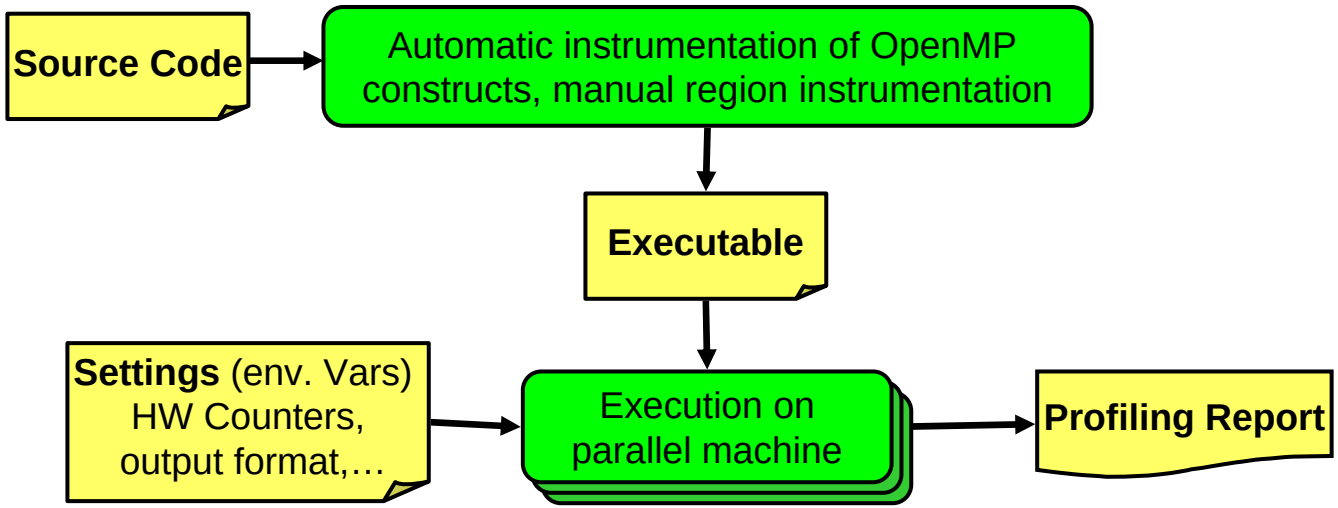
OpenMP profiling with ompP

Karl Fürlinger
karl@cs.utk.edu
March 5th 2008

THE UNIVERSITY of TENNESSEE

- ompP: Profiling tool for OpenMP
  - Based on source code instrumentation and direct measurement
  - (Mostly) independent of compiler and runtime
  - Tested and supported: Linux, Solaris, AIX and Intel, Pathscale, PGI, IBM, gcc, SUN studio compilers
  - Supports HW counters through PAPI



ompP's Profiling Report

- Header
  - Date, time, duration of the run, number of threads, used hardware counters,…
- Region Overview
  - Number of OpenMP regions (constructs) and their source-code locations
- Flat Region Profile
  - Inclusive times, counts, hardware counter data
- Callgraph and Callgraph Profiles
  - With Inclusive and exclusive times
- Overhead Analysis Report
  - Four overhead categories, per-parallel region breakdown, absolute times and percentages
- Performance Property Detection Report
  - Points out common inefficiency situations

Profiling Data

- Example profiling data

Code:

```
#pragma omp parallel
{
 #pragma omp critical
 {
   sleep(1)
 }
}
```

Profile:

```
R00002 main.c (34-37) (default) CRITICAL
 TID     execT    execC     bodyT    enterT     exitT  PAPI_TOT_INS
   0      3.00        1      1.00      2.00      0.00          1595
   1      1.00        1      1.00      0.00      0.00          6347
   2      2.00        1      1.00      1.00      0.00          1595
   3      4.00        1      1.00      3.00      0.00          1595
 SUM     10.01        4      4.00      6.00      0.00         11132
```

- Components:
  - Region number
  - Source code location and region type
  - Timing data and execution counts, **depending on the particular construct**
  - One line per thread, last line sums over all threads
  - Hardware counter data (if PAPI is available and HW counters are selected)
  - Data is exact (measured, not based on sampling)

Flat Region Profile (2)

- Times and counts reported by ompP for various OpenMP constructs

	main		*enter*		*body*					*barr*	*exit*	
construct	execT	execC	enterT	startupT	bodyT	sectionT	sectionC	singleT	singleC	exitBarT	exitT	shutdwnT
MASTER	•	•										
ATOMIC	•	•										
BARRIER	•	•										
FLUSH	•	•										
USER REGION	•	•										
CRITICAL	•	•	•		•						•	
LOCK	•	•	•		•						•	
LOOP	•	•			•					•		
WORKSHARE	•	•			•					•		
SECTIONS	•	•				•	•			•		
SINGLE	•	•						•	•	•		
PARALLEL	•	•		•	•					•		•
PARALLEL LOOP	•	•		•	•					•		•
PARALLEL SECTIONS	•	•		•		•	•			•		•
PARALLEL WORKSHARE	•	•		•	•					•		•

____T: time

____C: count

Main =
enter +
body +
barr +
exit

Callgraph

- Callgraph view
  - ‘Callgraph’ or ‘region stack’ of OpenMP constructs
  - Functions can be included by using Opari’s mechanism to instrument user defined regions: #pragma pomp inst begin(…), #pragma pomp inst end(…)
- Callgraph profile
  - Similar to flat profile, but with inclusive/exclusive times
- Example:

```
main()
{
#pragma omp parallel
  {
    foo1();
    foo2();
  }
}
```

```
void foo1()
{
#pragma pomp inst begin(foo1)
   bar();
#pragma pomp inst end(foo1)
}
```

```
void foo2()
{
#pragma pomp inst begin(foo2)
   bar();
#pragma pomp inst end(foo2)
}
```

```
void bar()
{
#pragma omp critical
  {
     sleep(1.0);
  }
}
```

Callgraph (2)

- Callgraph display

```
  Incl. CPU time
  32.22 (100.0%)            [APP 4 threads]
  32.06 (99.50%)  PARALLEL   +-R00004 main.c (42-46)
  10.02 (31.10%)   USERREG      |-R00001 main.c (19-21) ('foo1')
  10.02 (31.10%)  CRITICAL      |  +-R00003 main.c (33-36) (unnamed)
  16.03 (49.74%)   USERREG     +-R00002 main.c (26-28) ('foo2')
  16.03 (49.74%)  CRITICAL        +-R00003 main.c (33-36) (unnamed)
```

- Callgraph profiles

```
[*00] critical.ia64.ompp
[+01] R00004 main.c (42-46) PARALLEL
[+02] R00001 main.c (19-21) ('foo1') USER REGION
 TID    execT/I    execT/E      execC
   0       1.00       0.00          1
   1       3.00       0.00          1
   2       2.00       0.00          1
   3       4.00       0.00          1
 SUM      10.01       0.00          4

[*00] critical.ia64.ompp
[+01] R00004 main.c (42-46) PARALLEL
[+02] R00001 main.c (19-21) ('foo1') USER REGION
[=03] R00003 main.c (33-36) (unnamed) CRITICAL
 TID      execT      execC    bodyT/I    bodyT/E     enterT      exitT
   0       1.00          1       1.00       1.00       0.00       0.00
   1       3.00          1       1.00       1.00       2.00       0.00
   2       2.00          1       1.00       1.00       1.00       0.00
   3       4.00          1       1.00       1.00       3.00       0.00
 SUM      10.01          4       4.00       4.00       6.00       0.00
```

Performance Property Detection Report

- Detection of common inefficiency situations:
  - Example: WaitAtBarrier
  - Severity is fraction of total runtime lost due to the inefficiency
  - Implemented performance properties:
    - WaitAtBarrier
    - ImbalanceInParallelRegion
    - ImbalanceInParallelLoop, -Workshare ,-Sections
    - ImbalanceDueToNotEnoughSections
    - ImbalanceDueToUnevenSectionDistribution
    - CriticalSectionContention, LockContention
    - FrequentAtomic
    - InsufficienWorkInParallelLoop
    - UnparallelizedInMasterRegion, -SingleRegion

```
------------------------------------------------------------------------
----     ompP Performance Properties Report    ----------------------
------------------------------------------------------------------------

PROPERTY 'ImbalanceInParallelRegion' holds for
     'PARALLEL zaxpy.F (48-81)', with a severity of 0.041476

...
```

Overhead Analysis (1)

- Certain timing categories reported by ompP can be classified as overheads:
  - Example: **`exitBarT`**: Time wasted by threads idling at the exit barrier of work-sharing constructs. Reason is most likely an **imbalanced** amount of work

- Four overhead categories are defined in ompP:

  - **Imbalance**: waiting time incurred due to an imbalanced amount of work in a worksharing or parallel region

  - **Synchronization**: overhead that arises due to threads having to synchronize their activity, e.g. **`barrier`** call

  - **Limited Parallelism**: idle threads due to not enough parallelism being exposed by the program

  - **Thread management**: overhead for the creation and destruction of threads, and for signaling critical sections or locks as available

Overhead Analysis (2)

	main		*enter*		*body*					*barr*	*exit*	
construct	execT	execC	enterT	startupT	bodyT	sectionT	sectionC	singleT	singleC	exitBarT	exitT	shutdwnT
MASTER	•	•										
ATOMIC	•(S)	•										
BARRIER	•(S)	•										
FLUSH	•(S)	•										
USER REGION	•	•										
CRITICAL	•	•	•(S)		•						•(M)	
LOCK	•	•	•(S)		•						•(M)	
LOOP	•	•			•					•(I)		
WORKSHARE	•	•			•					•(I)		
SECTIONS	•	•				•	•			•(I/L)		
SINGLE	•	•						•	•	•(L)		
PARALLEL	•	•		•(M)	•					•(I)		•(M)
PARALLEL LOOP	•	•		•(M)	•					•(I)		•(M)
PARALLEL SECTIONS	•	•		•(M)		•	•			•(I/L)		•(M)
PARALLEL WORKSHARE	•	•		•(M)	•					•(I)		•(M)

S: Synchronization overhead

I: Imbalance overhead

M: Thread management overhead

L: Limited Parallelism overhead

ompP's Overhead Analysis Report

```
---------------------------------------------------------------------
----     ompP Overhead Analysis Report    ---------------------------
---------------------------------------------------------------------
Total runtime (wallclock)   : 172.64 sec [32 threads]
Number of parallel regions  : 12
Parallel coverage           : 134.83 sec (78.10%)


Parallel regions sorted by wallclock time:
         Type                              Location      Wallclock (%)
R00011  PARALL                    mgrid.F (360-384)      55.75 (32.29)
R00019  PARALL                    mgrid.F (403-427)      23.02 (13.34)
R00009  PARALL                    mgrid.F (204-217)      11.94 ( 6.92)
...
                                                 SUM     134.83 (78.10)

Overheads wrt. each individual parallel region:
         Total        Ovhds (%)  =   Synch  (%)  +  Imbal   (%)  +   Limpar (%)   +    Mgmt (%)
R00011  1783.95   337.26 (18.91)    0.00 ( 0.00)  305.75 (17.14)    0.00 ( 0.00)   31.51 ( 1.77)
R00019   736.80   129.95 (17.64)    0.00 ( 0.00)  104.28 (14.15)    0.00 ( 0.00)   25.66 ( 3.48)
R00009   382.15   183.14 (47.92)    0.00 ( 0.00)   96.47 (25.24)    0.00 ( 0.00)   86.67 (22.68)
R00015   276.11    68.85 (24.94)    0.00 ( 0.00)   51.15 (18.52)    0.00 ( 0.00)   17.70 ( 6.41)
...


Overheads wrt. whole program:
         Total        Ovhds (%)  =   Synch  (%)  +  Imbal   (%)  +   Limpar (%)   +    Mgmt (%)
R00011  1783.95   337.26 ( 6.10)    0.00 ( 0.00)  305.75 ( 5.53)    0.00 ( 0.00)   31.51 ( 0.57)
R00009   382.15   183.14 ( 3.32)    0.00 ( 0.00)   96.47 ( 1.75)    0.00 ( 0.00)   86.67 ( 1.57)
R00005   264.16   164.90 ( 2.98)    0.00 ( 0.00)   63.92 ( 1.16)    0.00 ( 0.00)  100.98 ( 1.83)
R00007   230.63   151.91 ( 2.75)    0.00 ( 0.00)   68.58 ( 1.24)    0.00 ( 0.00)   83.33 ( 1.51)
...
   SUM  4314.62  1277.89 (23.13)    0.00 ( 0.00)  872.92 (15.80)    0.00 ( 0.00)  404.97 ( 7.33)
```

Number of threads, parallel regions, parallel coverage

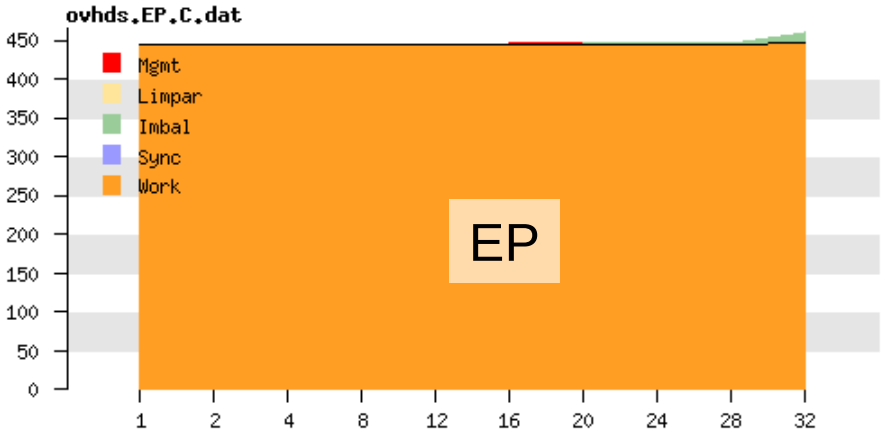
Wallclock time x number of threads

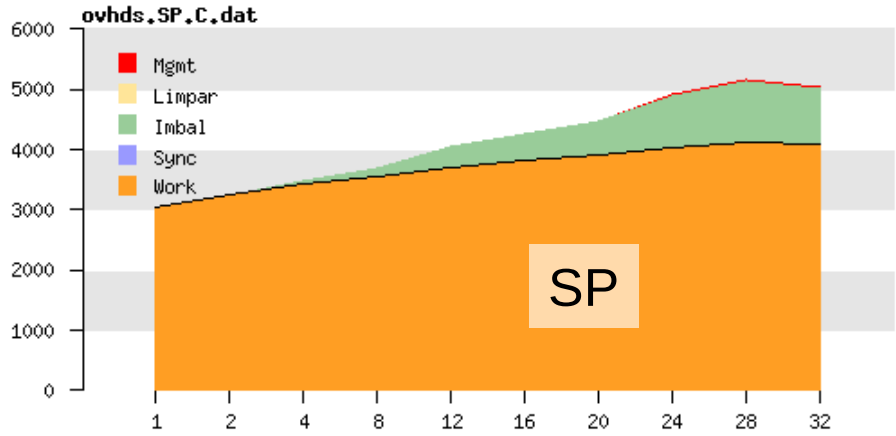
Overhead percentages wrt. this particular parallel region

Overhead percentages wrt. whole program, same absolut numbers

OpenMP Scalability Analysis

- Methodology
  - Classify execution time into “Work” and four overhead categories: “Thread Management”, “Limited Parallelism”, “Imbalance”, “Synchronization”
  - Analyze how overheads behave for increasing thread counts
  - Graphs show accumulated runtime over all threads for fixed workload (strong scaling)
  - Horizontal line = perfect scalability
- Example: NAS parallel benchmarks
  - Class C, SGI Altix machine (Itanium 2, 1.6 GHz, 6MB L3 Cache)





Usage:

- Instrumentation and linking:
  - Prepend any compile and link commands with
  - **`kinst-ompp`** or **`kinst-ompp-papi`** (Version with PAPI support)

- Example:

```
kinst-ompp icc –openmp test.c –o myapp
```

- Options are passed as environment variables

```
export OMPP_OUTFORMAT=CSV
```

Usage: Hardware Counters

- HW performance counters are supported using **PAPI**
  - Counters are selectable via **environment variables**:
    Example:

```
export OMPP_CTR1=PAPI_TOT_INS
export OMPP_CTR2=L2_MISSES
```

  - General format:

```
export OMPP_CTR<n>=<counter name> (bash, ksh)
setenv OMPP_CTR<n> <counter name> (csh, tcsh)
```

  - PAPI preset event names and native event names are supported, counter name is passed through to PAPI
  - Maximum number of counters is a compile-time constant (default: 4)
  - Maximum number of simultaneous counters depends on platform
  - Some counters cannot be counted together, ompP will report a PAPI initialization error in case of conflicts

- Evaluate expressions automatically:
  - `export OMPP_EVAL1=1-(L3_MISSES-L3_WRITES_L2_WB_MISS)/ (L3_REFERENCES-L3_WRITES_L2_WB_ALL) #L3 hit rate on Itanium`
  - Counter names are extracted from the expression automatically

Usage: Settings (1)

- Disable collection of performance data for certain **types** of OpenMP constructs:
  - Set environment variable: **`OMPP_DISABLE_<construct>=1`**
  - Example:
    `export OMPP_DISABLE_ATOMIC=1`
  - To re-enable collection of performance data for next run:
    `unset OMPP_DISABLE_ATOMIC`
    or
    `export OMPP_DISABLE_ATOMIC=0`
  - Constructs that can be disabled:
    `ATOMIC, BARRIER, CRITICAL, FLUSH, LOOP, LOCK, MASTER, SECTIONS, SINGLE, WORKSHARE, USER_REGION`

Usage: Settings (2)

- Output format, directory, and file name
  - **`OMPP_OUTFORMAT=CSV`** will write the profiling report in comma-separated values format, suitable for post-processing with Excel, etc.
  - **`OMPP_OUTDIR=<directory>`** causes ompP's profiling reports to be stored in <directory>
  - **`OMPP_APPNAME=<name>`** set the name of the target application. Useful in cases where ompP might be unable to determine the target application's name. The default name of ompP's profiling report is **`$app.$m-$n.ompp.txt`**, where $m is the number of threads used and $n is a sequence number starting from 0
  - **`OMPP_OVERWRITE=yes`** causes old report-files to be overwritten by new ones
  - **`OMPP_REPORTFILE=<filename>`** requests a specific file name for ompP's profiling report
- Error and debug messages
  - **`OMPP_QUIET=yes`** causes ompP to omit any error messages. Otherwise error messages are written to stderr and prepended by "`ompP:` "

User instrumentation

- User-defined regions

```
#pragma pomp inst begin(name)

…code…

#pragma pomp inst end(name)
```

- User-defined initialization

```
#pragma pomp inst init
```

Limitations, User Guide

- Limitations:
  - Instrumentation with Opari
  - Nested parallelization

- User Guide at

- http://www.ompp-tool.com/