R00003   LOOP            pattern.omp.imbalance_in_parallel_loop.c (15--18)
   001:  [R0001]  imbalance_in_parallel_loop.c (17--34)
   002:  [R0002]  pattern.omp.imbalance_in_parallel_loop.c (11--20)
   003:  [R0003]  pattern.omp.imbalance_in_parallel_loop.c (15--18)

 TID     execT     execC  exitBarT  exitBarC
  0       6.32         1      2.03         1
  1       6.32         1      2.02         1
  2       6.32         1      0.00         1
  3       6.32         1      0.00         1
  *      25.29         4      4.05         4
```

Quicksort (1)

- Parallel implementations of the quicksort algorithm are compared in [Suess04]

- Code available in the OpenMP Sourcecode repositroy (OmpSCR: http://www.pcg.ull.es/ompscr/)

- We compare two versions:
  1. Global stack of work elements. Access is protected by two critical sections
  2. Local stack of work elements (global stack is only accessed when local stack is empty)

Quicksort (2)

- Version 1.0: global stack
  - Total execution time: 61.02 seconds
  - ∑enterT + exitT = 7.01 / 4.56

```
R00002   CRITICAL        cpp_qsomp1.cpp (156--177)
   001:  [R0001]  cpp_qsomp1.cpp (307--321)
   002:  [R0002]  cpp_qsomp1.cpp (156--177)
 TID     execT     execC     enterT     enterC      exitT      exitC
  0       1.61    251780       0.87     251780       0.31     251780
  1       2.79    404056       1.54     404056       0.54     404056
  2       2.57    388107       1.38     388107       0.51     388107
  3       2.56    362630       1.39     362630       0.49     362630
  *       9.53   1406573       5.17    1406573       1.84    1406573

R00003   CRITICAL        cpp_qsomp1.cpp (211--215)
   001:  [R0001]  cpp_qsomp1.cpp (307--321)
   002:  [R0003]  cpp_qsomp1.cpp (211--215)
 TID     execT     execC     enterT     enterC      exitT      exitC
  0       1.60    251863       0.85     251863       0.32     251863
  1       1.57    247820       0.83     247820       0.31     247820
  2       1.55    229011       0.81     229011       0.31     229011
  3       1.56    242587       0.81     242587       0.31     242587
  *       6.27    971281       3.31     971281       1.25     971281
```

Quicksort (3)

- Version 2.0: local stacks
  - Total execution time: 53.44
  - ∑enterT + exitT = 5.55 / 3.32 => 25% improvement

```
R00002   CRITICAL        cpp_qsomp2.cpp (175--196)
   001:  [R0001]  cpp_qsomp2.cpp (342--358)
   002:  [R0002]  cpp_qsomp2.cpp (175--196)
 TID     execT     execC    enterT    enterC     exitT     exitC
  0       0.67    122296      0.34    122296      0.16    122296
  1       2.47    360702      1.36    360702      0.54    360702
  2       2.41    369585      1.31    369585      0.53    369585
  3       1.68    246299      0.93    246299      0.37    246299
  *       7.23   1098882      3.94   1098882      1.61   1098882

R00003   CRITICAL        cpp_qsomp2.cpp (233--243)
   001:  [R0001]  cpp_qsomp2.cpp (342--358)
   002:  [R0003]  cpp_qsomp2.cpp (233--243)
 TID     execT     execC    enterT    enterC     exitT     exitC
  0       1.22    255371      0.55    255371      0.31    255371
  1       1.16    242924      0.53    242924      0.30    242924
  2       1.32    278241      0.59    278241      0.34    278241
  3       0.98    194745      0.45    194745      0.24    194745
  *       4.67    971281      2.13    971281      1.19    971281
```

Summary

- **ompP**: simple profiling tool for OpenMP, based on POMP instrumentation
  - Simple, but can be very effective as a first step in performance tuning
  - Platform independent, can be used to compare performance on different platforms
  - Dependent on POMP instrumentation approach
  - We would *really* like to have a *standard* profiling interface

Thank You!

- Availablility:
  - First version was written in C++, → problems when linking with the ompP library (C++ run-time needs to be included as well...)
  - ompP v2.0: C-only version, same functionality
  - will be available soon from

```
http://wwwbode.informatik.tu-muenchen.de/~fuerling/ompp
```

References

- **Suess04**: Michael Süß and Claudia Leopold. A user’s experience with parallel sorting and OpenMP. In Proceedings of the Sixth Workshop on OpenMP (EWOMP’04), October 2004.

- **Cownie03**: James Cownie, John DelSignore Jr., Bronis R. de Supinski, and Karen Warren. DMPL: An OpenMP DLL debugging interface. In Proceedings of the Workshop on OpenMP Applications and Tools (WOMPAT 2003), pages 137-146, 2003.

- **Mohr02**: Bernd Mohr, Allen D. Malony, Hans-Christian Hoppe, Frank Schlimbach, Grant Haab, Jay Hoeinger, and Sanjiv Shah. A performance monitoring interface for OpenMP. In Proceedings of the Fourth Workshop on OpenMP (EWOMP 2002), September 2002.