Continuous Runtime Profiling of OpenMP Applications

Karl Fuerlinger, Shirley Moore

{karl,shirley}@cs.utk.edu
Outline

- Motivation
  - OpenMP and tools support

- ompP: a profiling tool for OpenMP
  - Flat profiles, callgraph profiles, data model
  - Performance properties
  - Overhead classification

- Continuous runtime profiling
  - Adding temporal dimension to profiling-type performance data
  - New performance data views:
    - Overheads, properties over time
    - Performance counter heat-maps
  - Examples from the SPEC OpenMP benchmark suite

- Conclusion and Future work
  - Integration and extension
Motivation (1)

- **OpenMP**
  - Threads-based and fork/join-oriented programming model
  - Worksharing constructs

![Diagram](image.png)

- **Characteristics**
  - Directive based (compiler pragmas, comments)
  - Incremental parallelization approach
  - Well suited for loop-based parallel programming
  - Less well suited for irregular parallelism (*task/taskpool* concepts to be included in upcoming version 3.0 of the OpenMP specification).
  - One of the contending programming paradigms for the “multicore era”
  - Traditional roots in the scientific computing community, but:
    - **Microsoft Visual C** and **gcc** now support OpenMP
Motivation (2)

- Tool support for OpenMP is limited
  - No standardized tools interface yet, cf. PMPI interface for MPI
  - Proposal for interface from SUN for version 3.0 of the OpenMP standard

- Vendor specific tools
  - SUN Studio, Intel Thread Profiler, Intel Thread Checker
  - Limited to the particular platform
  - Sampling-based approach
  - Work on system- not user level

- Research tools
  - Most tools use the POMP proposal for a performance interface for OpenMP developed by Bernd Mohr et al. (FZ Juelich)
  - Accompanying Source-to-source instrumenter called Opari
  - Successful tools for automatic and manual performance analysis have been developed based on this approach
    - TAU (Univ. of Oregon)
    - KOJAK (Univ. of Tennessee and FZ Juelich)
    - ompP: Pure profiling tool with text-based output
Outline

- **Motivation**
  - OpenMP and tools support

- **ompP: a profiling tool for OpenMP**
  - Flat profiles, callgraph profiles, data model
  - Performance properties
  - Overhead classification

- **Continuous runtime profiling**
  - Adding temporal dimension to profiling-type performance data
  - New performance data views:
    - Overheads, properties over time
    - Performance counter heat-maps
  - Examples from the SPEC OpenMP benchmark suite

- **Conclusion and Future work**
  - Integration and extension
ompP is a profiling tool for OpenMP
- Available with user guide and manual under GPL license from http://www.ompp-tool.com
- Works with all Unix-like OS and OpenMP compiler combinations
- Tested and supported: Linux, Solaris, AIX and Intel, Pathscale, PGI, IBM, gcc compilers

Diagram:
- Source Code → Instrumentation: OpenMP constructs, manual region instrumentation
- Settings (env. Vars) → Execution on parallel machine
- Execution on parallel machine → Profiling Report
ompP (2)

Characteristics:
- Target application can be written in FORTRAN or C/C++
- ompP is implemented as a static library linked to the target application
- Source-code instrumentation with Opari
- Simple usage with wrapper script, e.g., `kinst-ompp icc -o test test.c`
- Setting options as environment variables, e.g., `export OMPP_OUTFORMAT=csv`
- Reports execution times and counts for various OpenMP constructs
- Data is presented in terms of the user model of execution, not the system model
- Supports HW counters using PAPI
- ASCII based profiling report is delivered at program end
Advanced productivity features

- **Evaluators**
  - Compute expressions involving HW counters directly
  - Ex: `OMPP_EVAL1=1-L2_MISSES/L2_REFERENCES`

- **Mid-run dumping** of the profiling report is supported
  - Useful for long-running applications

- **Overhead Analysis**
  - Four well defined overhead categories of parallel execution
  - Analysis for individual parallel region and whole program

- **Scalability Analysis**
  - Analyze overheads for increasing thread counts

- **Performance Properties**
  - Detect common inefficiency situations

- **Continuous runtime profiling**
  - Profiling-over-time adds temporal dimension
ompP’s Profiling Report

- General Information
  - Date, time, duration of the run, number of threads, used hardware counters,…

- Region Overview
  - Number of regions and their source-code locations

- Flat Region Profile
  - Inclusive times, counts, hardware counter data

- Callgraph (-tree)

- Callgraph Profiles
  - Inclusive and exclusive data

- Overhead Analysis Report
  - Four overhead categories, per-parallel region breakdown, absolute times and percentages

- Performance Property Detection Report
  - Points out common inefficiency situations
Flat Region Profile (1)

- Example:

<table>
<thead>
<tr>
<th>TID</th>
<th>execT</th>
<th>execC</th>
<th>bodyT</th>
<th>enterT</th>
<th>exitT</th>
<th>PAPI_TOT_INS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.00</td>
<td>1</td>
<td>1.00</td>
<td>2.00</td>
<td>0.00</td>
<td>1595</td>
</tr>
<tr>
<td>1</td>
<td>1.00</td>
<td>1</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>6347</td>
</tr>
<tr>
<td>2</td>
<td>2.00</td>
<td>1</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td>1595</td>
</tr>
<tr>
<td>3</td>
<td>4.00</td>
<td>1</td>
<td>1.00</td>
<td>3.00</td>
<td>0.00</td>
<td>1595</td>
</tr>
<tr>
<td>SUM</td>
<td>10.01</td>
<td>4</td>
<td>4.00</td>
<td>6.00</td>
<td>0.00</td>
<td>11132</td>
</tr>
</tbody>
</table>

- 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)
Flat Region Profile (2)

- Times and counts reported by ompP for various OpenMP constructs

<table>
<thead>
<tr>
<th>construct</th>
<th>main</th>
<th>enter</th>
<th>startup</th>
<th>body</th>
<th>sectionT</th>
<th>sectionC</th>
<th>single</th>
<th>singleC</th>
<th>exitBarT</th>
<th>exit</th>
<th>shutdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>MASTER</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATOMIC</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BARRIER</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLUSH</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER REGION</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRITICAL</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOCK</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOOP</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WORKSHARE</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECTIONS</td>
<td>●</td>
<td>●</td>
<td></td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SINGLE</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PARALLEL</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PARALLEL LOOP</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PARALLEL SECTIONS</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PARALLEL WORKSHARE</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

T: time  
C: count

Main =  
enter + body + barr + exit
Overhead Analysis (1)

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

- Four overhead categories are defined in ompP:
  - \textbf{Imbalance}: waiting time incurred due to an imbalanced amount of work in a worksharing or parallel region
  - \textbf{Synchronization}: overhead that arises due to threads having to synchronize their activity, e.g. \texttt{barrier} call
  - \textbf{Limited Parallelism}: idle threads due not enough parallelism being exposed by the program
  - \textbf{Thread management}: overhead for the creation and destruction of threads, and for signaling critical sections, locks as available
# Overhead Analysis (2)

<table>
<thead>
<tr>
<th>construct</th>
<th>main</th>
<th>enter</th>
<th>startup</th>
<th>body</th>
<th>sectionT</th>
<th>sectionC</th>
<th>singleT</th>
<th>singleC</th>
<th>exitBarT</th>
<th>exitT</th>
<th>shutdownT</th>
</tr>
</thead>
<tbody>
<tr>
<td>MASTER</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATOMIC</td>
<td>●(S)</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BARRIER</td>
<td>●(S)</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLUSH</td>
<td>●(S)</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER REGION</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRITICAL</td>
<td>●</td>
<td>●</td>
<td>●(S)</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>●(M)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOCK</td>
<td>●</td>
<td>●</td>
<td>●(S)</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>●(M)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOOP</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>●(I)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WORKSHARE</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>●(I)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECTIONS</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td>●(I/L)</td>
<td></td>
<td></td>
<td></td>
<td>●(L)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SINGLE</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td>●(M)</td>
<td>●(L)</td>
<td></td>
<td></td>
<td>●(I)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PARALLEL</td>
<td>●</td>
<td>●</td>
<td>●(M)</td>
<td>●</td>
<td></td>
<td>●(I)</td>
<td>●(M)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PARALLEL LOOP</td>
<td>●</td>
<td>●</td>
<td>●(M)</td>
<td>●</td>
<td></td>
<td>●(I)</td>
<td>●(M)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PARALLEL SECTIONS</td>
<td>●</td>
<td>●</td>
<td>●(M)</td>
<td>●</td>
<td>●</td>
<td>●(I/L)</td>
<td>●(M)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PARALLEL WORKSHARE</td>
<td>●</td>
<td>●</td>
<td>●(M)</td>
<td>●</td>
<td></td>
<td>●(I)</td>
<td>●(M)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **S**: Synchronization overhead
- **I**: Imbalance overhead
- **M**: Thread management overhead
- **L**: Limited Parallelism overhead
OpenMP Scalability Analysis

Methodology

- Analyze how overheads behave for increasing thread counts
- Graphs show accumulated runtime over all threads for fixed workload (strong scaling)
- Application example: 314.mgrid_m from the SPEC OpenMP benchmark suite

- Scaling from 2 to 32 processors on an SGI Altix machine
- Markedly smaller load imbalance for thread counts of 32 and 16. Only three parallel loops show this behavior
- In all three cases, the iteration count is always a power of two (2 to 256), hence thread counts which are not a power of two exhibit larger load imbalance
Performance Properties (1)

- Other way to look at overheads
  - Example: parallel region with 4 threads, load is imbalanced

  `exitBarT[i]` time that thread `i` spends in the “exit barrier”, quantifies load imbalance

  \[
  \text{imbal} = \sum_{i=0}^{N-1} \text{exitBarT}[i]
  \]

  Condition: `(region.type)==\text{PARALLEL} && (\text{imbal}>0.0)`;

  Severity: \(\frac{\text{imbal}}{\text{total runtime} \times \text{number of threads}}\);

ASL formalism to specify properties
Performance Properties (2)

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

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

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

PROPERTY 'ImbalanceInParallelLoop' holds for
  'LOOP zaxpy.F (55-59)', with a severity of 0.035408

...
Outline

- **Motivation**
  - OpenMP and tools support

- **ompP: a profiling tool for OpenMP**
  - Flat profiles, callgraph profiles, data model
  - Performance properties
  - Overhead classification

- **Continuous runtime profiling**
  - Adding temporal dimension to profiling-type performance data
  - New performance data views:
    - Overheads, properties over time
    - Performance counter heat-maps
  - Examples from the SPEC OpenMP benchmark suite

- **Conclusion and Future work**
  - Integration and extension
Continuous Runtime Profiling

- Profiling vs. Tracing
  - Profiling:
    - Low overhead, smaller amounts of data
    - Easier to comprehend, textual interpretation possible
  - Tracing:
    - Large quantities of data, hard to comprehend manually
    - Can explain temporal phenomena, causal relationships of events are preserved

- Continuous runtime profiling: try to combine advantages of profiling and tracing
  - Add a temporal dimension to profiling-type performance data
  - See what happens during the execution without capturing full traces
  - Manual interpretation becomes harder since a new dimension is added to the performance data

- Implementation:
  - Capture and dump profiling reports not only at the end of the execution but several times while the application executes
  - Analyze how profiling reports change over time

“One-shot” Profiling

Continuous Profiling

Parco 2007, Juelich, Aachen
Triggers for capturing profiles:

- **Timer-based, fixed**: capture profiles in regular, uniform intervals: predictable storage requirements (depends only on duration of program run, size of dataset).

- **Timer-based, adaptive**: Adapt the capture rate to the behavior of the application: dump often if application behavior changes, decrease rate if application behavior stays the same.

- **Overflow-based**: Dump a profile if a hardware counter overflows. Interesting for floating point intensive application.

- **User-added**: Expose API for dumping profiles to the user aligned to outer loop iterations or phase boundaries.
Continuous Runtime Profiling

- Trigger currently implemented in ompP:
  - Capture profiles in regular intervals (selectable, 1 sec used in the experiments)
  - Timer signal is registered and delivered to profiler
  - Profiling data up to capture point stored to memory buffer
  - Dumped as individual profiling reports at the end of program execution
  - Perl scripts to analyze reports and generate graphs

- Experiments
  - SPEC OpenMP benchmark suite
    - Medium variant, 11 applications
  - 32 CPU SGI Altix machine
    - Itanium-2 processors with 1.6 GHz and 6 MB L3 cache
    - Used in batch mode
1. Region invocations over Time
   - See which OpenMP region was executed how often and when during the execution of the application
   - Either for a particular thread or summed over all threads
   - Two most time-consuming regions of application 328.fma3d:

2: Region execution time over time
   - same as invocations but use time instead of execution count
3. Overheads over time
- See how overheads develop over the application run
- How is each $\Delta t$ (1sec) spent for work or for one of the overhead classes
- Either for whole program or for a specific parallel region
- Total incurred overhead=integral under this function

Application: 328.fma3d_m

Initialization in a critical section, effectively serializing the execution for approx. 15 seconds. Overhead=31/32=96%
4. Performance Properties over time
   - Severity: negative impact on performance up to the capture point: percentage of CPU time lost due to inefficiency situation
4. Performance Properties over time (contd.)
   - 318.galgel
   - 324.equake
Continuous Profiling

- Performance counter heatmaps
  - x-axis: Time, y-axis: Thread-ID
  - Color: number of hardware counter events observed during sampling period
  - Application “applu”, medium-sized variant, counter: LOADS RETIRED
  - Visible phenomena: iterative behavior, thread grouping (pairs)
Continuous Profiling

- Performance counter heatmaps (contd.)
  - Application “apsi”, medium-sized variant, counter: FP_OPS RETIRED
  - Visible phenomena: difference in thread behavior. Maybe related to placement of threads on processors
Continuous Profiling

- Performance counter heatmaps contd.
  - Application “galgel”, medium-sized variant, counter: `DATA_EAR_CACHE_LAT1024`
  - Visible phenomena: iterative behavior, stagger-pattern
  - Middle of the timeline cut-out
Outline

- Motivation
  - OpenMP and tools support

- ompP: a profiling tool for OpenMP
  - Flat profiles, callgraph profiles, data model
  - Performance properties
  - Overhead classification

- Continuous runtime profiling
  - Adding temporal dimension to profiling-type performance data
  - New performance data views:
    - Overheads, properties over time
    - Performance counter heat-maps
  - Examples from the SPEC OpenMP benchmark suite

- Conclusion and future work
  - Integration and extension
Conclusion

- Continuous runtime profiling
  - Add temporal dimension to profiling type performance data
  - Good balance between simplicity of profiling and insight of tracing

- Phenomena that can be identified
  - Temporal location of contention for resources
  - When constructs get executed
  - Grouping of threads
  - Iterative behavior
Future Work

- Integration with MPI profiler mpiP
  - Profiling for mixed-parallel codes
  - Different models of combined usage of OpenMP and MPI
  - MPI-time as communication overhead in ompP’s overhead analysis

- Support for nested OpenMP parallelism
  - Increasing interest in this model due to hierarchical organization of processing elements

- Further investigation of continuous profiling
  - Other triggers (API, hardware-counter based)
  - Analyze and explain visible patterns, starting from application kernels.

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

Thank you for your attention!
Backup Slides
### 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:**

<table>
<thead>
<tr>
<th>Type</th>
<th>Location</th>
<th>Wallclock (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R00011</td>
<td>PARALL</td>
<td></td>
</tr>
<tr>
<td>mgrid.F (360-384)</td>
<td>55.75 (32.29)</td>
<td></td>
</tr>
<tr>
<td>R00019</td>
<td>PARALL</td>
<td></td>
</tr>
<tr>
<td>mgrid.F (403-427)</td>
<td>23.02 (13.34)</td>
<td></td>
</tr>
<tr>
<td>R00009</td>
<td>PARALL</td>
<td></td>
</tr>
<tr>
<td>mgrid.F (204-217)</td>
<td>11.94 (6.92)</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>SUM</td>
<td>134.83 (78.10)</td>
</tr>
</tbody>
</table>

---

**Overheads wrt. each individual parallel region:**

<table>
<thead>
<tr>
<th>Total Ovhd (%)</th>
<th>Synch (%)</th>
<th>Imbal (%)</th>
<th>Limpar (%)</th>
<th>Mgmt (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R00011</td>
<td>1783.95</td>
<td>337.26 (18.91)</td>
<td>0.00 (0.00)</td>
<td>305.75 (17.14)</td>
</tr>
<tr>
<td>R00019</td>
<td>736.80</td>
<td>129.95 (17.64)</td>
<td>0.00 (0.00)</td>
<td>104.28 (14.15)</td>
</tr>
<tr>
<td>R00009</td>
<td>382.15</td>
<td>183.14 (47.92)</td>
<td>0.00 (0.00)</td>
<td>96.47 (25.24)</td>
</tr>
<tr>
<td>R00015</td>
<td>276.11</td>
<td>68.85 (24.94)</td>
<td>0.00 (0.00)</td>
<td>51.15 (18.52)</td>
</tr>
<tr>
<td>...</td>
<td>SUM</td>
<td>4314.62</td>
<td>1277.89 (23.13)</td>
<td>0.00 (0.00)</td>
</tr>
</tbody>
</table>

---

**Overheads wrt. whole program:**

<table>
<thead>
<tr>
<th>Total Ovhd (%)</th>
<th>Synch (%)</th>
<th>Imbal (%)</th>
<th>Limpar (%)</th>
<th>Mgmt (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R00011</td>
<td>1783.95</td>
<td>337.26 (6.10)</td>
<td>0.00 (0.00)</td>
<td>305.75 (5.53)</td>
</tr>
<tr>
<td>R00009</td>
<td>382.15</td>
<td>183.14 (3.32)</td>
<td>0.00 (0.00)</td>
<td>96.47 (1.75)</td>
</tr>
<tr>
<td>R00005</td>
<td>264.16</td>
<td>164.90 (2.98)</td>
<td>0.00 (0.00)</td>
<td>63.92 (1.16)</td>
</tr>
<tr>
<td>R00007</td>
<td>230.63</td>
<td>151.91 (2.75)</td>
<td>0.00 (0.00)</td>
<td>68.58 (1.24)</td>
</tr>
<tr>
<td>...</td>
<td>SUM</td>
<td>4314.62</td>
<td>1277.89 (23.13)</td>
<td>0.00 (0.00)</td>
</tr>
</tbody>
</table>