

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

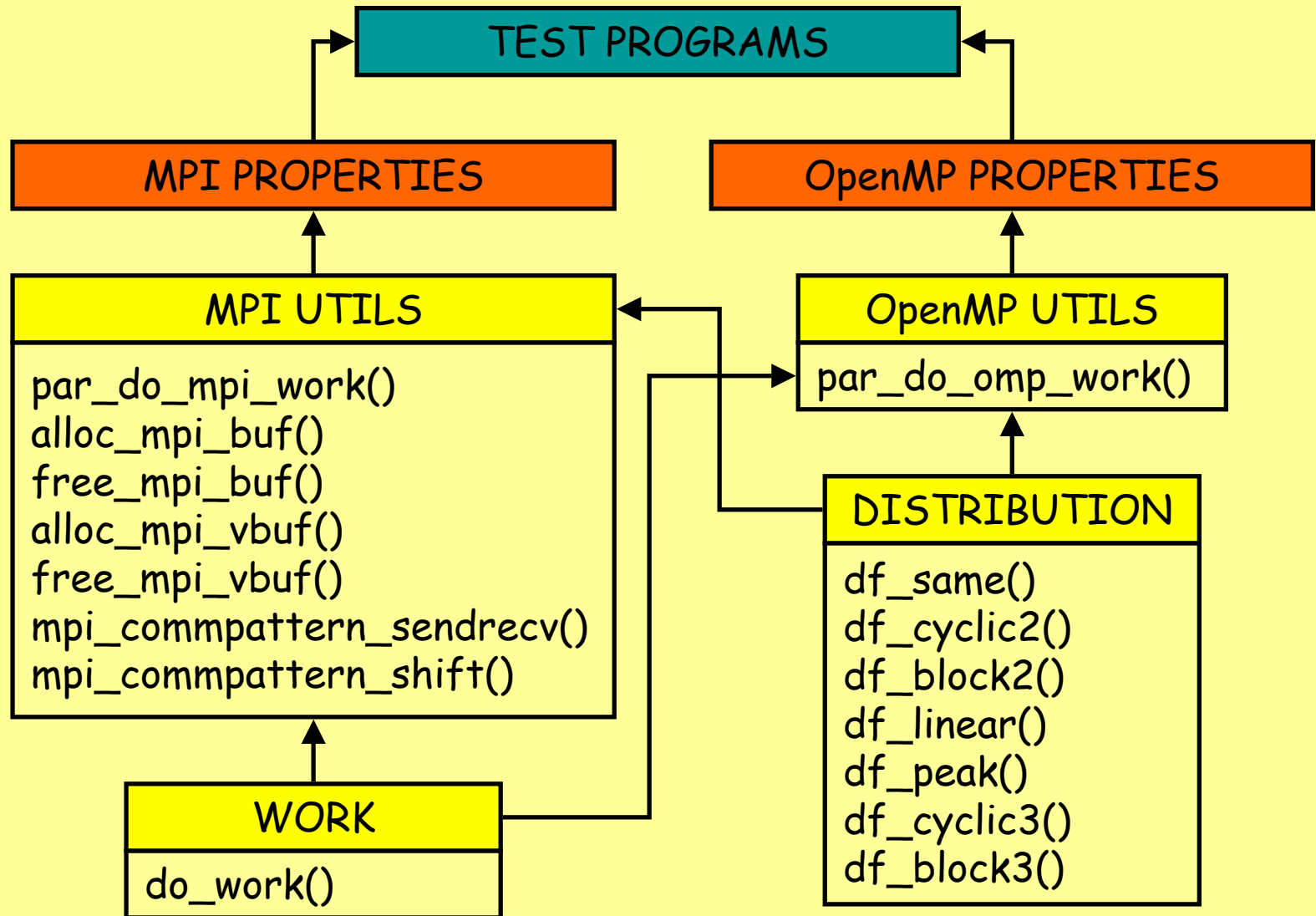
APART Overview

- 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

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/
- ATS is collection of property functions

Current Design of ATS Framework



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`

Imbalance_In_Parallel_Loop

```
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 `-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`

pmpr

imbalance_due_to_uneven_section_distribution[2] (omp_pattern.c+560)

CPU time	FLOP	Inst	LD/ST	D-cache	MFLOPS	MIPS	Times	
IP0	4.492<	16>	76903k>	36190k>	2272k	0.000>	17.120>	4>
IP1	4.492	16>	76903k>	36190k>	2273k>	0.000	17.120	4>
IP2	4.493>	16>	76903k>	36190k>	2272k	0.000	17.116	4>
IP3				36190k>	2272k	0.000	17.119	4>
IP4				18095k<	1136k<	0.000<	8.559<	4>
IP5	4.492	8<	38452k<	18095k<	1137k	0.000	8.559	4>
IP6	4.493	8<	38452k<	18095k<	1137k	0.000	8.559	4>
IP7	4.493	8<	38452k<	18095k<	1137k	0.000	8.559	4>
TOTAL	35.940	96	461419k	217138k	13635k	0.000	102.693	32

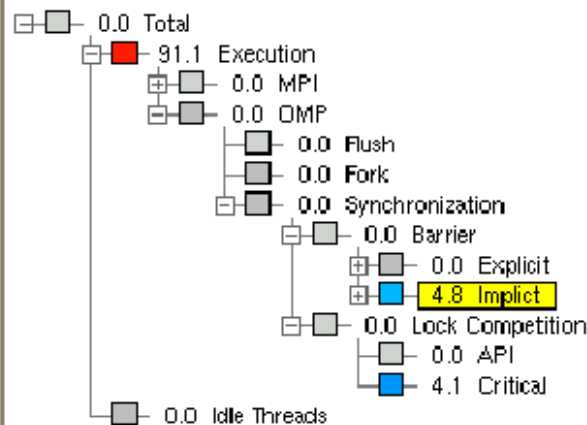
Hitachi Profiling Results

- **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**

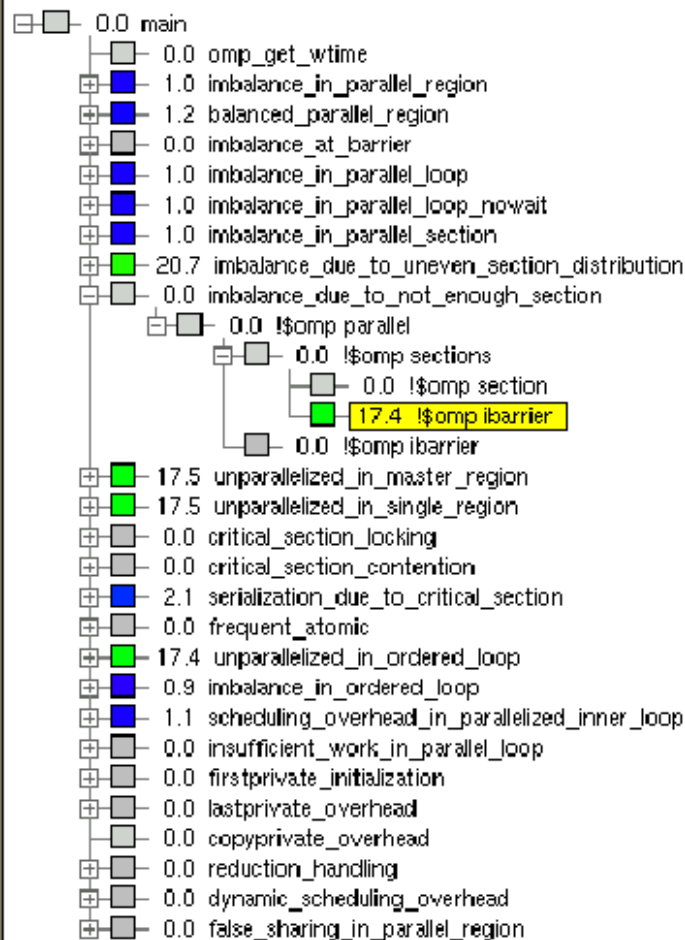
Automatic Analysis with Expert

- 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

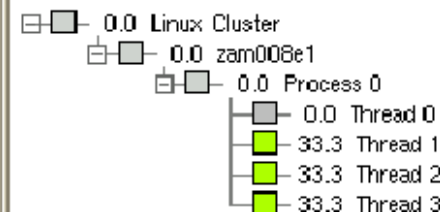
Performance Properties



Dynamic Call Tree



Locations



OMP 4

Expert Results

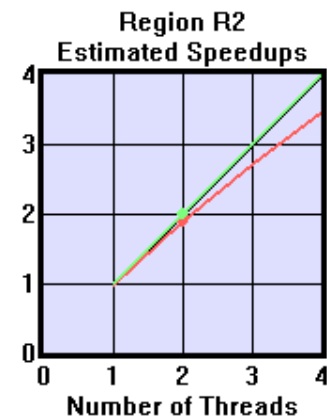
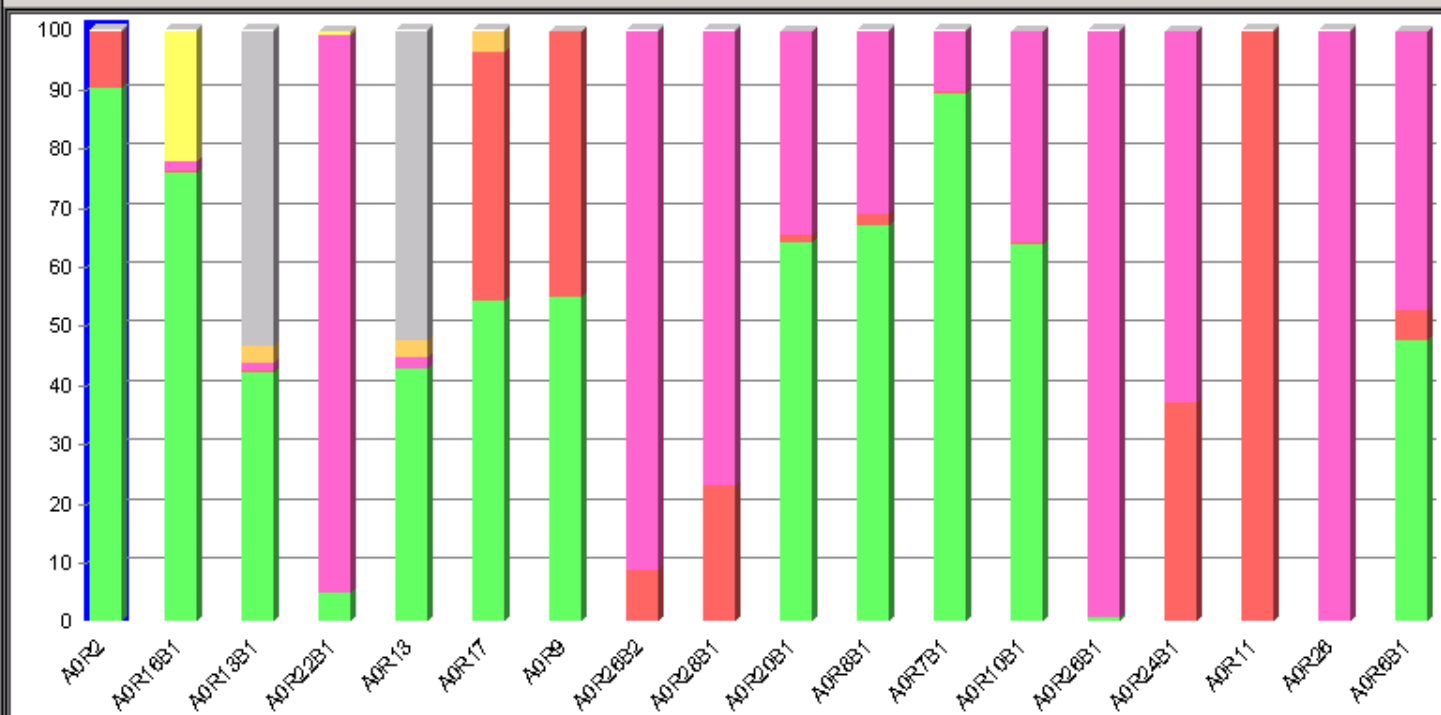
- Detects problems related to
 - Imbalance as barrier time
 - Synchronization overhead for critical section and locking
 - Parallel overhead
 - Sequential code
- No support for
 - Detailed parallel overhead
 - Ordered loops
 - Cache problems
- No details, e.g., about reason for imbalance

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.



Speedup Curves:
 — upper, — lower, — ideal
 Speedups of Runs:
 ◆ upper, ◆ lower, ◆ actual

Legend

Label	Num. threads	Total	% Parallel	% Sequential	% Imbalance	% Barrier	% Locks	% Synchronized	% Parallel overheads
AOR2	2	10,925	90,325	0,000	9,675	0,000	0,000	0,000	0,000
AOR16...	2	2,890	75,896	0,000	0,311	1,522	0,000	0,000	0,000
AOR13...	2	1,089	42,268	0,000	0,100	1,469	2,663	53,489	0,000
AOR22...	2	0,444	5,060	0,000	0,008	94,257	0,000	0,000	0,000
AOR13	2	0,345	42,742	0,000	0,000	2,029	2,754	52,464	0,000
AOR17	2	0,318	54,392	0,000	41,829	0,000	3,778	0,000	0,000
AOR9	2	0,322	54,964	0,000	45,036	0,000	0,000	0,000	0,000
AOR26...	2	0,101	0,000	0,000	8,911	91,087	0,000	0,000	0,000

1 runs 1 showing, 105 regions 105 showing

Data Threads **Regions** Summary

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