Evaluating OpenMP Analysis Tools with the APART Test Suite

Bernd Mohr, Forschungszentrum Jülich
Jesper Larsson Träff, NEC
Hao Zhu, TUM

Michael Gerndt
Technische Universität München
gerndt@in.tum.de
APART Overview

• IST Working Group, 01/99 - 07/04

• Automatic Performance Analysis: Real Tools

• Focus:
  • Automatic Performance Analysis
  • Network of European projects
  • Extend work to grid environments

www.fz-juelich.de/apart
The APART Test Suite (ATS)

- **Goal:** Evaluation of automatic performance tools

- **APART Test Suite**
  - Common project inside APART group
  - Main focus: automatic performance analysis tools
  - But also useful for "regular" performance tools
  - [www.fz-juelich.de/apart/ats/](http://www.fz-juelich.de/apart/ats/)

- **ATS is collection of property functions**
Current Design of ATS Framework

- **MPI UTILS**
  - par_do_mpi_work()
  - alloc_mpi_buf()
  - free_mpi_buf()
  - alloc_mpi_vbuf()
  - free_mpi_vbuf()
  - mpi_commpattern_sendrecv()
  - mpi_commpattern_shift()

- **MPI PROPERTIES**
- **OpenMP PROPERTIES**
- **OpenMP UTILS**
  - par_do_omp_work()

- **DISTRIBUTION**
  - df_same()
  - df_cyclic2()
  - df_block2()
  - df_linear()
  - df_peak()
  - df_cyclic3()
  - df_block3()

- **WORK**
  - do_work()

- **TEST PROGRAMS**

- **MPI PROPERTIES**

- **OpenMP PROPERTIES**

- **OpenMP UTILS**

- **DISTRIBUTION**

- **WORK**
Currently Implemented OMP Property Functions

- **Imbalance**
  - 
  - _in_parallel_region
  - _in_parallel_loop, _in_parallel_loop_nowait
  - _due_to_uneven_section_distribution
  - _in_ordered_loop
  - ...

- **Unparallelized code**
  - 
  - _in_master_region, _in_single_region
  - _in_ordered_loop

- **Synchronization**
  - 
  - critical_section_locking, _contention
  - serialization_due_to_critical_section
  - all_threads_lock_contention, pairwise_lock_contention
  - ...
Currently Implemented OMP Property Functions

• Parallel Overhead
  • dynamic_scheduling_overhead
  • scheduling_overhead_in_inner_loop
  • firstprivate_initialization, ...
  • reduction_handling

• Inefficient serial execution
  • false_sharing_in_parallel_region
void imbalance_in_parallel_loop
distr_func_t df, distr_t* dd, int r) {
    int i, j, sz;
    #pragma omp parallel private(sz,i,j)
    {
        sz = omp_get_num_threads();
        for (i=0; i<r; ++i) {
            #pragma omp for schedule(static,1)
            for (j=0; j<sz; ++j) {
                par_do_omp_work(df, dd, default_sf);
            }
        }
    }
}
Performance Property Test Programs

- **Single performance property testing**
  - Programs are *generated automatically* from performance property function signature
    - Based on PDT (University of Oregon)
  - Property parameters become test program arguments

- **Composite performance property testing**
  - Programs with *multiple* performance property functions
  - Complexity only limited by imagination
  - Currently: *manually* implemented
Evaluation

- Hitachi Profiling
- Expert
- Vampir
- Intel Vtune
Hitachi Profiling

- Automatic instrumentation of parallel regions with 
  
  \texttt{-pmfunc -pmpar}

- No support for worksharing constructs

- Performance data are collected per process

- An OpenMP process runs with up to 8 threads

- ASCII format output with pmpr
### imbalance_due_to_uneven_section_distribution[2](omp_pattern.c+560)

<table>
<thead>
<tr>
<th>Task</th>
<th>CPU time</th>
<th>FLOP</th>
<th>Inst</th>
<th>LD/ST</th>
<th>D-cache</th>
<th>MFLOPS</th>
<th>MIPS</th>
<th>Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP0</td>
<td>4.492&lt;</td>
<td>16&gt;</td>
<td>76903k&gt;</td>
<td>36190k&gt;</td>
<td>2272k</td>
<td>0.000&gt;</td>
<td>17.120&gt;</td>
<td>4&gt;</td>
</tr>
<tr>
<td>IP1</td>
<td>4.492</td>
<td>16&gt;</td>
<td>76903k&gt;</td>
<td>36190k&gt;</td>
<td>2273k&gt;</td>
<td>0.000</td>
<td>17.120</td>
<td>4&gt;</td>
</tr>
<tr>
<td>IP2</td>
<td>4.493&gt;</td>
<td>16&gt;</td>
<td>76903k&gt;</td>
<td>36190k&gt;</td>
<td>2272k</td>
<td>0.000</td>
<td>17.116</td>
<td>4&gt;</td>
</tr>
<tr>
<td>IP3</td>
<td>4.493</td>
<td>16&gt;</td>
<td>76903k&gt;</td>
<td>36190k&gt;</td>
<td>2272k</td>
<td>0.000</td>
<td>17.119</td>
<td>4&gt;</td>
</tr>
<tr>
<td>IP4</td>
<td>4.492</td>
<td>8&lt;</td>
<td>38452k&lt;</td>
<td>18095k&lt;</td>
<td>1136k&lt;</td>
<td>0.000&lt;</td>
<td>8.559&lt;</td>
<td>4&gt;</td>
</tr>
<tr>
<td>IP5</td>
<td>4.492</td>
<td>8&lt;</td>
<td>38452k&lt;</td>
<td>18095k&lt;</td>
<td>1137k</td>
<td>0.000</td>
<td>8.559</td>
<td>4&gt;</td>
</tr>
<tr>
<td>IP6</td>
<td>4.493</td>
<td>8&lt;</td>
<td>38452k&lt;</td>
<td>18095k&lt;</td>
<td>1137k</td>
<td>0.000</td>
<td>8.559</td>
<td>4&gt;</td>
</tr>
<tr>
<td>IP7</td>
<td>4.493</td>
<td>8&lt;</td>
<td>38452k&lt;</td>
<td>18095k&lt;</td>
<td>1137k</td>
<td>0.000</td>
<td>8.559</td>
<td>4&gt;</td>
</tr>
<tr>
<td>TOTAL</td>
<td>35.940</td>
<td>96</td>
<td>461419k</td>
<td>217138k</td>
<td>13635k</td>
<td>0.000</td>
<td>102.693</td>
<td>32</td>
</tr>
</tbody>
</table>
Hitachi Profiling Results

- Limitation due to instrumentation
  - Only parallel regions instrumented
  - No synchronization information
  - No parallel overhead

- Imbalance and unparallelized code
  - Imbalance only indirect via operation count

- Ordered loops executed as sequential loops

- But: false_sharing via cache counters
Automatic Analysis with Expert

- Expert developed at Research Centre Jülich
  - Felix Wolf, Bernd Mohr

- Searches trace files for patterns

- Extensible pattern database

- Visualization by the Expert presenter

- Available on Hitachi, PC clusters and many other machines
Expert Results

- Detects problems related to
  - Imbalance as barrier time
  - Synchronization overhead for critical section and locking
  - Parallel overhead
  - Sequential code
- No support for
  - Detailed parallel overhead
  - Ordered loops
  - Cache problems
- No details, e.g., about reason for imbalance
Vampir

- Visualization and Analysis of MPI Programs
  - Support for OpenMP is not yet included
  - Trace generation must support OpenMP
  - Our tests convert to standard events
Vtune

- Formerly known as GuideView
- Integrated into Vtune as Thread Profiler
- Profiling supported via compiler switch of Intel F90 and C++ compilers
Overhead Categories

- **Barrier**
  - Time spent waiting for other threads to arrive at a barrier.

- **Locks**
  - Time spent waiting to enter critical sections and acquire locks.

- **Parallel Overhead**
  - Estimated time spent inside of parallel regions in the OpenMP Runtime Engine, which implements OpenMP.

- **Imbalance**
  - Time spent waiting for other threads to reach the end of a parallel region.

- **Synchronized**
  - Time spent inside critical sections and locks.

- **Sequential Overhead**
  - Sequential Overhead is an estimate of the time the application spent in OpenMP regions that were not executed within an OpenMP parallel region.
Vtune Results

• Only tool identifying serialization due to critical region via synchronized overhead.

• Only tool showing unbalanced loop with nowait

• Imbalances are marked as imbalance or barrier

• Parallel overhead is not identified
  • Too small compared to barrier overhead.
Summary

• The ATS will be available at APART web site soon
• The performance tools find imbalance properties
• Expert and Vtune allow to inspect sequential code
• Vtune is the only tool
  • Distinguishing lock competition and synchronized time
  • Identifying imbalance in nowait loop
• None of the tools is able to give more detailed information, e.g., distinguishing
  • Imbalance in section and imbalanced number of section
  • Imbalance and unparallelized code
• Parallel overhead, e.g., firstprivate, is hidden behind synchronization time