Integrated Measurement for Cross-Platform OpenMP Performance Analysis

Kevin A. Huck, Allen D. Malony,
Sameer Shende, Doug W. Jacobsen

IWOMP 2014 : Salvador, Brazil :
September 30, 2014

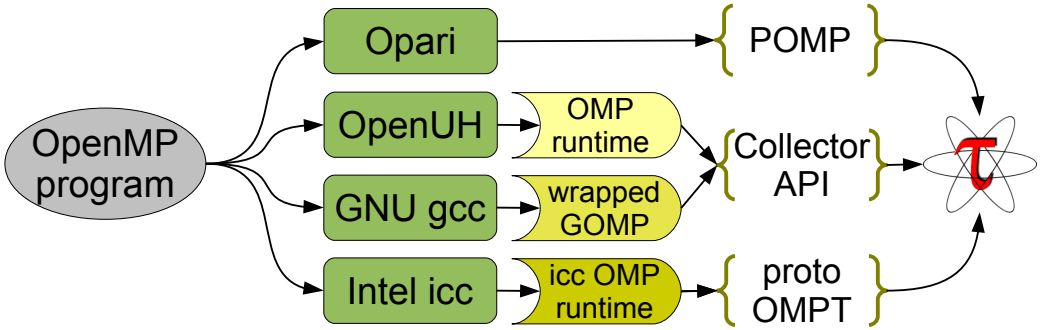
UNIVERSITY OF OREGON

OpenMP introspection: Motivation

- OpenMP: programing language extensions requiring **compiler interpretation**, **code transformation** and **runtime support**
- How can external tools observe the execution-time performance of resulting code?
- Needed:
  - Performance visibility
  - Semantic context
- Wanted: **portable**, **robust, efficient, always-on/ available** approach to observe OpenMP concurrency, including **internals**

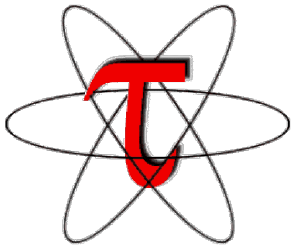
Our Approach

- Survey four methods for OpenMP measurement:
  - POMP
  - ORA w/OpenUH
  - ORA w/GOMP
  - OMPT w/Intel



- …All integrated into TAU (see next slide)
- Compare performance visibility, semantic context, measurement overhead

TAU v2.23.2b3



- Tuning and Analysis Utilities (20+ year project)
- Performance problem solving framework for HPC
  - Integrated, scalable, flexible, portable
  - Target all parallel programming / execution paradigms
- Integrated performance toolkit
  - Multi-level performance instrumentation
  - Flexible and configurable performance measurement
  - Widely-ported performance profiling / tracing system
  - Performance data management and data mining
  - Open source (BSD-style license)
- Broad use in complex software, systems, applications
- http://tau.uoregon.edu

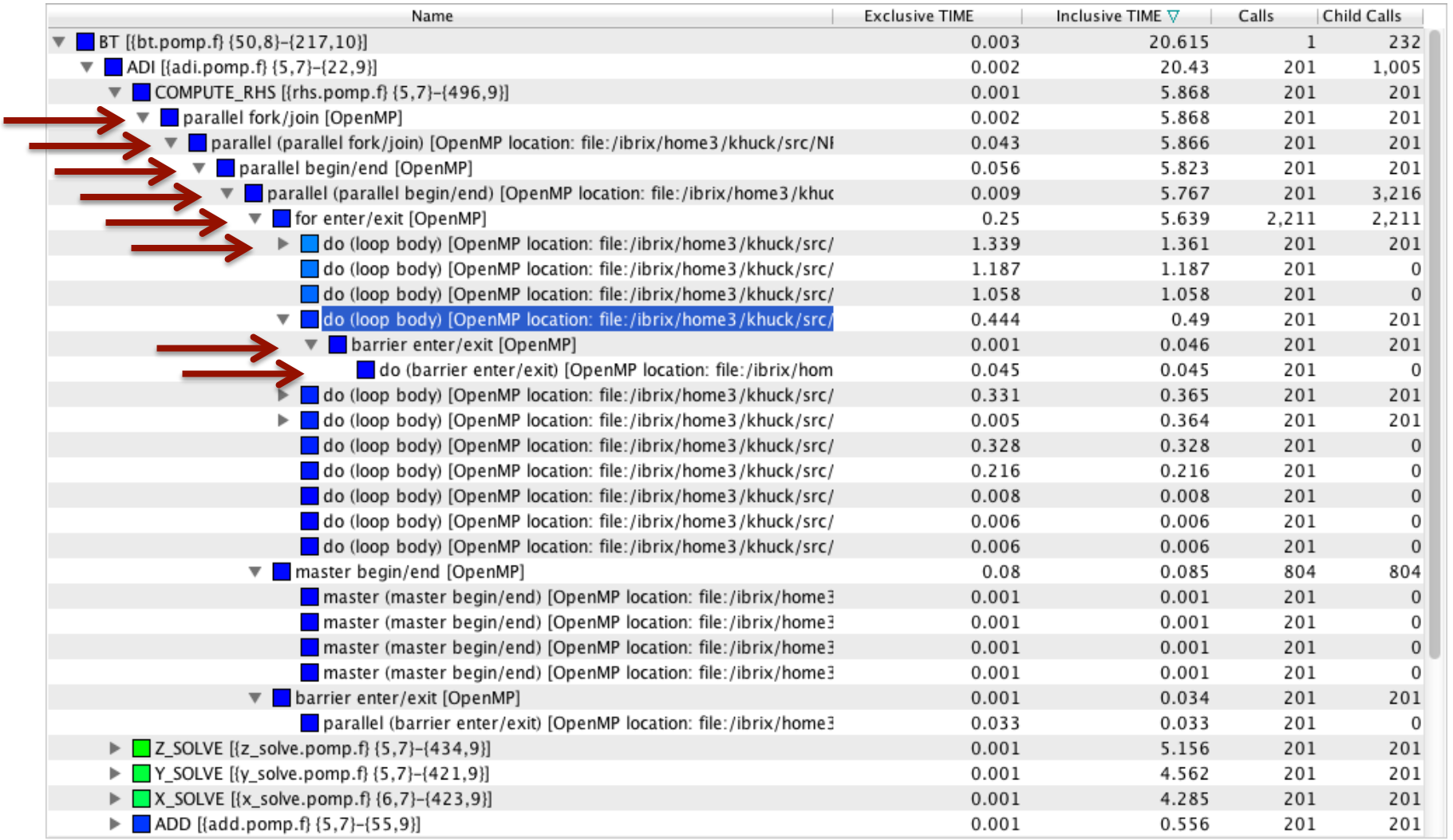
POMP & OPARI

- POMP: “Profiling OpenMP”
- OPARI: “OpenMP Pragma And Region Instrumentor”
- Explicit instrumentation of OpenMP directives: Mohr et al., SC 2001
- Pros
  - Highly portable
  - Full source context
- Cons
  - Requires instrumentation, recompile
  - All barriers are explicit, extra barrier for all regions
  - Lacks explicit sampling tool support
  - Ignorance of OpenMP behavior in uninstrumented libraries
  - Ignorance of runtime library internals
- Runtimes: **all Fortran and C/C++ compilers/runtimes**
- Tool examples: Vampir, Scalasca/KOJAK, TAU, ompP…

POMP Instrumentation Example

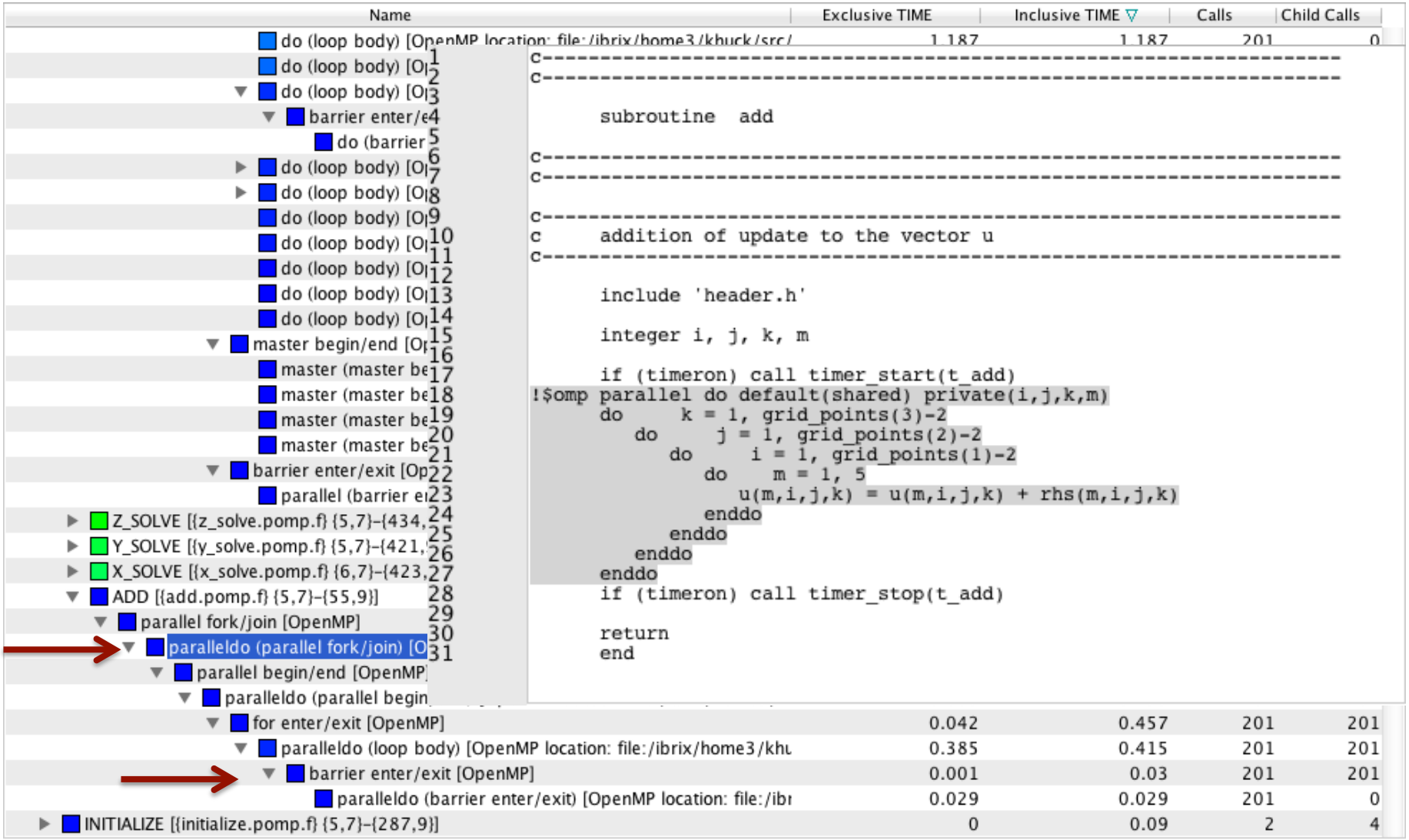
```
{       int pomp2_num_threads = omp_get_max_threads();
        int pomp2_if = 1;
        POMP2_Task_handle pomp2_old_task;
        POMP2_Parallel_fork(&pomp2_region_1, pomp2_if, pomp2_num_threads, &pomp2_old_task, pomp2_ctc_1);
#pragma omp parallel POMP2_DLIST_00 firstprivate(pomp2_old_task) if (pomp2_if) num_threads(pomp2_num_threads)
        {       POMP2_Parallel_begin(&pomp2_region_1 );
                {       POMP2_For_enter(&pomp2_region_1, pomp2_ctc_1);
                        #pragma omp for nowait
                        for (i=nStart; i<=nEnd; ++i) {
                                foo();
                        }
                        {       POMP2_Task_handle pomp2_old_task;
                                POMP2_Implicit_barrier_enter( &pomp2_region_1, &pomp2_old_task );
#pragma omp barrier
                                POMP2_Implicit_barrier_exit( &pomp2_region_1, pomp2_old_task );
                        }
                        POMP2_For_exit( &pomp2_region_1 );
                }
                POMP2_Parallel_end( &pomp2_region_1 );
        }
        POMP2_Parallel_join( &pomp2_region_1, pomp2_old_task );
}
```

Profile Example NPB3.2 BT.B (32)



Name	Exclusive TIME	Inclusive TIME ▽	Calls	Child Calls
BT [{bt.pomp.f} {50,8}-{217,10}]	0.003	20.615	1	232
ADI [{adi.pomp.f} {5,7}-{22,9}]	0.002	20.43	201	1,005
COMPUTE_RHS [{rhs.pomp.f} {5,7}-{496,9}]	0.001	5.868	201	201
parallel fork/join [OpenMP]	0.002	5.868	201	201
parallel (parallel fork/join) [OpenMP location: file:/ibrix/home3/khuck/src/NI	0.043	5.866	201	201
parallel begin/end [OpenMP]	0.056	5.823	201	201
parallel (parallel begin/end) [OpenMP location: file:/ibrix/home3/khuc	0.009	5.767	201	3,216
for enter/exit [OpenMP]	0.25	5.639	2,211	2,211
do (loop body) [OpenMP location: file:/ibrix/home3/khuck/src/	1.339	1.361	201	201
do (loop body) [OpenMP location: file:/ibrix/home3/khuck/src/	1.187	1.187	201	0
do (loop body) [OpenMP location: file:/ibrix/home3/khuck/src/	1.058	1.058	201	0
do (loop body) [OpenMP location: file:/ibrix/home3/khuck/src/	0.444	0.49	201	201
barrier enter/exit [OpenMP]	0.001	0.046	201	201
do (barrier enter/exit) [OpenMP location: file:/ibrix/hom	0.045	0.045	201	0
do (loop body) [OpenMP location: file:/ibrix/home3/khuck/src/	0.331	0.365	201	201
do (loop body) [OpenMP location: file:/ibrix/home3/khuck/src/	0.005	0.364	201	201
do (loop body) [OpenMP location: file:/ibrix/home3/khuck/src/	0.328	0.328	201	0
do (loop body) [OpenMP location: file:/ibrix/home3/khuck/src/	0.216	0.216	201	0
do (loop body) [OpenMP location: file:/ibrix/home3/khuck/src/	0.008	0.008	201	0
do (loop body) [OpenMP location: file:/ibrix/home3/khuck/src/	0.006	0.006	201	0
do (loop body) [OpenMP location: file:/ibrix/home3/khuck/src/	0.006	0.006	201	0
master begin/end [OpenMP]	0.08	0.085	804	804
master (master begin/end) [OpenMP location: file:/ibrix/home3	0.001	0.001	201	0
master (master begin/end) [OpenMP location: file:/ibrix/home3	0.001	0.001	201	0
master (master begin/end) [OpenMP location: file:/ibrix/home3	0.001	0.001	201	0
master (master begin/end) [OpenMP location: file:/ibrix/home3	0.001	0.001	201	0
barrier enter/exit [OpenMP]	0.001	0.034	201	201
parallel (barrier enter/exit) [OpenMP location: file:/ibrix/home3	0.033	0.033	201	0
Z_SOLVE [{z_solve.pomp.f} {5,7}-{434,9}]	0.001	5.156	201	201
Y_SOLVE [{y_solve.pomp.f} {5,7}-{421,9}]	0.001	4.562	201	201
X_SOLVE [{x_solve.pomp.f} {6,7}-{423,9}]	0.001	4.285	201	201
ADD [{add.pomp.f} {5,7}-{55,9}]	0.001	0.556	201	201

Profile Example NPB3.2 BT.B (32)



OpenMP Runtime API / Collector API

- Proposed callback interface for tool support in OpenMP runtimes
- Itzkowitz et al., 2006 – Sun / Oracle
- 1 API function for tools to make requests (registration, query state)

```
int __omp_collector_api(void *message);
```

- Callback interface to tools from runtime
  - 22 events (profiling/tracing), 11 states (sampling)
- Pros
  - Instrumentation not required
  - Event, sampling support
  - Partial view of runtime internals (implicit barriers, locks)
  - No more library blind spots
- Cons
  - Not widely implemented
  - No static executable support (requires `dlsym();` call)
  - No explicit support for locks, stack frame locations, blame shifting
  - No support for source context (integer "region id")
- Runtimes: **Solaris Studio, Open64, OpenUH***

*with task extensions presented @IWOMP2013

ORA Events & States

- Fork/Join
- Begin/End Idle
- Begin/End Implicit Barrier
- Begin/End Explicit Barrier
- Begin/End Lock wait
- Begin/End Critical wait
- Begin/End Ordered wait
- Begin/End Master
- Begin/End Single
- Begin/End Ordered
- Begin/End Atomic Wait

- Overhead
- Working
- Implicit Barrier
- Explicit Barrier
- Idle
- Serial
- Reduction
- Lock Wait
- Critical Wait
- Ordered Wait
- Atomic Wait

OpenUH new Task Events

- Begin/End Create Task Immediate/Delayed
- Begin/End Schedule Task
- Begin/End Suspend Task
- Begin/End Steal Task
- Fetched Task
- Begin Execute Task
- Begin/End Finish Task

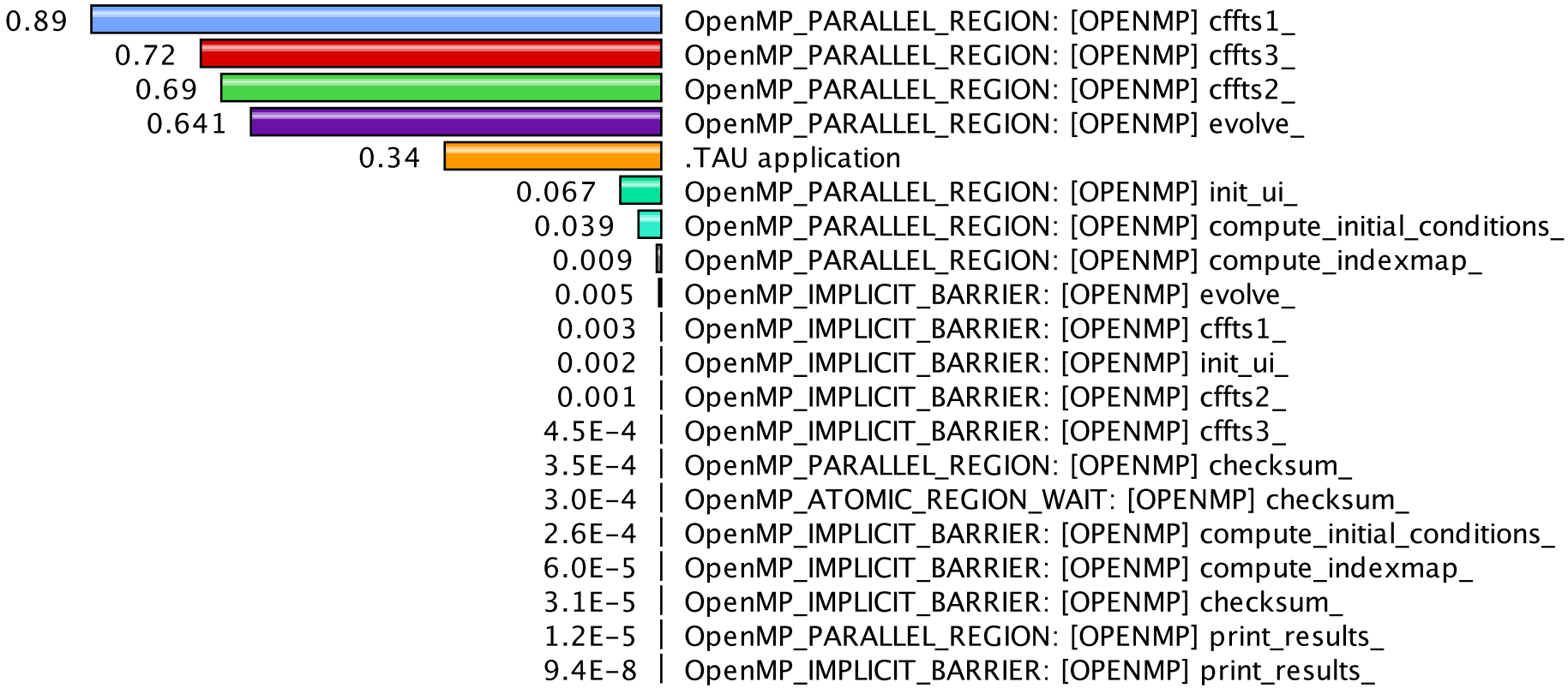
Profile Example NPB3.2 BT.B (32)

Name	Exclusive TIME	Inclusive TIME ▽	Calls	Child Calls
BT [{bt.f} {49,8}-{215,10}]	0.003	20.306	1	232
ADI [{adi.f} {4,7}-{20,9}]	0.003	20.148	201	1,005
COMPUTE_RHS [{rhs.f} {4,7}-{426,9}]	0.006	5.522	201	201
OpenMP_PARALLEL_REGION: COMPUTE_RHS [{rhs.f} {4,7}-{426,9}]	4.987	5.516	201	1,809
OpenMP_IMPLICIT_BARRIER: COMPUTE_RHS [{rhs.f} {4,7}-{426,9}]	0.517	0.517	1,005	0
OpenMP_MASTER_REGION: COMPUTE_RHS [{rhs.f} {4,7}-{426,9}]	0.012	0.012	804	0
Z_SOLVE [{z_solve.f} {4,7}-{410,9}]	0.006	5.382	201	201
OpenMP_PARALLEL_REGION: Z_SOLVE [{z_solve.f} {4,7}-{410,9}]	5.325	5.376	201	201
OpenMP_IMPLICIT_BARRIER: Z_SOLVE [{z_solve.f} {4,7}-{410,9}]	0.051	0.051	201	0
Y_SOLVE [{y_solve.f} {4,7}-{397,9}]	0.006	4.421	201	201
OpenMP_PARALLEL_REGION: Y_SOLVE [{y_solve.f} {4,7}-{397,9}]	4.334	4.416	201	201
OpenMP_IMPLICIT_BARRIER: Y_SOLVE [{y_solve.f} {4,7}-{397,9}]	0.082	0.082	201	0
X_SOLVE [{x_solve.f} {5,7}-{399,9}]	0.006	4.23	201	201
OpenMP_PARALLEL_REGION: X_SOLVE [{x_solve.f} {5,7}-{399,9}]	4.106	4.224	201	201
OpenMP_IMPLICIT_BARRIER: X_SOLVE [{x_solve.f} {5,7}-{399,9}]	0.118	0.118	201	0
ADD [{add.f} {4,7}-{31,9}]	0.005	0.59	201	201
OpenMP_PARALLEL_REGION: ADD [{add.f} {4,7}-{31,9}]	0.385	0.585	201	201
OpenMP_IMPLICIT_BARRIER: ADD [{add.f} {4,7}-{31,9}]	0.2	0.2	201	0
INITIALIZE [{initialize.f} {4,7}-{222,9}]	0	0.088	2	4
EXACT_RHS [{exact_rhs.f} {5,7}-{351,9}]	0	0.035	1	1
VERIFY [{verify.f} {5,9}-{326,11}]	0	0.031	1	3
PRINT_RESULTS [{print_results.f} {2,7}-{136,12}]	0	0.001	1	1
TIMER_START [{timers.f} {23,7}-{38,9}]	0	0	1	1
TIMER_STOP [{timers.f} {44,7}-{61,9}]	0	0	1	1
SET_CONSTANTS [{set_constants.f} {4,7}-{200,9}]	0	0	1	0
TIMER_CLEAR [{timers.f} {4,7}-{17,9}]	0	0	22	0
TIMER_READ [{timers.f} {67,7}-{79,9}]	0	0	1	0

GOMP ORA Support

- GOMP Implementation of OpenMP Collector API (ORA)
- GOMP_* library functions wrapped dynamically, statically
- Pros
  - Same as Collector API (mostly)
  - Dynamic, static executable support (static requires relinking)
- Cons
  - Only available for GNU
  - No introspection of GOMP internals
  - Only partial support for implicit barriers
  - GNU runtime is functional, widely available, but not optimal
  - No explicit support for blame shifting, source context
- Runtimes: **GNU**

Profile Example NPB3.2 FT.B (32)



Flat profile from uninstrumented binary, but getting events from runtime callbacks

OMPT: OpenMP Tools Interface

- Proposal to add callback API to OpenMP runtimes
- Pros:
  - No instrumentation required
  - Event, sampling support
  - Dynamic, static executable support
  - Internals knowledge, surgical measurement, always on
  - Blame-shifting, debugger support
- Cons
  - Still in development
  - Not (yet) an accepted standard
- Available: **IBM** (limited/experimental on BGQ), **GNU** (prototyped by Rice, will be reimplemented), **Intel** (in development), **OpenUH** (planned), **MPC** (CEA)

Profile Example: NPB 3.2 BP (32)

Name	Exclusive TIME	Inclusive TIME ▽	Calls	Child Calls
BT [{bt.f} {49,8}-{215,10}]	0.005	18.247	1	232
ADI [{adi.f} {4,7}-{20,9}]	0.003	18.107	201	1,005
COMPUTE_RHS [{rhs.f} {4,7}-{426,9}]	0.004	5.37	201	201
OpenMP_PARALLEL_REGION: COMPUTE_RHS [{rhs.f} {4,7}-{426,9}]	0.045	5.367	201	4,020
OpenMP_LOOP: COMPUTE_RHS [{rhs.f} {4,7}-{426,9}]	4.344	4.344	2,211	0
OpenMP_BARRIER: COMPUTE_RHS [{rhs.f} {4,7}-{426,9}]	0.021	0.957	1,005	2,640
IDLE	0.936	0.936	2,640	0
OpenMP_MASTER_REGION: COMPUTE_RHS [{rhs.f} {4,7}-{426,9}]	0.021	0.021	804	0
Z_SOLVE [{z_solve.f} {4,7}-{410,9}]	0.005	4.57	201	201
OpenMP_PARALLEL_REGION: Z_SOLVE [{z_solve.f} {4,7}-{410,9}]	0.003	4.565	201	402
OpenMP_LOOP: Z_SOLVE [{z_solve.f} {4,7}-{410,9}]	4.194	4.194	201	0
OpenMP_BARRIER: Z_SOLVE [{z_solve.f} {4,7}-{410,9}]	0.002	0.367	201	202
IDLE	0.366	0.366	202	0
Y_SOLVE [{y_solve.f} {4,7}-{397,9}]	0.004	3.981	201	201
OpenMP_PARALLEL_REGION: Y_SOLVE [{y_solve.f} {4,7}-{397,9}]	0.003	3.977	201	402
OpenMP_LOOP: Y_SOLVE [{y_solve.f} {4,7}-{397,9}]	3.856	3.856	201	0
OpenMP_BARRIER: Y_SOLVE [{y_solve.f} {4,7}-{397,9}]	0.002	0.118	201	203
IDLE	0.116	0.116	203	0
X_SOLVE [{x_solve.f} {5,7}-{399,9}]	0.004	3.709	201	201
OpenMP_PARALLEL_REGION: X_SOLVE [{x_solve.f} {5,7}-{399,9}]	0.003	3.704	201	402
OpenMP_LOOP: X_SOLVE [{x_solve.f} {5,7}-{399,9}]	3.683	3.683	201	0
OpenMP_BARRIER: X_SOLVE [{x_solve.f} {5,7}-{399,9}]	0.002	0.018	201	161
IDLE	0.016	0.016	161	0
ADD [{add.f} {4,7}-{31,9}]	0.004	0.475	201	201
OpenMP_PARALLEL_REGION: ADD [{add.f} {4,7}-{31,9}]	0.003	0.47	201	402
OpenMP_LOOP: ADD [{add.f} {4,7}-{31,9}]	0.345	0.345	201	0
OpenMP_BARRIER: ADD [{add.f} {4,7}-{31,9}]	0.001	0.123	201	228
IDLE	0.121	0.121	228	0
INITIALIZE [{initialize.f} {4,7}-{222,9}]	0.014	0.07	2	4
EXACT_RHS [{exact_rhs.f} {5,7}-{351,9}]	0	0.032	1	1

Event/State Coverage

Feature	OPARI	ORA	OpenUH-ORA	GOMP-ORA	OMPT
Parallel Region	Enter/exit, begin/end	✔	✔	✔	✔
Thread create/exit	Pthread	✖	✖	✔	✔
Work Sharing	Enter/exit, begin/end	✔	✔	✔	✔
Atomics	Enter/exit	Wait State	Wait State	Wait State	✔
Barriers	Explicit only	✔	✔	Explicit, Some Implicit	✔
Critical	Enter/exit, begin/end	Wait State	Wait State	Wait State	✔
Master	Begin/end	✔	✔	✔	✔
Ordered	Enter/exit, begin/end	✔, Wait	✔, Wait State	✔, Wait State	✔
Sections	Enter/exit, begin/end	✖	✖	✖	✔
Single	Begin/end	✔	✔	✔	✔
Locks	Lock wrapper	Wait State	Wait State	Wait State	✔
Task Creation	Begin/end	✔	✔	✔	✔
Task Schedule	✖	✖	✔	n/a	✔
Task Wait	Begin/end	✔	✔	✔	✔
Task Execution	Begin/end	✔	✔	✔	✔
Task Completion	✖	✖	✔	✖	✔
Task Yield	✖	✖	✔	n/a	✔

Benchmark Experiments

- **Compilers**: GNU 4.8.0, OpenUH 3.0.26, Intel 13.1.2, with –O3 or -Ofast
- **System**: 1 “fatnode” on ACISS@UO, with four Intel X7560 2.27GHz 8-core CPUs and 384GB of memory (32 total cores)
- **Benchmarks**: NAS Parallel Benchmarks 3.2.1, SPEC 2012, BOTS 1.1.2 (tied tasks only)
- All experiments executed with 32 threads to stress maximal concurrency
- All other settings left at default*
- All benchmarks minimally instrumented (Opari), or executed with `tau_exec`
  - Baseline, with Opari, with runtime support

Overhead comparisons – NPB 3.2.1

	GNU			INTEL			OPENUH		
NPB 3.2.1 Benchmark	Baseline	Opari	ORA	Baseline	Opari	OMPT	Baseline	Opari	ORA
BT.B	17.92	19.96	22.42	16.18	16.56	16.80	18.58	18.85	19.37
CG.B	7.49	7.68	7.51	7.31	7.96	7.63	8.02	8.93	9.39
EP.B	2.92	2.99	2.96	1.40	1.42	1.40	1.47	1.51	1.51
FT.B	3.17	3.13	3.13	3.21	3.27	3.29	3.60	3.57	3.65
IS.B	0.75	0.74	0.74	0.75	0.75	0.75	0.81	0.83	0.82
LU.B	12.37	**17.43**	12.38	12.25	**26.71**	13.22	13.31	**26.37**	14.32
LU-HP.B	22.00	**42.86**	**31.26**	19.24	**51.74**	**30.93**	23.13	**86.17**	**39.04**
MG.B	1.21	1.61	1.30	1.08	1.27	1.13	1.16	1.51	1.31
SP.B	16.02	16.10	16.53	14.55	15.53	15.59	16.61	18.07	19.62

- High overhead for POMP (LU, LU-HP)?
  - Many parallel region instances require many calls to the POMP API (at least 8 per)
- High overhead for ORA, OMPT (LU-HP)?
  - Many parallel region instances require many calls to the ORA callback function (at least 2, 4 per)

Opari Additional Events - LU

Name	Exclusive TIME	Inclusive TIME	Inclusive TIME / Call ▽	Calls	Child Calls
for enter/exit [OpenMP]	5.9177	54.9039	0.0002	300,736	300,736
barrier enter/exit [OpenMP]	3.3869	51.7246	0.0002	300,479	300,479
parallel begin/end [OpenMP]	11.4200	108.7703	0.0004	299,963	299,963
parallel fork/join [OpenMP]	0.2048	131.8197	0.0004	299,963	299,963
do (loop body) [OpenMP location: file:/ibrix/home3/khuck/src/NPB3.2.1/NPB3.2-O	3.8780	3.8780	0.0001	74,798	0
do (loop body) [OpenMP location: file:/ibrix/home3/khuck/src/NPB3.2.1/NPB3.2-O	3.9082	3.9082	0.0001	74,798	0
parallel (barrier enter/exit) [OpenMP location: file:/ibrix/home3/khuck/src/NPB3.2.	11.2482	11.2482	0.0002	74,798	0
parallel (barrier enter/exit) [OpenMP location: file:/ibrix/home3/khuck/src/NPB3.2.	11.4249	11.4249	0.0002	74,798	0
paralleldo (barrier enter/exit) [OpenMP location: file:/ibrix/home3/khuck/src/NPB3.	11.6627	11.6627	0.0002	74,798	0
paralleldo (barrier enter/exit) [OpenMP location: file:/ibrix/home3/khuck/src/NPB3.	11.8941	11.8941	0.0002	74,798	0
paralleldo (loop body) [OpenMP location: file:/ibrix/home3/khuck/src/NPB3.2.1/NP	5.4058	17.9379	0.0002	74,798	74,798
paralleldo (loop body) [OpenMP location: file:/ibrix/home3/khuck/src/NPB3.2.1/NP	5.4488	18.2168	0.0002	74,798	74,798
parallel (parallel begin/end) [OpenMP location: file:/ibrix/home3/khuck/src/NPB3.2	4.7560	22.1298	0.0003	74,798	149,596
parallel (parallel begin/end) [OpenMP location: file:/ibrix/home3/khuck/src/NPB3.2	4.6536	22.3450	0.0003	74,798	149,596
paralleldo (parallel begin/end) [OpenMP location: file:/ibrix/home3/khuck/src/NPB3	4.0559	23.4379	0.0003	74,798	74,798
paralleldo (parallel begin/end) [OpenMP location: file:/ibrix/home3/khuck/src/NPB3	4.0142	23.6783	0.0003	74,798	74,798
parallel (parallel fork/join) [OpenMP location: file:/ibrix/home3/khuck/src/NPB3.2.1	4.3245	30.6811	0.0004	74,798	74,798
parallel (parallel fork/join) [OpenMP location: file:/ibrix/home3/khuck/src/NPB3.2.1	4.4580	30.7417	0.0004	74,798	74,798
JACLD [{jacld.pomp.f} {6,7}-{387,9}]	0.0480	30.7774	0.0004	74,798	74,798
JACU [{jacu.pomp.f} {6,7}-{386,9}]	0.0474	30.8371	0.0004	74,798	74,798
paralleldo (parallel fork/join) [OpenMP location: file:/ibrix/home3/khuck/src/NPB3.:	4.2251	31.9492	0.0004	74,798	74,798
BUTS [{buts.pomp.f} {5,7}-{271,9}]	0.0463	32.0417	0.0004	74,798	74,798
paralleldo (parallel fork/join) [OpenMP location: file:/ibrix/home3/khuck/src/NPB3.:	4.3732	32.3329	0.0004	74,798	74,798
BLTS [{blts.pomp.f} {5,7}-{272,9}]	0.0469	32.4282	0.0004	74,798	74,798
master begin/end [OpenMP]	0.0253	0.0389	0.0000	1,013	1,013
master (master begin/end) [OpenMP location: file:/ibrix/home3/khuck/src/NPB3.2	0.0002	0.0002	0.0000	253	0

~34,239,023 timer invocations!

Collector API Extra Events – LU-HP

Name	Exclusive TIME	Inclusive TIME	Inclusive TIME / Call	Calls ▽	Child Calls
OpenMP_FINISH_TASK [THROTTLED]	3.5530	3.5530	0.0000	298,959	0
JACLD [{jacld.f} {5,7}-{366,9}]	0.7801	34.1022	0.0005	74,798	74,798
JACU [{jacu.f} {5,7}-{365,9}]	0.7720	34.0634	0.0005	74,798	74,798
BUTS [{buts.f} {4,7}-{248,9}]	0.7775	33.9265	0.0005	74,798	74,296
BLTS [{blts.f} {4,7}-{249,9}]	0.7483	33.5503	0.0004	74,798	74,296
OpenMP_PARALLEL_REGION: JACU [{jacu.f} {5,7}-{365,9}]	11.3871	24.8788	0.0003	74,798	149,596
OpenMP_PARALLEL_REGION: JACLD [{jacld.f} {5,7}-{366,9}]	11.1563	24.7111	0.0003	74,798	149,596
OpenMP_IMPLICIT_BARRIER: JACLD [{jacld.f} {5,7}-{366,9}]	12.6658	12.6658	0.0002	74,798	0
OpenMP_IMPLICIT_BARRIER: JACU [{jacu.f} {5,7}-{365,9}]	12.6106	12.6106	0.0002	74,798	0
OpenMP_PARALLEL_REGION: BUTS [{buts.f} {4,7}-{248,9}]	10.9541	24.4962	0.0003	74,296	148,592
OpenMP_PARALLEL_REGION: BLTS [{blts.f} {4,7}-{249,9}]	10.8518	24.3023	0.0003	74,296	148,592
OpenMP_IMPLICIT_BARRIER: BUTS [{buts.f} {4,7}-{248,9}]	12.6434	12.6434	0.0002	74,296	0
OpenMP_IMPLICIT_BARRIER: BLTS [{blts.f} {4,7}-{249,9}]	12.5688	12.5688	0.0002	74,296	0
OpenMP_MASTER_REGION: RHS [{rhs.f} {5,7}-{448,9}]	0.0077	0.0077	0.0000	1,012	0
OpenMP_IMPLICIT_BARRIER: RHS [{rhs.f} {5,7}-{448,9}]	1.7619	1.7619	0.0023	759	0
OpenMP_PARALLEL_REGION: SSOR [{ssor.f} {4,7}-{273,9}]	0.5451	0.9610	0.0019	504	1,008
OpenMP_IMPLICIT_BARRIER: SSOR [{ssor.f} {4,7}-{273,9}]	0.4149	0.4149	0.0008	504	0
RHS [{rhs.f} {5,7}-{448,9}]	0.0060	4.9819	0.0197	253	253
OpenMP_PARALLEL_REGION: RHS [{rhs.f} {5,7}-{448,9}]	3.1801	4.9436	0.0195	253	1,043.625
TIMER_CLEAR [{timers.f} {4,7}-{17,9}]	0.0000	0.0000	0.0000	44	0

Obvious takeaway: unless needed, Events will add overhead for lightweight loops

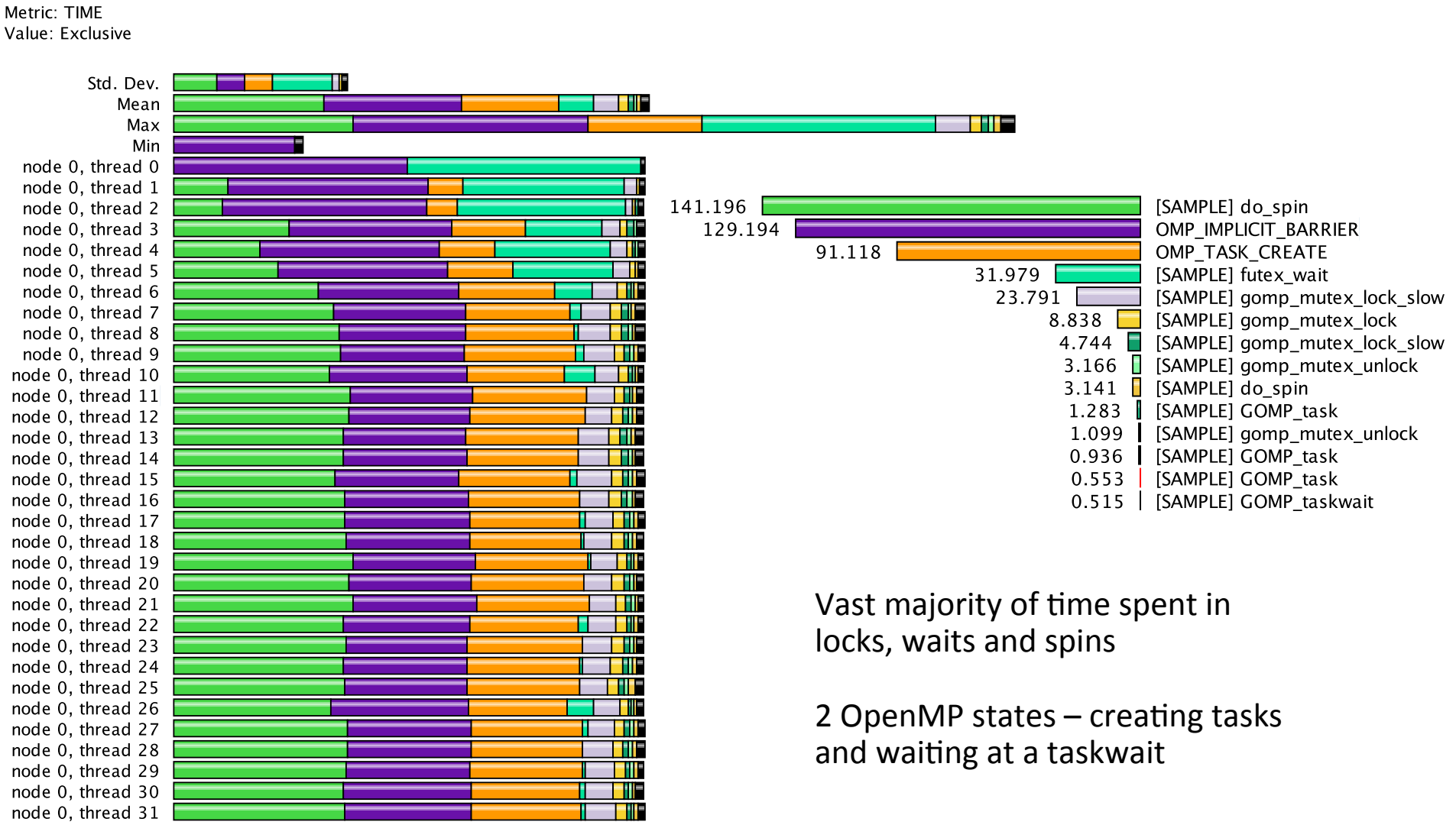
~3,456,152 extra timers!

Overhead comparisons – BOTS 1.1.2

	GNU			INTEL			OPENUH		
BOTS 1.1.2 Benchmark	Baseline	Opari	ORA	Baseline	Opari	OMPT	Baseline	Opari	ORA
alignment.single 1000	70.59	73.99	70.77	48.85	48.90	48.86	56.19	56.44	56.20
fft 134217728	**132.55**	124.93	127.94	3.57	5.52	4.61	11.69	13.05	13.78
fib 30	**23.91**	27.11	28.89	0.10	1.71	1.33	0.97	4.16	3.18
floorplan 15	**226.58**	225.71	234.35	0.70	**11.44**	**6.06**	18.95	**24.09**	21.38
health small	**38.78**	37.04	29.47	0.46	1.78	0.98	2.41	2.70	2.72
nqueens 12	**120.18**	134.28	156.44	0.13	**5.14**	**3.87**	10.17	11.92	11.18
sort 134217728	**20.76**	26.83	22.73	2.44	2.49	2.55	2.90	3.15	3.17
sparselu.single 100x100	4.11	4.19	4.10	11.49	11.52	11.60	3.82	3.85	3.83
strassen 8192	0.02	0.02	0.03	0.03	0.03	0.04	0.01	0.04	0.03
uts small	**53.54**	**212.57**	**218.78**	11.00	**64.84**	**34.26**	*segv*	n/a	n/a

- GNU horrible baseline performance
  - Spawning, waiting for small granularity tasks with taskwait
  - Lock contention in the task scheduler
- OpenUH measurement overhead (floorplan)
- Intel measurement overhead (floorplan, nqueens, uts)
  - Millions of short-lived, lightweight tasks that take longer to measure than to execute
  - Sampling gives similar measurement results, without overhead

GCC BOTS Floorplan Profile (samples)



Vast majority of time spent in locks, waits and spins

2 OpenMP states – creating tasks and waiting at a taskwait

Task Creation (scheduling, taskwait similar)

```
gomp_mutex_lock (&team->task_lock);
 if (parent->children)
{
 task->next_child = parent->children;
 task->prev_child = parent->children->prev_child;
 task->next_child->prev_child = task;
 task->prev_child->next_child = task;
}
 else
{
 task->next_child = task;
 task->prev_child = task;
}
 parent->children = task;
 if (team->task_queue)
{
 task->next_queue = team->task_queue;
 task->prev_queue = team->task_queue->prev_queue;
 task->next_queue->prev_queue = task;
 task->prev_queue->next_queue = task;
}
 else
{
 task->next_queue = task;
 task->prev_queue = task;
 team->task_queue = task;
}
 ++team->task_count;
 gomp_team_barrier_set_task_pending (&team->barrier);
 do_wake = team->task_running_count + !parent->in_tied_task
  < team->nthreads;
gomp_mutex_unlock (&team->task_lock);
```

- 387 lines of code
- 150+ are inside locks
- 6 locations
- 25+ lines per locked block!

!!!

GNU Task Scheduler modifications

- 4.8.2 implementation uses double-linked lists and locks for task queue and child dependencies
- Re-implemented with lock-free circular queues and smart pointers (for memory management)
- ***4.9 implementation will be worse*** – 2 more queues added for task groups, task dependencies
- Our implementation needs debugging, hardening, then will be submitted to GNU maintainers

Benchmark	Baseline	Lock-free
alignment	70.59	0.73
fft	132.55	4.18
fib	23.91	0.80
floorplan	226.58	1.42
health	38.78	5.72
nqueens	120.18	0.73
sort	20.76	2.00
sparselu	4.11	4.05
strassen	0.02	11.85
uts	53.54	45.73

Overhead comparisons – SPEC 2012

	GNU			INTEL			OPENUH		
SPEC 2012 Benchmark	Baseline	Opari	ORA	Baseline	Opari	OMPT	Baseline	Opari	ORA
351.bwaves	23.99	24.38	24.18	*segv*	n/a	n/a	*segv*	n/a	n/a
352.nab	24.00	25.85	29.41	20.40	22.96	21.40	35.13	35.80	35.68
357.bt331	20.69	n/a	21.44	18.27	22.49	18.75	21.41	n/a	n/a
358.botsalgn	1.07	1.21	3.65	1.07	1.25	1.07	1.05	1.08	1.15
359.botsspar	2.67	2.97	2.87	2.78	2.31	1.88	2.87	1.12	2.89
360.ilbdc (test)	**502.76**	**282.18**	**313.58**	13.33	13.47	13.48	14.16	14.36	15.65
362.fma3d	14.31	15.05	47.73	13.15	13.58	13.16	19.88	24.74	24.57
363.swim	10.18	11.13	10.21	10.26	10.97	10.29	22.36	n/a	23.00
367.imagick	19.40	19.44	19.52	5.84	5.86	6.04	145.97	145.85	146.06
370.mgrid331	0.72	1.37	0.86	0.65	1.85	1.10	0.75	2.71	2.57
371.applu331	3.82	12.26	6.16	3.45	13.07	4.29	*segv*	n/a	n/a
372.smithwa	1.89	2.19	2.18	1.45	1.45	1.50	2.58	2.62	2.61

- GNU horrible performance (360.ibdc)
  - TAU ORA profile shows vast majority of time in the GOMP scheduler (overhead state)
  - Pragma specifies “runtime” scheduler
  - Default scheduler for GOMP: “dynamic, 1”
  - After setting to OMP_SCHEDULER to “static”, times reduced to 39.25, 43.08 and 42.25 seconds

SPEC 2012 360.ilbdc ORA profile

TAU: ParaProf: Mean Statistics – /Users/khuck/Documents/papers/TAU_papers/2014-IWOMP-OpenMP-Survey/...

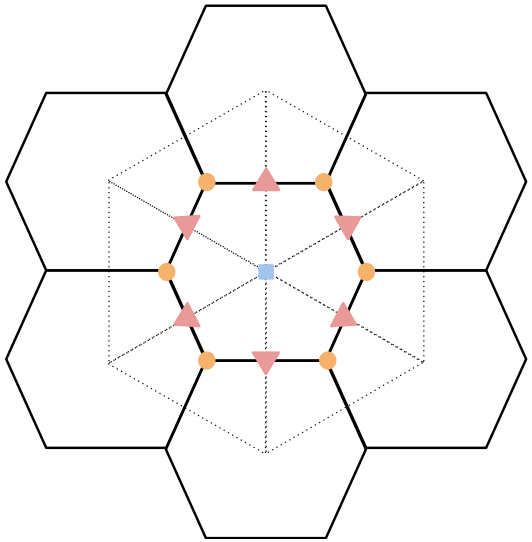
Name	Exclusive TIME	Inclusive TIME ▽	Calls
ILBDC_KERNEL	0.018	333.705	1
OMP_IDLE			
[CONTEXT] OMP_IDLE	0	1.148	114.774
OMP_IMPLICIT_BARRIER			
[CONTEXT] OMP_IMPLICIT_BARRIER	0	0.06	6
OMP_OVERHEAD			
[CONTEXT] OMP_OVERHEAD			2
OMP_SERIAL			
[CONTEXT] OMP_SERIAL			79
OMP_WORKING			
[CONTEXT] OMP_WORKING	0	332.51	33,242.781
[SAMPLE] UNRESOLVED /usr/lib64/libgomp.so.1.0.0	245.473	245.473	24,540.625
[SAMPLE] __mod_relax_MOD_relax_collstream.omp_fn.0	42.61	42.61	4,260.219
[SAMPLE] __mod_relax_MOD_relax_collstream.omp_fn.0	6.882	6.882	688.312
[SAMPLE] __mod_relax_MOD_relax_collstream.omp_fn.0	6.661	6.661	665.875
[SAMPLE] __mod_relax_MOD_relax_collstream.omp_fn.0	5.177	5.177	517.688
[SAMPLE] __mod_relax_MOD_relax_collstream.omp_fn.0	3.429	3.429	342.875
[SAMPLE] __mod_relax_MOD_relax_collstream.omp_fn.0	1.949	1.949	194.906
[SAMPLE] __mod_relax_MOD_relax_collstream.omp_fn.0	1.668	1.668	166.719
[SAMPLE] __mod_relax_MOD_relax_collstream.omp_fn.0	1.399	1.399	139.969
[SAMPLE] __mod_relax_MOD_relax_collstream.omp_fn.0	1.159	1.159	115.875
[SAMPLE] __mod_relax_MOD_relax_collstream.omp_fn.0	1.076	1.076	107.625
[SAMPLE] __mod_relax_MOD_relax_collstream.omp_fn.0	1.076	1.076	107.531
[SAMPLE] UNRESOLVED /ibrix/users/home3/khuck/src/omp20	0.943	0.943	94.25

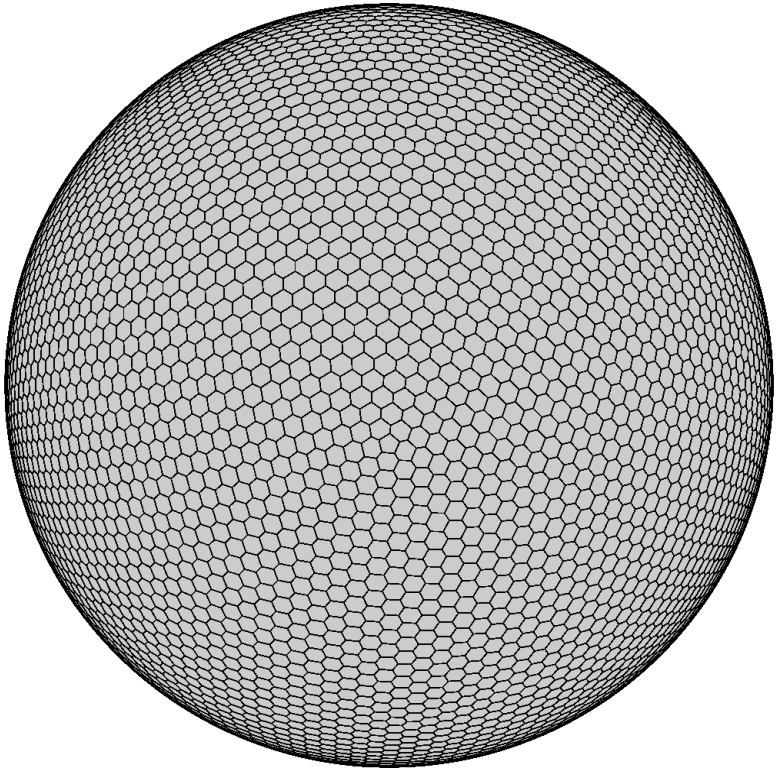
Not really WORKING: this time should be measured as OVERHEAD. This is a limitation of the wrapper approach

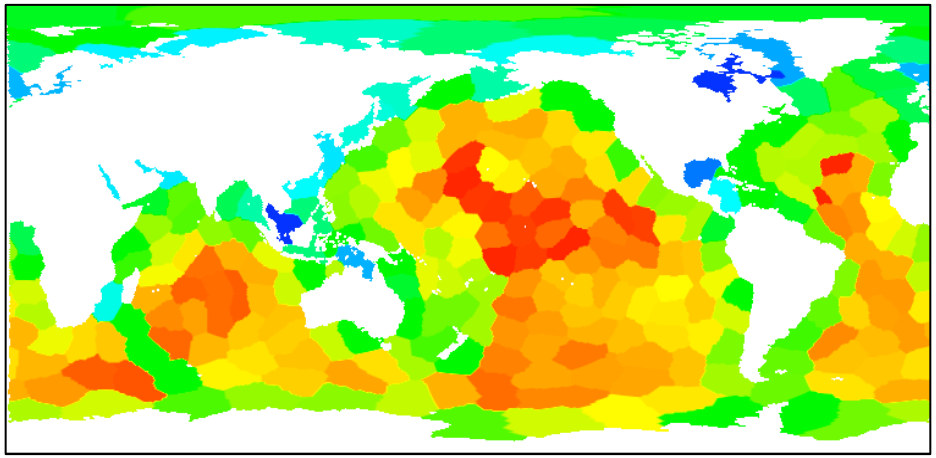
Application Example: MPAS-Ocean

- The Model for Prediction Across Scales (NCAR, LANL)
- Framework for rapid prototyping of single-component climate system models
- Currently evaluating OpenMP pragmas to efficiently increase concurrency
- TAU was used to evaluate MPI+OpenMP approach, suggest improvements
- Tests performed by Doug Jacobsen (LANL), Kevin Huck and Sameer Shende (U. Oregon)
- Linked with TAU, "no" instrumentation, Intel + OMPT
- Observations, optimizations found:
  1. MPI block decomposition + OpenMP element decomposition reduces total instructions in computational regions (~10% faster) when compared to MPI block decomposition alone
  2. Guided schedule balances work across threads (~6% faster)
- Evaluation of OpenMP implementation is ongoing…
- http://mpas-dev.github.io

MPAS Unstructured Mesh Structure



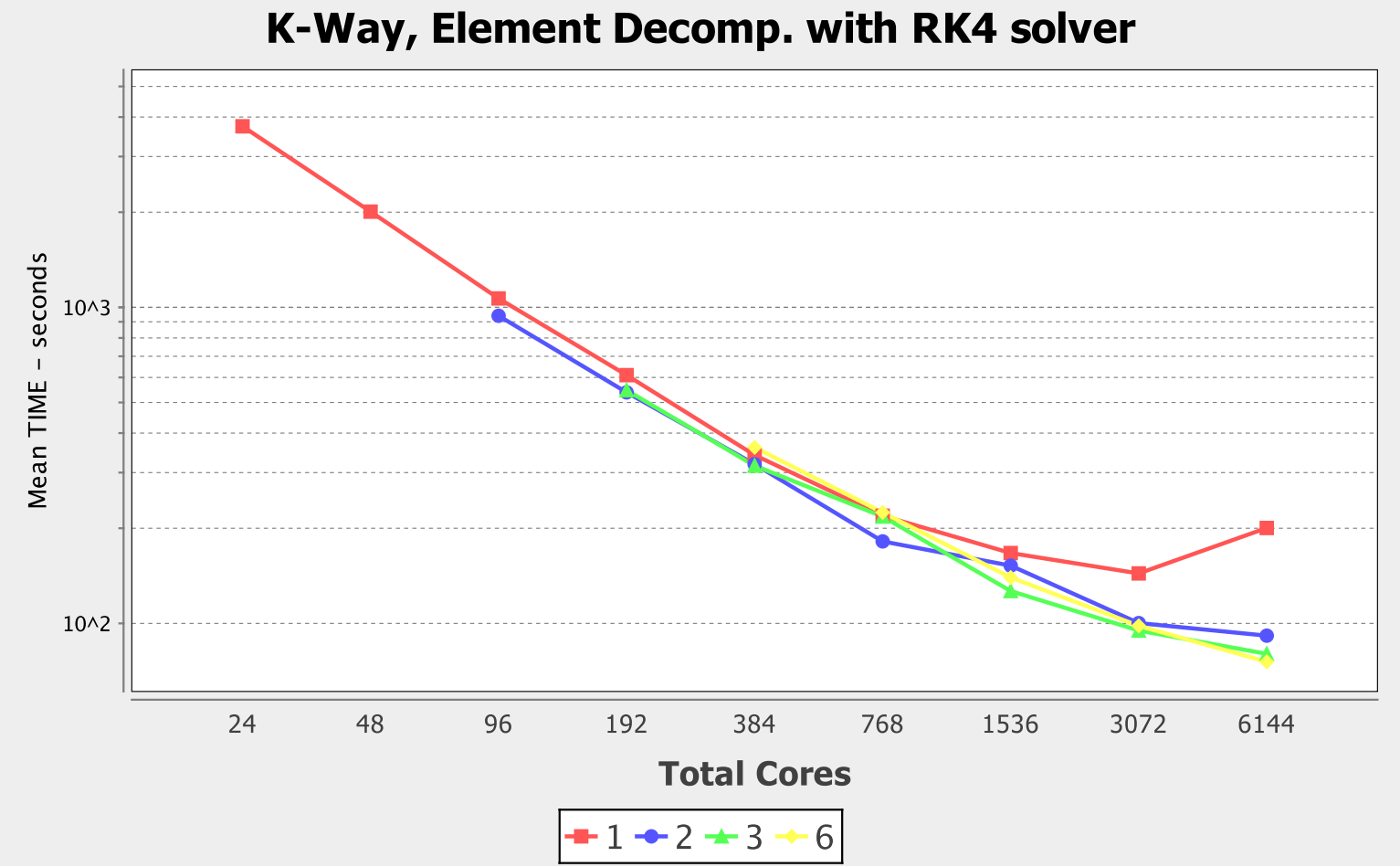




Spherical Centroidal Voronoi Tesselations (SCVT)

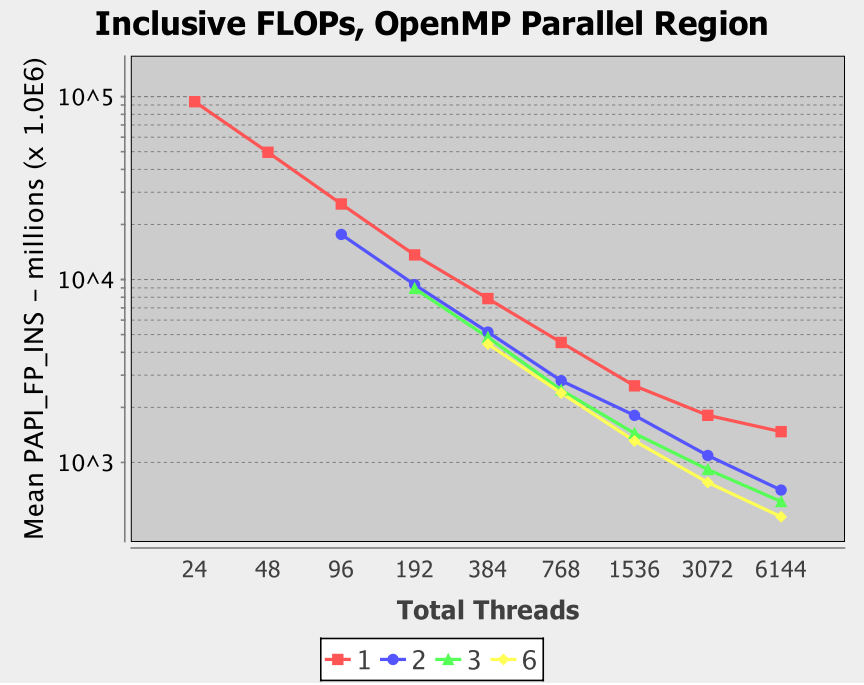
Partitioned into blocks, with 3 layers of halo (ghost) cells from neighbor regions

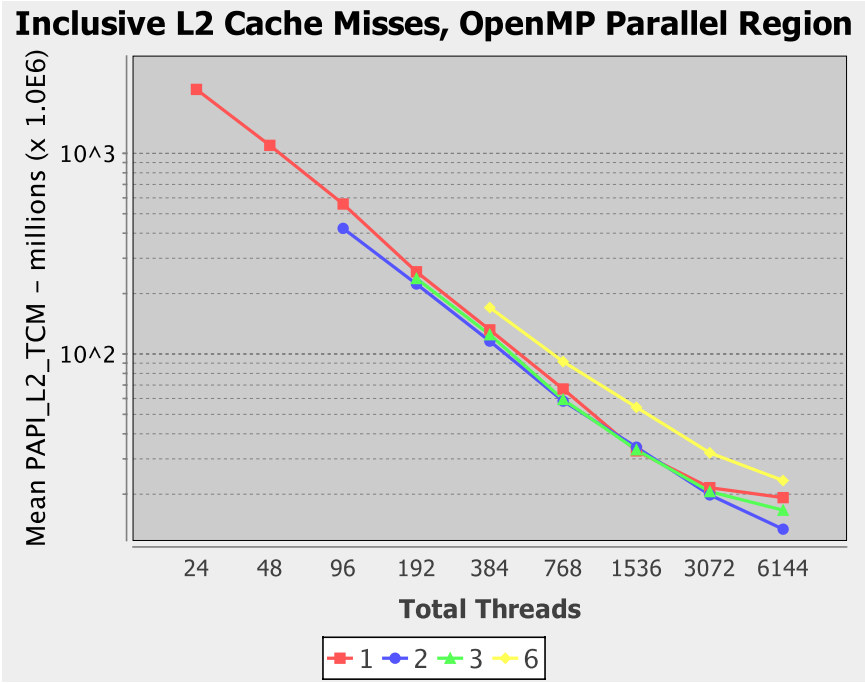
MPAS-Ocean Scaling Behavior



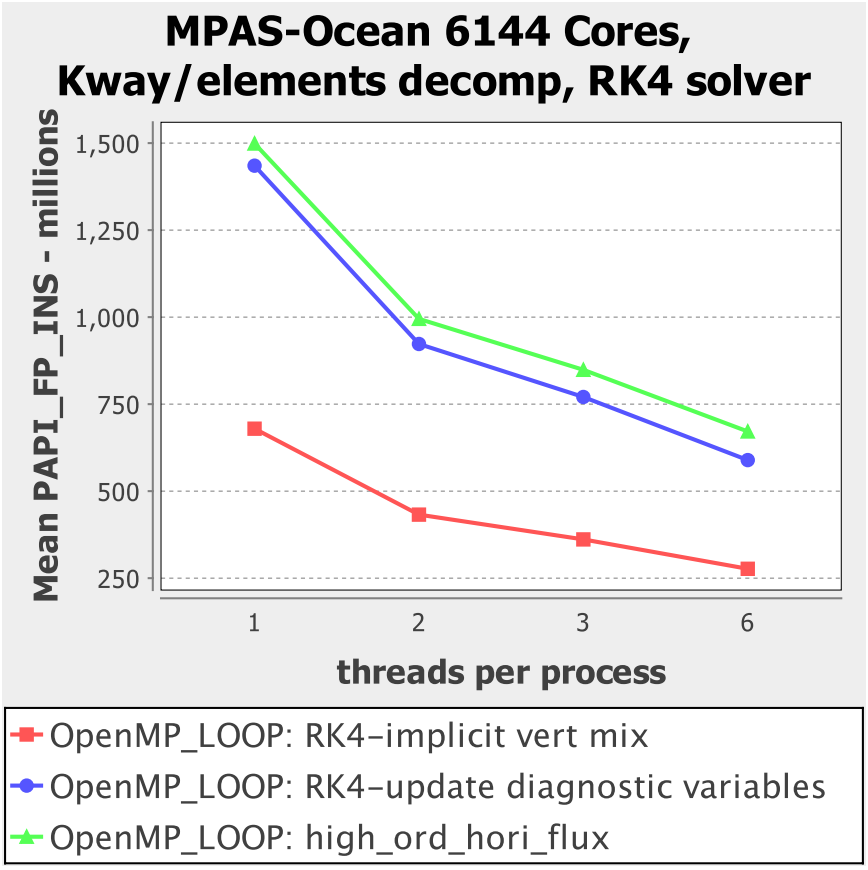
Executed on Hopper @ NERSC, a Cray XE6 with 2 twelve-core AMD 'MagnyCours' 2.1-GHz processors per node
Intel 13 compilers with prototype OMPT support

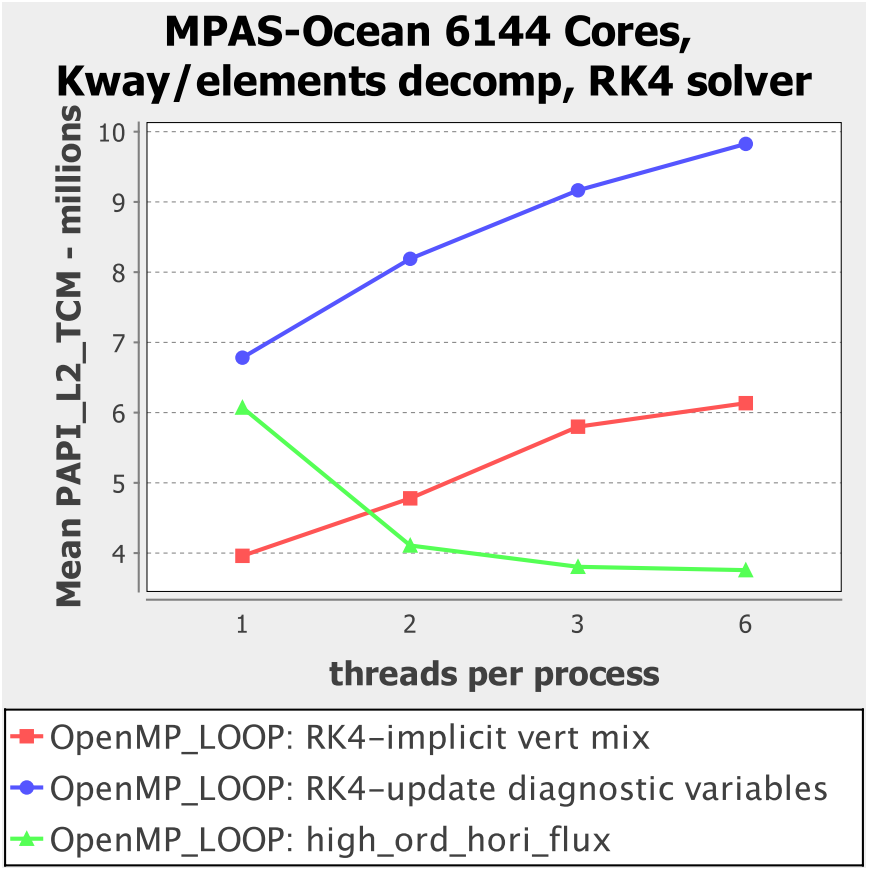
OpenMP Scaling Details





OpenMP Scaling Details





Summary

- Several OpenMP measurement options to choose from
  - POMP/Opari is the most portable alternative, but limited
- Obviously, best implementation support comes from within the runtime
- OMPT is a promising opportunity for OpenMP runtime tool support
- Still need to be careful about processing too many lightweight events (especially tasks) – even with OMPT
- OMPT needs an update to support OpenMP 4.0
- Acknowledgements: The research was supported by grants from the National Science Foundation and the US Department of Energy