A Proposal to OpenMP for Addressing the CPU Oversubscription Challenge

Yonghong Yan[1,5], Jeff R. Hammond[2,5], Chunhua Liao[3], and Alexandre E. Eichenberger[4,5]

[1] Department of Computer Science and Engineering, Oakland University
[2] Parallel Computing Lab, Intel Corp.
[3] Center for Applied Scientific Computing, Lawrence Livermore National Laboratory
[4] Thomas J. Watson Research Center, IBM
[5] OpenMP Interoperability Language Subcommittee

The 12th International Workshop on OpenMP, October 2016, Nara, Japan

Acknowledgement: OpenMP Language Committee,
Terry Wilmarth and Brian Bliss from Intel,
NSF under project SHF-1409946 and SHF-1551182,
Prepared by LLNL under Contract DE-AC52-07NA27344 (LLNL-PRES-703327)

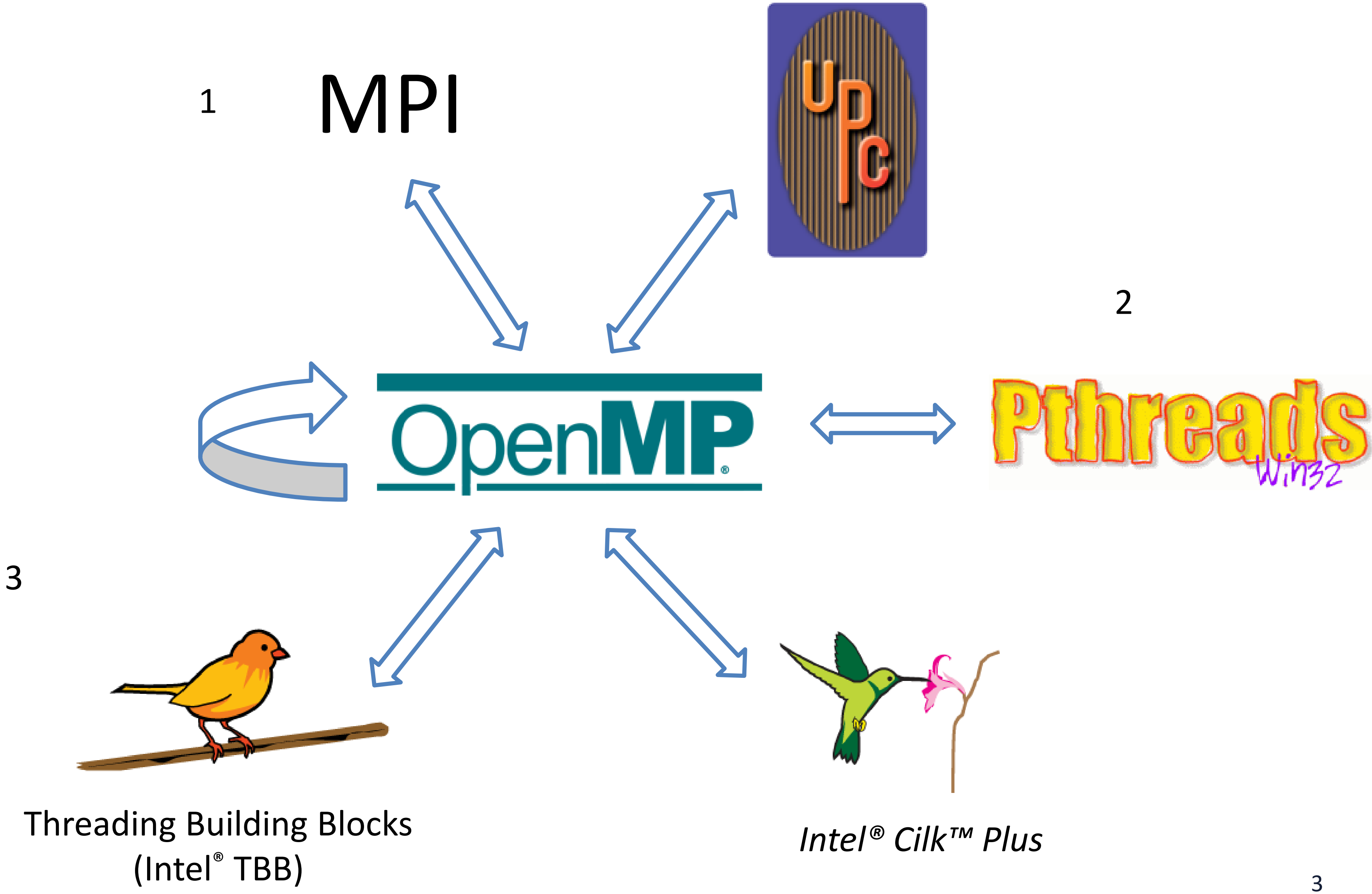
Oakland UNIVERSITY

LAWRENCE LIVERMORE NATIONAL LABORATORY

Contents

- Use Cases and Challenges of Interoperability
- Runtime extensions for addressing CPU oversubscription
- Implementation and Evaluation
- Related Work
- Conclusions

Use cases of OpenMP Interoperating with Others



Issues with No or Poor Interoperability

- CPU
  - Active oversubscription: Claiming or requesting more threads than what are available by the system.
  - Passive oversubscription: Thread resources are not released after parallel execution.
- Memory
  - Conflicting Thread Affinity: when the OpenMP runtime binds threads data to certain memory places (cache or NUMA region) that are already occupied by the affinity requests of another runtime
- Performance and efficiency penalty
  - Degraded CPU performance
  - Increased memory access latency

Limitations of the Current Specification

- OMP_DYNAMIC and void omp_set_dynamic(int val)
  - partially address the active oversubscription issue, depending on runtime implementation
  - does not address thread resource release (passive oversubscription)
- OMP_WAIT_POLICY and ACTIVE|PASSIVE
  - only allows one time setting when the program starts,
  - preventing the dynamic adjustment of thread waiting behavior during the execution.

Limitations of the current specification (Cont.)

- OMP_THREAD_LIMIT environment
  - an option to set the max number of OMP threads for the whole program
  - does not provide an interface to adjust the upper bound of the threads for an OpenMP program during program execution
- The global scope of its environment variables
  - users cannot set different wait policies for multiple concurrent parallel regions in nested parallel cases.

Summary of Our Extensions

Fine-gain wait policy definitions and changes

Terminate/suspend runtime

Integrate with user threads

```
typedef enum omp_wait_policy {
  OMP_SPIN_BUSY_WAIT = 1,   /* 0x1 */
  OMP_SPIN_PAUSE_WAIT = 2,  /* 0x10 */
  OMP_SPIN_YIELD_WAIT = 4,  /* 0x100 */
  OMP_SUSPEND_WAIT = 8,     /* 0x1000 */
  OMP_TERMINATE = 16, /* 0x10000 */

  OMP_ACTIVE_WAIT = OMP_SPIN_PAUSE_WAIT,
  OMP_PASSIVE_WAIT = OMP_SUSPEND_WAIT;
} omp_wait_policy_t;

int omp_get_num_threads_runtime(omp_wait_policy_t state);

void omp_set_wait_policy(omp_wait_policy_t wait_policy);
int omp_get_wait_policy(void);

int omp_quiesce(omp_wait_policy_t state);

typedef void * omp_thread_t;
int omp_thread_create (omp_thread_t * th, int place,
   void *(*start_routine)(void *), void *arg, void * new_stack);
void omp_thread_exit(void *value_ptr);
int omp_thread_join(omp_thread_t thread, void **value_ptr);
```

Thread wait policies, sub-policies, and semantics

- Current spec.: ACTIVE vs. PASSIVE
  - consuming CPU power or not
- Fine-grain specification of waiting behaviors

Wait Policy		Description	Pseudo Code
ACTIVE	SPIN_BUSY	Busy wait in user level	while (!finished()) ;
	SPIN_PAUSE	Busy wait while pausing CPU	while (!finished()) cpu_pause();
PASSIVE	SPIN_YIELD	Busy wait with yield	while (!finished()) sched_yield();
	SUSPEND	Thread sleeps. Others wake it up.	mutex_wait(); mutex_wake();
	TERMINATE	Thread terminates.	pthread_exit();

void omp_set_wait_policy (omp_ wait_policy_t p)

- Enable dynamic adjustment of wait policies during the execution of OMP programs
  - runtime or users can exploit this info to mitigate oversubscriptions
- Binding thread set:
  - Called inside a sequential region: all the threads of the binding contention group.
  - Called inside a parallel region, the calling thread only.
- Impact on ICV
  - Call inside a sequential region: changes the wait-policy-var ICV
  - Call inside a parallel region: supersedes the waiting behavior setting by the ICV for the calling thread

int omp_get_num_threads_runtime (omp_ wait_policy_t)

- Returns the number of runtime threads that are under the specified policy
  - runtime or users can exploit this info to make decisions
- Binding threads: the threads queried upon
  - Called inside a sequential region: the contention group
  - Called inside a parallel region: the team

Interface to terminate/suspend runtime

- Current specification:
  - No interface to control over runtime termination/suspension
- Our extension: int omp_quiesce (omp_wait_policy_t s)
  - quiesces all OpenMP threads of the runtime
  - threads are put into a status specified by the argument
    - either OMP_SUSPEND_WAIT or OMP_TERMINATE.
- Benefit
  - frees up resources in a flexible way, avoid conflicts with other runtimes/APIs
    - address oversubscription

Interacting with user threads: omp_thread_create()

- Current situation:
  - OMP runtimes are not aware of user threads
  - Easily cause oversubscription
- Our solution: runtime interface to create user threads
  - int omp_thread_create(omp_thread_t, int place, void* (*sroutine) (void*), void * args, void * new_stack)
  - Enable better integration of OpenMP threads and user threads
  - Runtime awareness of user threads → better manage threads overall
- Accompanying routines
  - void omp_thread_exit(void *value_ptr)
  - int omp_thread_join (omp_thread_t thread, void ** value_ptr);

Implementation and Evaluation

- Implemented in LLVM/Intel Runtime
  - LLVM compiler version 3.8.0
    - https://github.com/passlab/llvm-openmp
  - Ongoing implementation in GCC 6.1.0 OpenMP also
- Platform
  - Intel Xeon E5-2699v3 (Haswell) processors with total 36 cores supporting 72 threads.
- Benchmarks
  - Microbenchmarks: parallel regions
  - Microkernels calling new APIs: Written in pthreads and OpenMP

Overhead

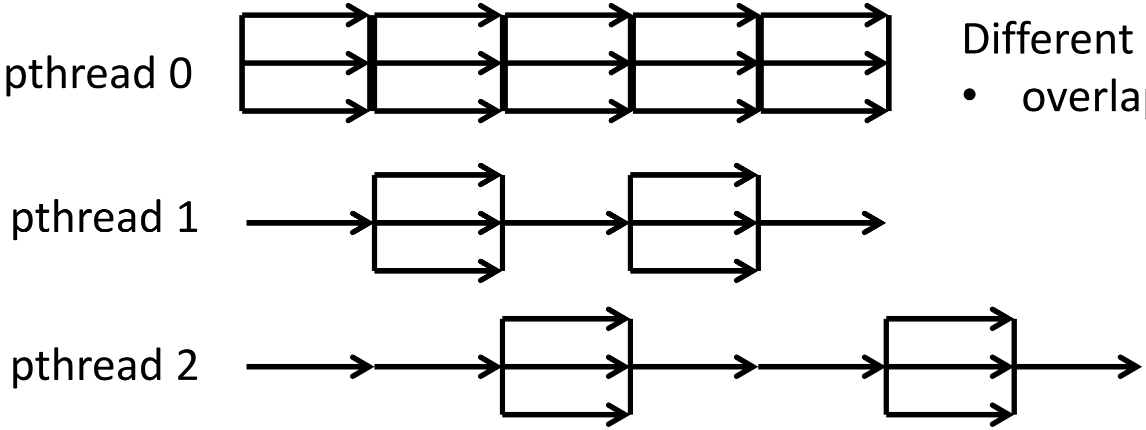
- Overhead of creation of 2nd and later parallel regions after 1st parallel region ends with different wait policies
- Overhead of the new runtime routines themselves

		Number of OpenMP Threads								
Overhead (us)	**Policies**	**1**	**2**	**4**	**8**	**16**	**32**	**36**	**48**	**64**
Additional overhead for OpenMP parallel startup when applying wait policy	**ACTIVE**	0	0	0	0	3	4	4	4	6
	PASSIVE(SPIN_YIELD)	0	0	0	0	2	4	5	3	5
	PASSITVE(SUSPEND)	0	15	23	39	44	66	69	74	94
	QUIESCE(TERMINATE)	4383	4493	4530	6414	16498	35303	36890	60160	89746
Overhead for set_wait_policy/quiesce overhead	**ACTIVE**	0	1	1	0	0	1	1	6	6
	PASSIVE(SPIN_YIELD)	0	1	0	0	1	0	2	6	4
	PASSITVE(SUSPEND)	0	0	0	0	0	0	3	6	5
	QUIESCE(TERMINATE)	34	143	159	219	397	711	886	751	1173

Oversubscription Test Example

- Hybrid Pthreads + OpenMP: multiple Pthreads, each executes:

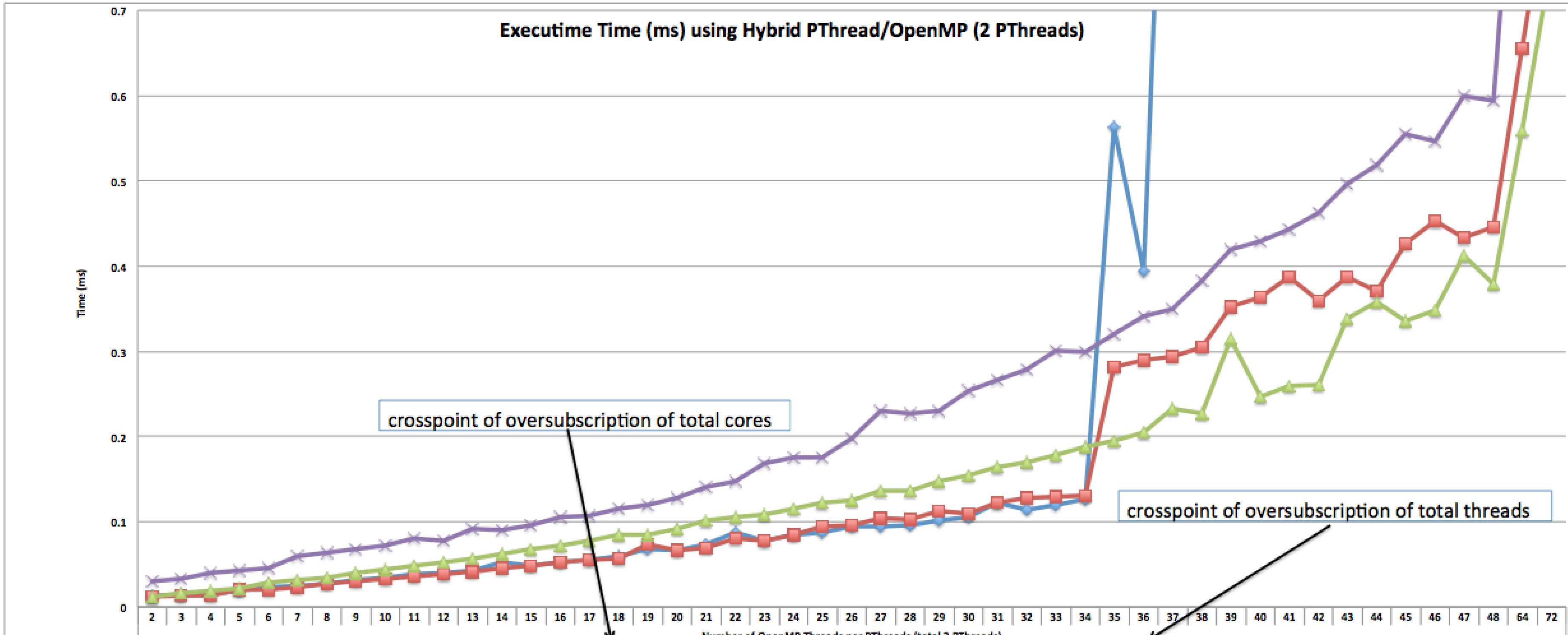
```
int user_thread_id = (int) ptr;
for (int i=0; i<NUM_ITERATIONS; i++) {
    busy_waiting(user_thread_id*3000);
#pragma omp parallel num_threads(num_ompthreads)
    {
        busy_waiting(3000); /* act as computation */
    }
    omp_set_wait_policy(policy);
}
```



Different waiting time to approximate

- overlapped sequential and parallel regions

2 Pthreads, each starting 2 to 72 OMP threads



	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
SPIN_BUSY	0.013	0.014	0.015	0.019	0.023	0.025	0.027	0.031	0.034	0.038	0.04	0.043	0.052	0.049	0.053	0.056	0.06
SPIN_PAUSE	0.012	0.013	0.014	0.02	0.021	0.023	0.027	0.03	0.033	0.036	0.038	0.041	0.046	0.049	0.052	0.056	0.057
SPIN_YIELD	0.012	0.016	0.019	0.022	0.029	0.031	0.035	0.04	0.044	0.049	0.052	0.057	0.062	0.068	0.072	0.077	0.084
SUSPEND	0.03	0.033	0.04	0.043	0.046	0.06	0.064	0.068	0.072	0.08	0.077	0.091	0.09	0.096	0.105	0.107	0.115

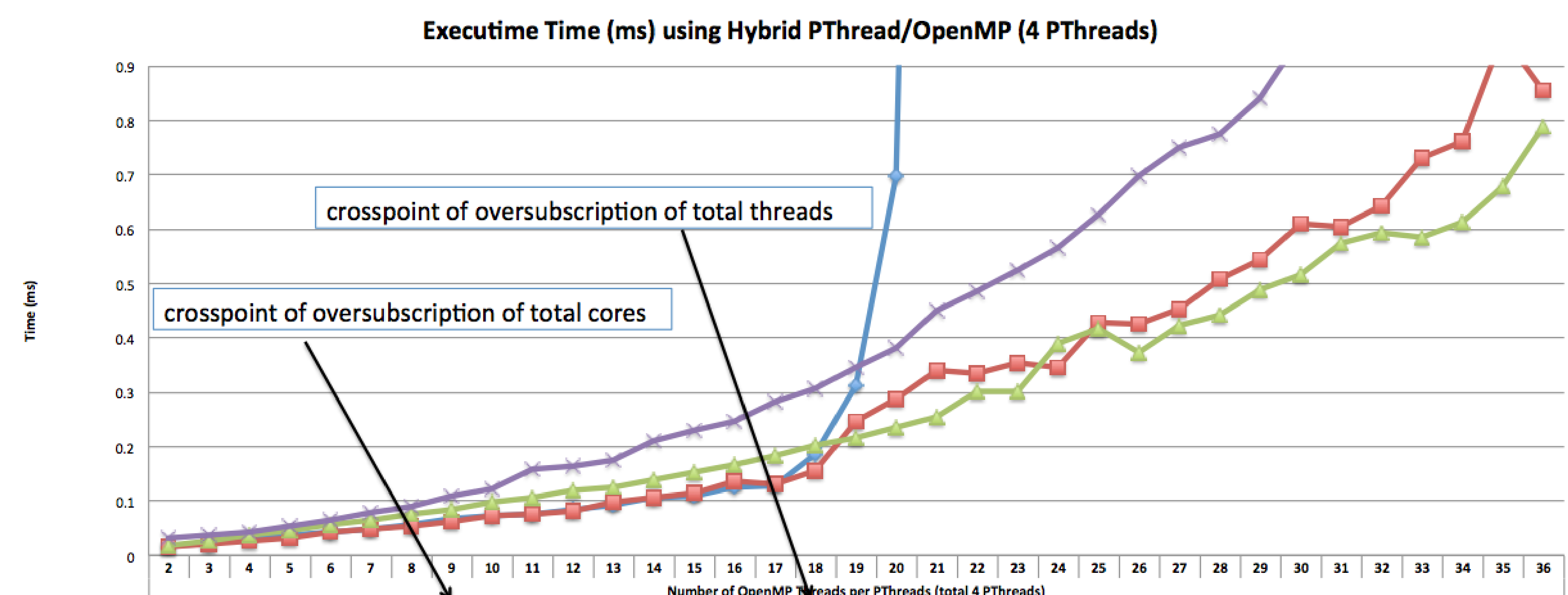
	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
SPIN_BUSY	0.068	0.067	0.073	0.087	0.078	0.084	0.088	0.094	0.094	0.096	0.102	0.105	0.122	0.114	0.119	0.127	0.56
SPIN_PAUSE	0.073	0.067	0.069	0.081	0.078	0.085	0.095	0.096	0.104	0.103	0.113	0.11	0.122	0.128	0.129	0.13	0.282
SPIN_YIELD	0.085	0.092	0.101	0.105	0.109	0.115	0.123	0.125	0.136	0.136	0.148	0.155	0.164	0.17	0.178	0.188	0.195
SUSPEND	0.12	0.128	0.14	0.148	0.169	0.176	0.175	0.198	0.23	0.227	0.23	0.253	0.266	0.278	0.301	0.299	0.32

	36	37	38	39	40	41	42	43	44	45	46	47	48	64	72
SPIN_BUSY	0.40	1.22	1.21	3.63	9.39	14.9	15.3	14.1	17.6	20.0	18.8	19.8	20.2	29.9	34.1
SPIN_PAUSE	0.29	0.294	0.305	0.353	0.364	0.387	0.36	0.387	0.37	0.427	0.453	0.433	0.446	0.656	0.868
SPIN_YIELD	0.205	0.233	0.227	0.315	0.247	0.259	0.26	0.339	0.358	0.336	0.349	0.412	0.379	0.559	0.757
SUSPEND	0.341	0.35	0.383	0.42	0.429	0.443	0.463	0.496	0.518	0.555	0.547	0.6	0.594	1.038	1.341

2*18 = 36 threads –> all cores are used

2*36 = 72 threads –> all hardware threads cores are used

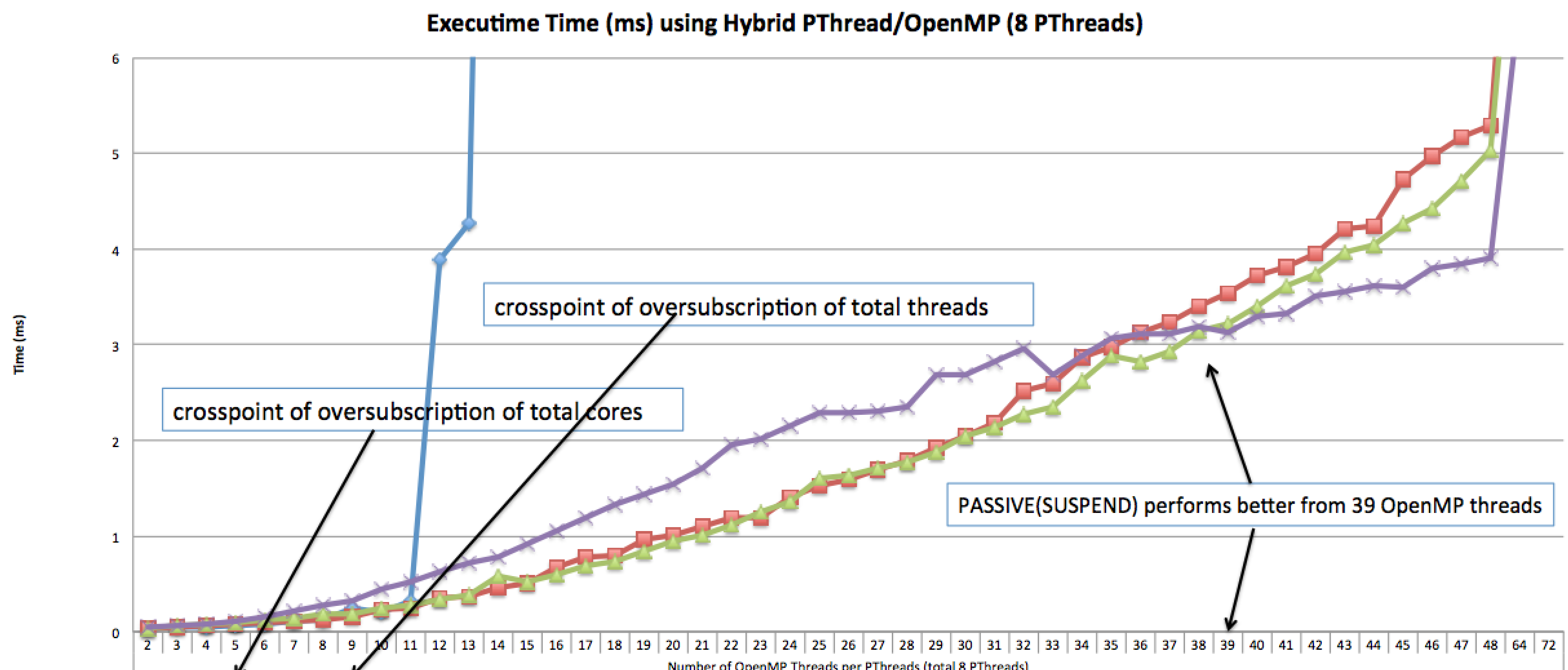
4 PThreads



	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
SPIN_BUSY	0.017	0.022	0.031	0.038	0.043	0.048	0.058	0.067	0.073	0.077	0.084	0.092	0.107	0.108	0.127	0.128	0.187
SPIN_PAUSE	0.016	0.022	0.026	0.031	0.044	0.049	0.055	0.061	0.073	0.077	0.082	0.097	0.106	0.116	0.136	0.132	0.155
SPIN_YIELD	0.018	0.027	0.038	0.046	0.058	0.065	0.075	0.083	0.098	0.106	0.12	0.127	0.139	0.152	0.167	0.184	0.202
SUSPEND	0.031	0.037	0.044	0.054	0.064	0.078	0.091	0.11	0.122	0.16	0.165	0.174	0.211	0.231	0.248	0.282	0.307

	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
SPIN_BUSY	0.313	0.70	3.71	10.8	15.2	16.5	20.1	21.1	20.7	20.1	24.0	23.2	23.5	23.5	24.1	24.5	35.6	29.8
SPIN_PAUSE	0.248	0.289	0.341	0.336	0.354	0.347	0.429	0.425	0.452	0.509	0.543	0.609	0.605	0.642	0.731	0.762	0.957	0.856
SPIN_YIELD	0.217	0.235	0.254	0.302	0.303	0.389	0.417	0.374	0.422	0.441	0.49	0.517	0.575	0.593	0.586	0.613	0.68	0.788
SUSPEND	0.346	0.381	0.451	0.487	0.524	0.567	0.626	0.697	0.749	0.774	0.842	0.95	1.002	1.035	1.096	1.178	1.219	1.374

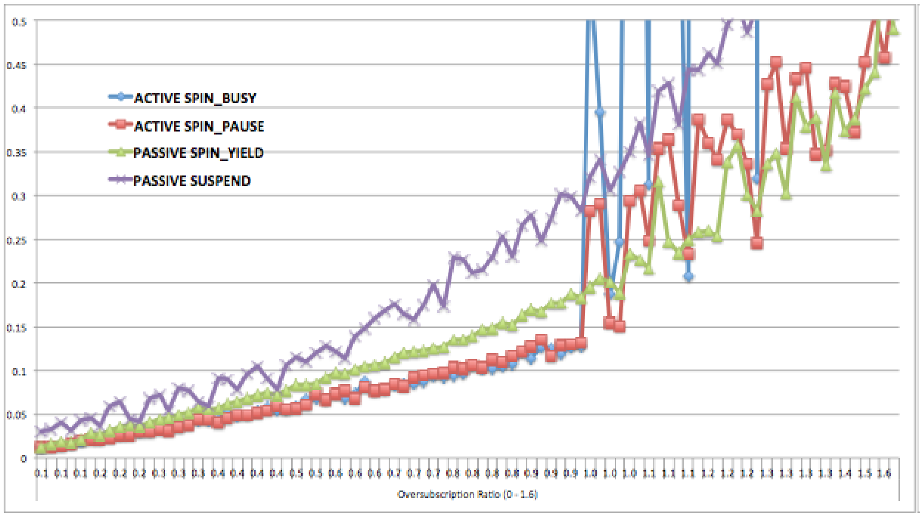
8 PThreads



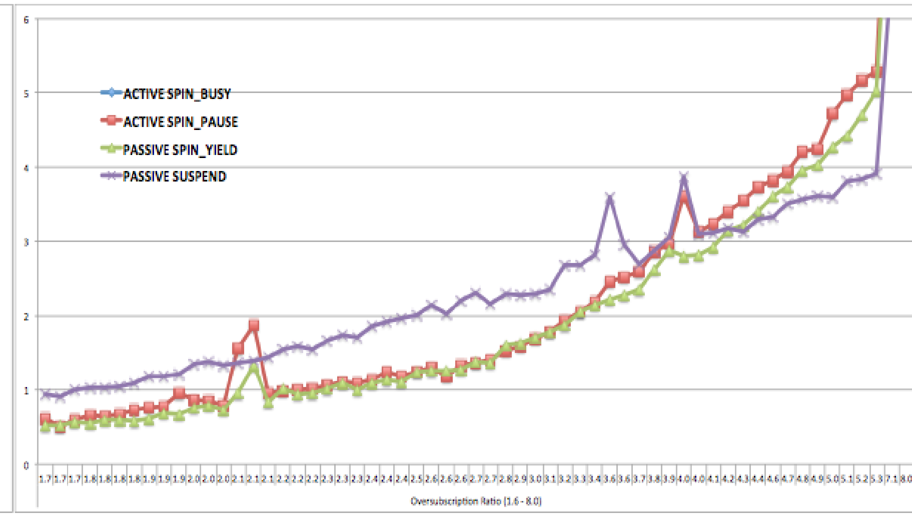
	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
SPIN_BUSY	0.03	0.04	0.06	0.07	0.09	0.1	0.13	0.25	0.21	0.32	3.89	4.28	17.6	20.3	20.9	22.9	29.1	25.4	28.7	32.4	42.2	51.6	47.8	58.3
SPIN_PAUSE	0.03	0.04	0.06	0.08	0.09	0.1	0.12	0.15	0.23	0.25	0.35	0.37	0.46	0.5	0.67	0.78	0.79	0.96	1.01	1.1	1.18	1.19	1.4	1.52
SPIN_YIELD	0.04	0.06	0.07	0.1	0.12	0.15	0.18	0.19	0.25	0.28	0.34	0.39	0.59	0.53	0.59	0.68	0.73	0.84	0.95	1.01	1.11	1.25	1.36	1.61
SUSPEND	0.04	0.06	0.08	0.11	0.16	0.22	0.27	0.33	0.44	0.51	0.63	0.72	0.79	0.91	1.05	1.19	1.33	1.44	1.55	1.71	1.96	2.02	2.15	2.3

	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	64	72
SPIN_BUSY	61.4	64.4	71.2	72.1	77.1	83.8	85.8	88.3	89.6	86.6	90.8	84.7	91.7	91.6	90.4	87.2	85.2	88.8	86.4	81.6	86.7	81.0	84.0	124.	141.
SPIN_PAUSE	1.59	1.69	1.78	1.93	2.05	2.19	2.51	2.59	2.87	2.97	3.13	3.24	3.41	3.55	3.73	3.82	3.95	4.22	4.24	4.72	4.98	5.17	5.29	9.31	11.9
SPIN_YIELD	1.63	1.71	1.78	1.88	2.05	2.14	2.27	2.35	2.62	2.88	2.82	2.93	3.14	3.22	3.4	3.62	3.74	3.96	4.04	4.27	4.42	4.71	5.03	8.49	10.9
SUSPEND	2.28	2.3	2.35	2.68	2.68	2.82	2.95	2.69	2.88	3.06	3.11	3.11	3.18	3.14	3.29	3.33	3.51	3.56	3.62	3.6	3.8	3.84	3.91	6.59	8.44

Oversubscription Ratio= threads_requested/threads_supported



ratio 0-1.6



ratio 1.6-8.0

- No or light oversubscription (0< ratio <1.3):
  - Good: ACTIVE SPIN_BUSY or SPIN_PAUSE policy are good choices
- Mild oversubscription (1.3 <ratio< 4)
  - Good: ACTIVE SPIN_PAUSE and PASSIVE SPIN_YIELD
- Heavily oversubscribed system (4< ratio)
  - Good: PASSIVE_SUSPEND

Related work

- OpenMP compilers: Tian et al IPDPS’03. explored interoperability between OpenMP threads and system threads.
- Runtime systems: starPU[IJHPCA’14] hypervisor for confined resources, MPC framework[EuroPar’08] process virtualization, common resource management by Callisto [EuroSys’14] and Lithe[HotPar’09]
- MPI interoperability: MPI endpoints[DinanEuroMPI’13] process-task relations, MPI calls as tasks in workstealing runtime[ChatterjeeIPDPS’13]
- Interoperability among distributed HPC programming models: Scientific Interface Definition Language and Babel Intermediate Object Representation [EpperlyLLNL’11]

Conclusion

- OpenMP has the interoperability challenges:
  - within itself and with other programming APIs
- We proposed solutions, focusing on the resource oversubscription issue
  - fine-grain specifying and controlling wait polices
  - terminating/suspending runtime
  - integrating of user threads
- Initial implementation and evalution
  - Two OpenMP runtime libraries: Intel OpenMP and GNU OpenMP
  - Different oversubscription ratios : different best wait policies

Two cases

```
omp_set_wait_policy(ACTIVE);
#pragma omp parallel num_threads(16)
{
  omp_set_wait_policy(PASSIVE);
  #pragma omp parallel num_threads(4)
  {
 /* The 4*16 threads are PASSIVE */
  }
    /* The 3*16 threads are PASSIVE, other 1*16 are ACTIVE master threads  */
}
/* ACTIVE for threads in the outer region */
```

```
omp_set_wait_policy(PASSIVE)
  #pragma omp parallel num_threads(16)
  {
  }

  #pragma omp parallel num_threads(12)
  {
  }

  // 12 threads are active here
  #pragma omp parallel num_threads(12)
  {
  }
```

Use Cases of Interoperating OpenMP with Others

- With explicit user threads
  - OpenMP applications mixed with Pthreads or Win 32 APIs.
- With inter-node programming models
  - The thread(s) inside of e.g. an MPI library used by an OpenMP application matches with the Concurrent motif.
- With parallel libraries
  - OpenMP threads calling application or library functions that use another threading model, e.g. OpenMP, TBB, Cilkplus, etc.
  - or the other way around.