Implementing Flexible Threading Support in Open MPI

PRESENTED BY

Jan Ciesko, Sandia National Laboratories, Albuquerque, NM

Coauthors: Noah Evans, Stephen Olivier, Howard Pritchard, Shintaro Iwasaki, Ken Raffanetti and Pavan Balaji

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Agenda

1: Motivation

2: Implementation

3: Evaluation

4: What's next?

5: Conclusion

1. Motivation: Threading Implementations Differ

Threading libraries differ in performance.

- Ptheads vs ULTs

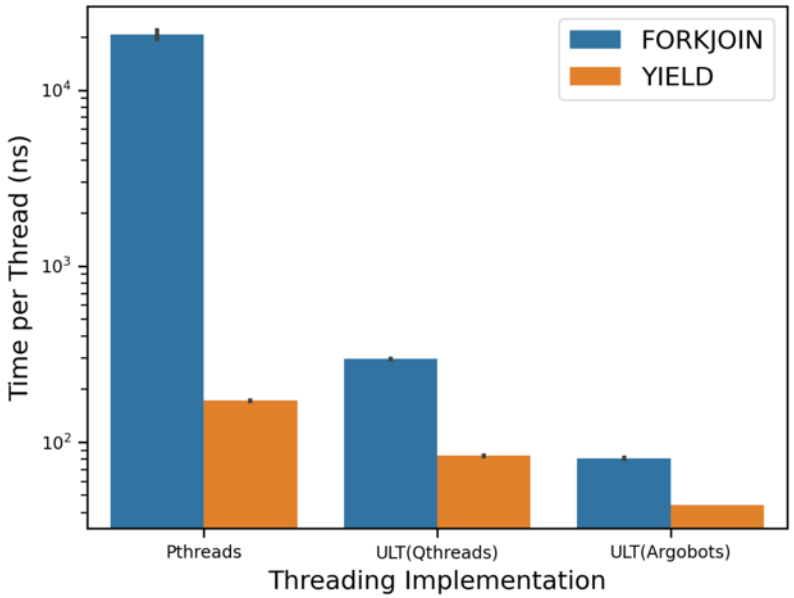
MPI implementations depend on the underlying threading implementation

ThreadOps-bench

```
1 ...
2 static void *yield(void *arg) {
3   size_t num_yields = (size_t)((intptr_t)arg);
4   for (int i = 0; i < num_yields; i++) {
5     pthread_yield();
6   }
7 }
8
9 static void kernel(int num_threads, int num_yields)
10 {
11   for (int i = 0; i < num_threads; i++) {
12     pthread_create(&g_threads[i], NULL, yield_f,
13                    (void *)((intptr_t)num_yields));
14   }
15   for (int i = 0; i < num_threads; i++) {
16     pthread_join(g_threads[i], NULL);
17   }
18 }
```

1st run:
num_yields = 0;
num_theads = 16;

2nd run:
num_yields=4000;
num_threads=16;



Blake.sandia.gov
Intel(R) Xeon(R) Platinum 8160 CPU @ 2.10GHz
16 threads, normalized to 1 thread
pthread_yield()…relinquish CPU

https://github.com/jancieskо/ThreadOpsBench.git

2. Implementation: Objective

Add generic threading support to Open MPI

- Add a new MCA base framework
- Add particular MCA components for Pthreads, Qthreads and Argobots
- Define a generic interface for threading
- Break MCA and add a configure-time option ☺
- Remove calls to Pthreads and use generic interface throughout the Open MPI code base
- Adjust configuration and build process
- Add configuration option --with-threads=<threading model>

Take-aways: What is MCA, how threading fits into into MCA and how you can select a threading implementation.

2. Implementation: MCA in Open MPI

Modular Component Architecture (MCA)

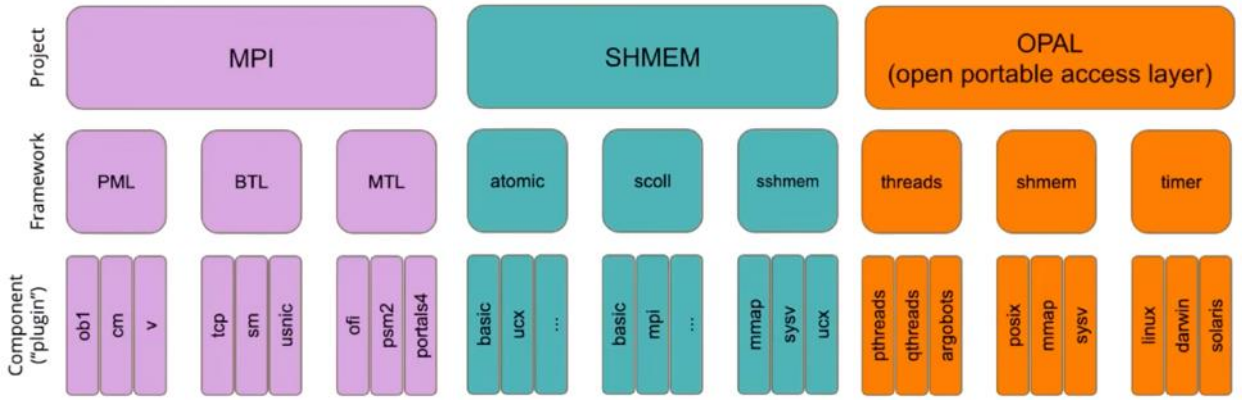
- Open MPI organized into projects, frameworks and components
- Components are loaded at runtime.

Examples:

```
1 mpirun --mca pml ob1 --mca btl
2 mpirun --mca pml cm --mca mtl
3 mpirun --mca pml ucx
```

Note: MCAs are specified by providing a key-value pair and are loaded at runtime (dlopen).

Open MPI software package



* From EasyBuild Tech Talk, Jeff Squyres and Ralph Castain

- ob1: Multi-device, multi-rail engine
  - Uses BTL components (byte transfer layer)
- cm: Engine for matching network layers
  - Uses MTL components (matching transport layer)
- ucx: Uses the UCX communication library (Unified Communications X)

2. Implementation: Add new MCA component

MCA defines a set of useful APIs → Threads MCA implements those as

ompi / opal / mca / mca.h

```
1 typedef int (*mca_open_component_fn_t)(void);
2 typedef int (*mca_close_component_fn_t)(void);
3 typedef int (*mca_query_component_fn_t)(mca_base_module_t
4  **module, int *priority);
5 typedef int (*mca_register_component_params_t)(void);
6
7 struct mca_component_t {
8   int mca_major_version;
9   /**< Major number of the MCA. */
10  int mca_minor_version;
11  /**< Minor number of the MCA. */
12  int mca_release_version;
13  /**< Release number of the MCA. */
14 ...
15  mca_open_component_fn_t mca_open_component;
16  /**< Method for opening this component. */
17  mca_close_component_fn_t mca_close_component;
18  /**< Method for closing this component. */
19  mca_query_component_fn_t mca_query_component;
20  /**< Method for querying this component. */
21  mca_register_component_params_fn_t mca_register_component_params;
22 }
```

ompi / opal / mca / threads / thread.h

```
1 struct opal_threads_base_component_t {
2     /** MCA base component */
3     mca_component_t mca_thread_component;
4     /** MCA base data */
5     mca_component_data_t threadsc_data;
6 };
```

Let's create a Pthread MCA component

ompi / opal / mca / threads / pthreads / threads_pthreads_component.c

```
1 int opal_threads_pthreads_open(void){
2     return OPAL_SUCCESS;
3 }
4 const opal_threads_base_component_t mca_threads_pthreads_component = {
5     .mca_thread_component  = {
6         OPAL_THREADS_BASE_VERSION_1_0_0,
7         /* Component name and version */
8         .mca_component_name = "pthreads",
9         MCA_BASE_MAKE_VERSION(component, OPAL_MAJOR_VERSION, OPAL_MINOR_VERSION,
10                               OPAL_RELEASE_VERSION),
11
12        .mca_open_component = opal_threads_pthreads_open,
```

Note: ompi_info lists the component. Can we dlopen this? Yes, but…

2. Implementation: Define Generic APIs

Generic Threading API implements

- TLS
- Synchronization
- Management
- Mutex
- Atomics

Definitions

Declarations

master ompi / opal / mca / threads /

..	
argobots	mca/threads: set THREAD_* flags in the component's root configure.m4
base	opal/thread: New TSD API
pthreads	mca/threads: set THREAD_* flags in the component's root configure.m4
qthreads	mca/threads: set THREAD_* flags in the component's root configure.m4
Makefile.am	Add threads framework
README.md	mca/threads: remove libevent hack
condition.h	Add threads framework
configure.m4	mca/threads: set THREAD_* flags in the component's root configure.m4
mutex.h	Add threads framework
thread.h	Add threads framework
thread_usage.h	Add threads framework
threads.h	mca/threads: remove libevent hack
tsd.h	Fix renamed interface functions for argo, q, and pthreads
wait_sync.h	ompi/request: move REQUEST constants from mca/threads to ompi/request

2. Implementation: Declarations of Members

Generic Threading API implements

- Management
- Synchronization
- TLS
- Mutexes
- Atomics

Note: Open MPI currently implements threading in a hybrid approach. Only management functions are implemented in a shared library.

Management:

ompi / opal / mca / threads / **threads.h**

```
1 ...
2 extern int  opal_thread_start(opal_thread_t *);
3 extern int  opal_thread_join(opal_thread_t *, void **thread_return);
4 extern bool opal_thread_self_compare(opal_thread_t *);
5 extern opal_thread_t *opal_thread_get_self(void);
6 extern void opal_thread_kill(opal_thread_t *, int sig);
7 extern void opal_thread_set_main(void);
```

Mutexes (**hot path**):

ompi / opal / mca / threads / **threads.h**

```
1 ...
2 static inline int opal_mutex_trylock(opal_mutex_t *mutex);
3 static inline void opal_mutex_lock(opal_mutex_t *mutex);
4 static inline void opal_mutex_unlock(opal_mutex_t *mutex);
5 static inline int opal_mutex_atomic_trylock(opal_mutex_t *mutex);
6 static inline void opal_mutex_atomic_lock(opal_mutex_t *mutex);
7 static inline void opal_mutex_atomic_unlock(opal_mutex_t *mutex);
```

Note: These are declarations that need the definition at compile time!

2. Implementation: Implementations of Static Members

Mutexes implementations in hot path

- Pthreads
- Qthreads
- Argobots

Argobots:

```
1 ...
2 static inline void opal_mutex_lock(opal_mutex_t *m)
3 {
4     if (OPAL_ABT_MUTEX_NULL == m->m_lock_argobots) {
5         opal_mutex_create(m);
6     }
7    ABT_mutex_lock(m->m_lock_argobots);
8 }
9
10 static inline void opal_mutex_unlock(opal_mutex_t *m)
11 {
12     if (OPAL_ABT_MUTEX_NULL == m->m_lock_argobots) {
13         opal_mutex_create(m);
14     }
15     ABT_mutex_unlock(m->m_lock_argobots);
16     /* For fairness of locking. */
17     ABT_thread_yield();
18 }
```

Pthreads:

```
1 ...
2 static inline void opal_mutex_lock(opal_mutex_t *m)
3 {
4   pthread_mutex_lock(&m->m_lock_pthread);
5 }
6
7 static inline void opal_mutex_unlock(opal_mutex_t *m)
8 {
9   pthread_mutex_unlock(&m->m_lock_pthread);
10 }
```

Qthreads:

```
1 ...
2 static inline void opal_mutex_lock(opal_mutex_t *m)
3 {
4     opal_threads_ensure_init_qthreads();
5     int ret = opal_atomic_trylock(&m->m_lock);
6     while (0 != ret) {
7         qthread_yield();
8         ret = opal_atomic_trylock(&m->m_lock);
9     }
10 }
11
12 static inline void opal_mutex_unlock(opal_mutex_t *m)
13 {
14     opal_threads_ensure_init_qthreads();
15     opal_atomic_unlock(&m->m_lock);
16     /* For fairness of locking. */
17     qthread_yield();
18 }
```

2. Implementation: Only a handful of APIs follow MCA

Current state

- Only a subset of the API is implemented in the generic API and would support exchangeable threading at runtime
- Hot-path functionality is statically in-lined, thus must be selected at configure time
- Hybrid approach to minimize impact

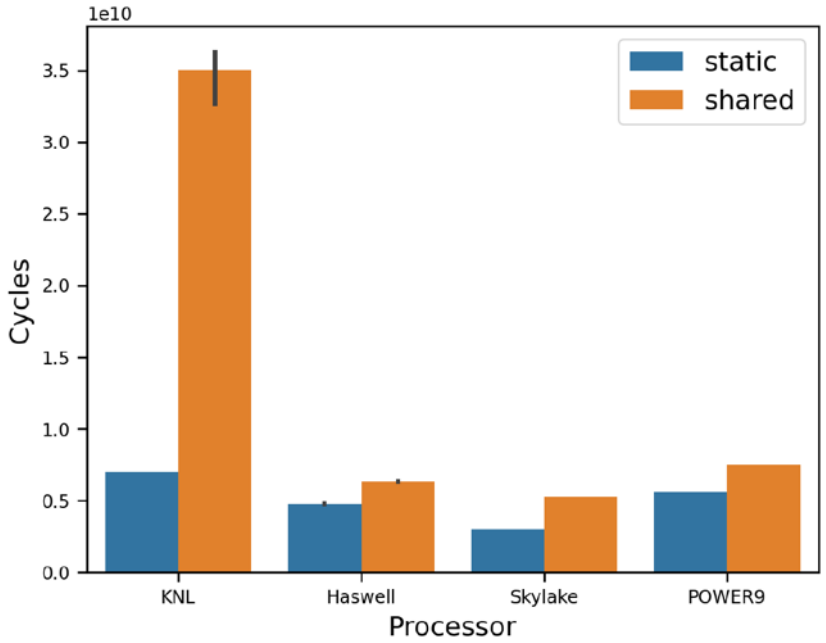
Note: Even though we can load the MCA threading library and runtime, we must select the threading library at compile time using *--with-threads=<threading library>*.

OpenMPI OPAL MCA				
Abstraction	**Interface**	**Threading**		
		Abstraction	**Interface**	**S**
Component handling	open	Condition	*opal_condition_wait*	
	close		*opal_condition_timedwait*	
	register		*opal_condition_signal*	
	query		*opal_condition_broadcast*	
		Mutex	*opal_mutex_trylock*	
			opal_mutex_lock	
			opal_mutex_unlock	
			opal_mutex_atomic_trylock	
			opal_mutex_atomic_unlock	
			opal_thread_lock	
			opal_thread_trylock	
			opal_thread_unlock	
			opal_thread_scoped_lock	
		Thread Usage	*opal_using_threads*	
			opal_set_using_threads	
			opal_atomic_NAME_OP_TYPE	
			opal_thread_fetch_NAME_OP_TYPE	
			opal_atomic_compare_exchange_strong_SUFFIX_ADDRTYPE_TYPE	
			opal_thread_swap_SUFFIX_ADDRTYPE_TYPE	
		Threads	*opal_acquire_thread*	
			opal_release_thread	
			opal_wakeup_thread	
			opal_post_object	
			opal_acquire_object	
			opal_thread_start	x
			opal_thread_join	x
			opal_thread_self_compare	x
			opal_thread_get_self	x
			opal_thread_kill	x
			opal_thread_set_main	x
		Thread-specific Datastore	opal_tsd_key_create	x
			opal_tsd_key_delete	x
			opal_tsd_setspecific	x
			opal_tsd_getspecific	x
			opal_tsd_keys_destruct	x
		Wait and Syncronize	*wait_sync_update*	

3. Evaluation: Function Call Overhead

FNbench

- Approximately 20-30% increase in cycles
- KNL misses 49% branch predictions
- PLT* look-up for PIC adds 30% of instructions per iteration



https://github.com/npe9/fnbench

*Procedure Linkage Table
**IBM Power9 RDTSC counter data adjusted to reflect polling frequency

3. Evaluation: Function Call Overhead

FNbench

As Static library:

```
1 void __attribute__ ((noinline)) noop() {
2     asm volatile("":::);
3 }
4
5 int main(void)
6 {
7   unsigned long long start, finish;
8   int i = 0;
9   start = rdtsc();
10  for(i = 0; i < ITERS; i++)
11    noop();
12  finish = rdtsc();
13  return 1;
14 }
```

./objdump bin.static

```
000010000790 <noop>:
10000790:    20 00 80 4e      blr
    ...

10000534:    40 8a bd 3b      addi    r29,r29,-30144
10000538:    78 f3 9e 7c      or      r30,r4,r30
1000053c:    00 00 e0 3b      li      r31,0
10000540:    51 02 00 48      bl      10000790 <noop>
10000544:    01 00 3f 39      addi    r9,r31,1
10000548:    00 00 fd cb      lfd     f31,0(r29)
```

As Shared library:

```
1 extern void noop( void );
2
3 int main(void)
4 {
5   unsigned long long start, finish;
6   int i = 0;
7   start = rdtsc();
8   for(i = 0; i < ITERS; i++)
9     noop();
10  finish = rdtsc();
11  return 1;
12 }
```

./objdump bin.shared

```
0000100005c0 <00000039.plt_call.noop>:
100005c0:    18 00 41 f8      std     r2,24(r1)
100005c4:    10 81 82 e9      ld      r12,-32496(r2)
100005c8:    a6 03 89 7d      mtctr   r12
100005cc:    20 04 80 4e      bctr

1000064c:    08 8b bd 3b      addi    r29,r29,-29944
10000650:    78 f3 9e 7c      or      r30,r4,r30
10000654:    00 00 e0 3b      li      r31,0
10000658:    00 00 49 f9      std     r10,0(r9)
1000065c:    00 00 42 60      ori     r2,r2,0
10000660:    61 ff ff 4b      bl      100005c0 <00000039.plt_call.noop>
10000664:    18 00 41 e8      ld      r2,24(r1)
10000668:    01 00 3f 39      addi    r9,r31,1
1000066c:    00 00 9d c9      lfd     f12,0(r29)
```

gcc -Wall -O3 src/testfn.c -c -o src/testfn.c.so
gcc -fPIC -shared src/testfn.c.so -o libtestfn.so
gcc -Wall -O3 -L./ ./src/fnbenchso.c -o main.exe.dynamic -ltestfn

https://github.com/npe9/fnbench

3. Evaluation: Performance Overhead

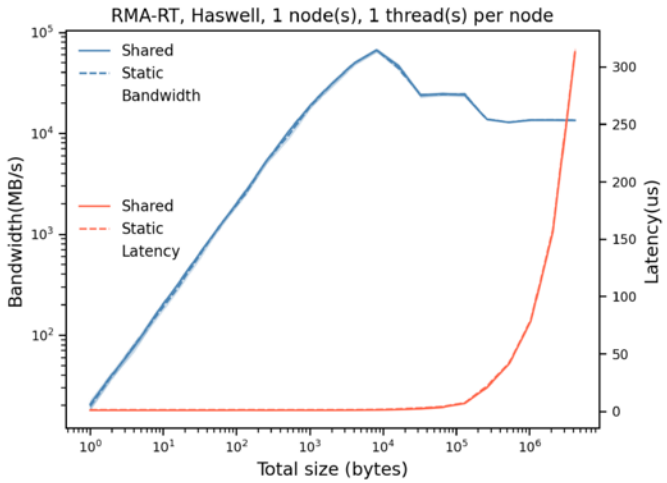
RMA-MT

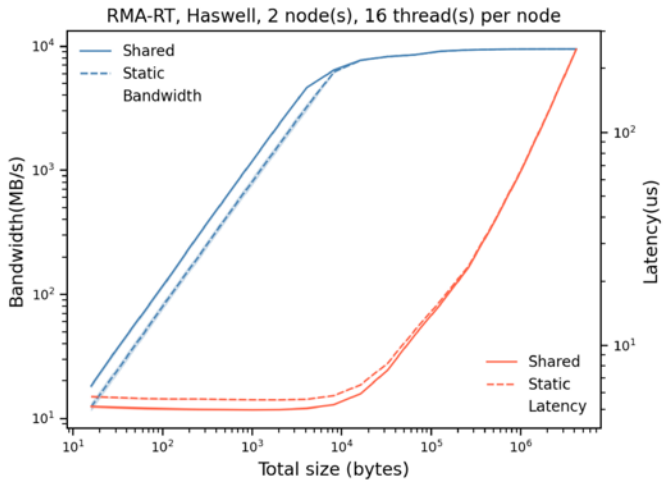
- Experimental implementation
  - Shared threading API
- Intel Haswell and Skylake
- Static versus shared library
- All function declared as *e xte rn*
- Using Pthreads

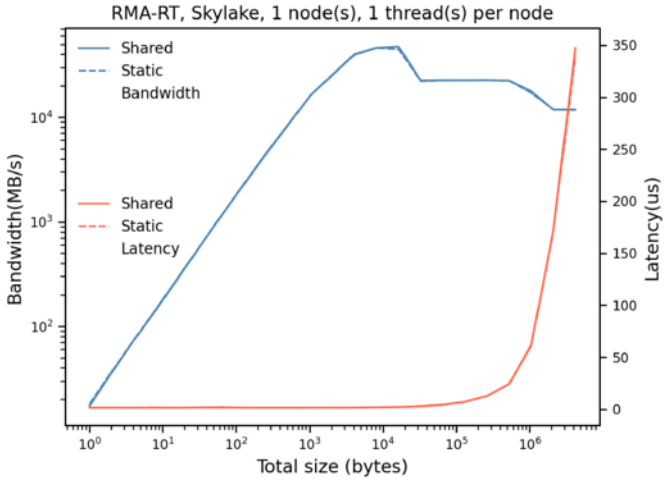
mpirun --np 2 --map-by ppr:<1,2>:node --bind-to socket **rmamt**_<*bw*,lat> -x -t <num_threads> -o put -s fence

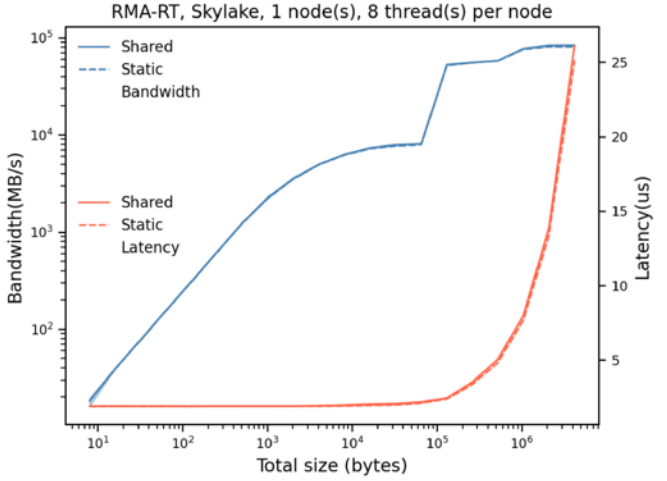
Intel Haswell
Bowman.sandia.gov
Intel Xeon E5-2698 (Haswell), 2.6GHz, 2x16 cores/node
	`Cisco usNIC: no`
	`Cray uGNI (Gemini/Aries): no`
	`Intel Omnipath (PSM2): yes`
	`Open UCX: yes`
	`OpenFabrics OFI Libfabric: yes`

Intel Skylake
Blake.sandia.gov
Intel Xeon Platinum (Skylake) 8160, 2.1 GHz, 2 x 24 cores / node
	`Cisco usNIC: no`
	`Cray uGNI (Gemini/Aries): no`
	`Intel Omnipath (PSM2): yes`
	`Open UCX: yes`
	`OpenFabrics OFI Libfabric: yes`









3. Evaluation: Performance Overhead

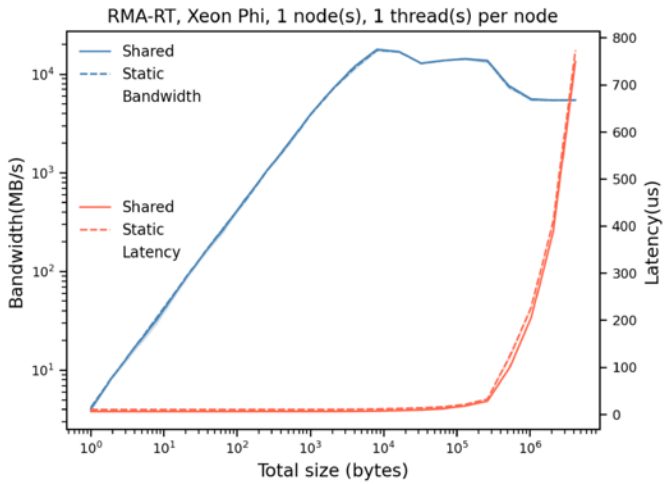
RMA-MT

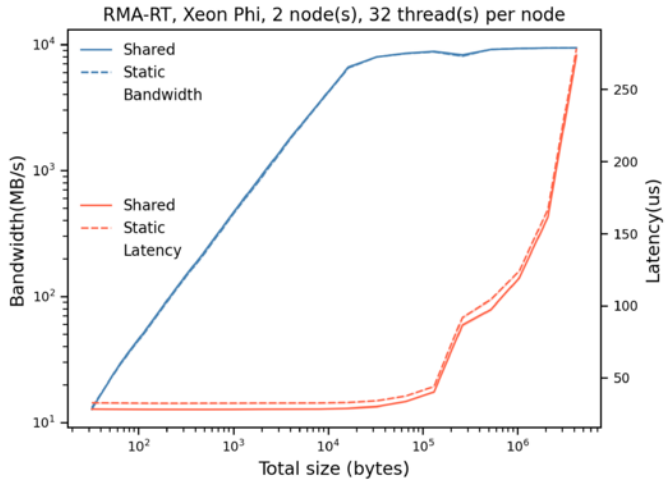
- Experimental implementation
  - Shared threading API
- Intel Xeon Phi and IBM Power9
- Static versus shared library
- All function declared as *e xte rn*
- Using Pthreads

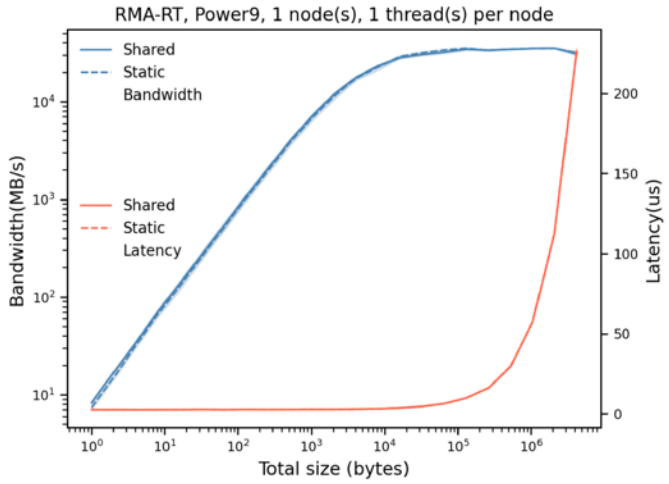
mpirun --np 2 --map-by ppr:<1,2>:node --bind-to socket **rmamt**_<*bw*,lat> -x -t <num_threads> -o put -s fence

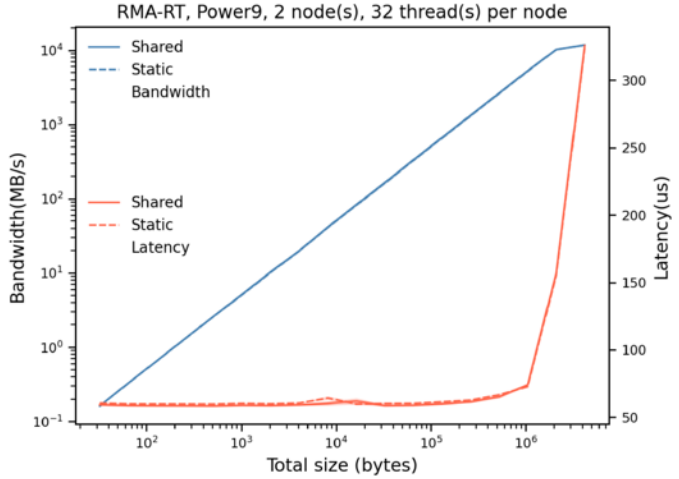
Intel Xeon Phi	`Cisco usNIC: no`
Voltrino.sandia.gov	`Cray uGNI (Gemini/Aries): yes`
Intel Xeon Phi	`Intel Omnipath (PSM2): no`
7250, 68 cores	`Open UCX: no`
	`OpenFabrics OFI Libfabric: no`

IBM Power9	`Cisco usNIC: yes`
Lassen.llnl.gov	`Cray uGNI (Gemini/Aries): no`
IBM Power9, 2.3 GHz, 2 x 22 cored / node	`Intel Omnipath (PSM2): no`
	`Intel TrueScale (PSM): no`
	`Mellanox MXM: no`
	`Open UCX: yes`









4. What's next?

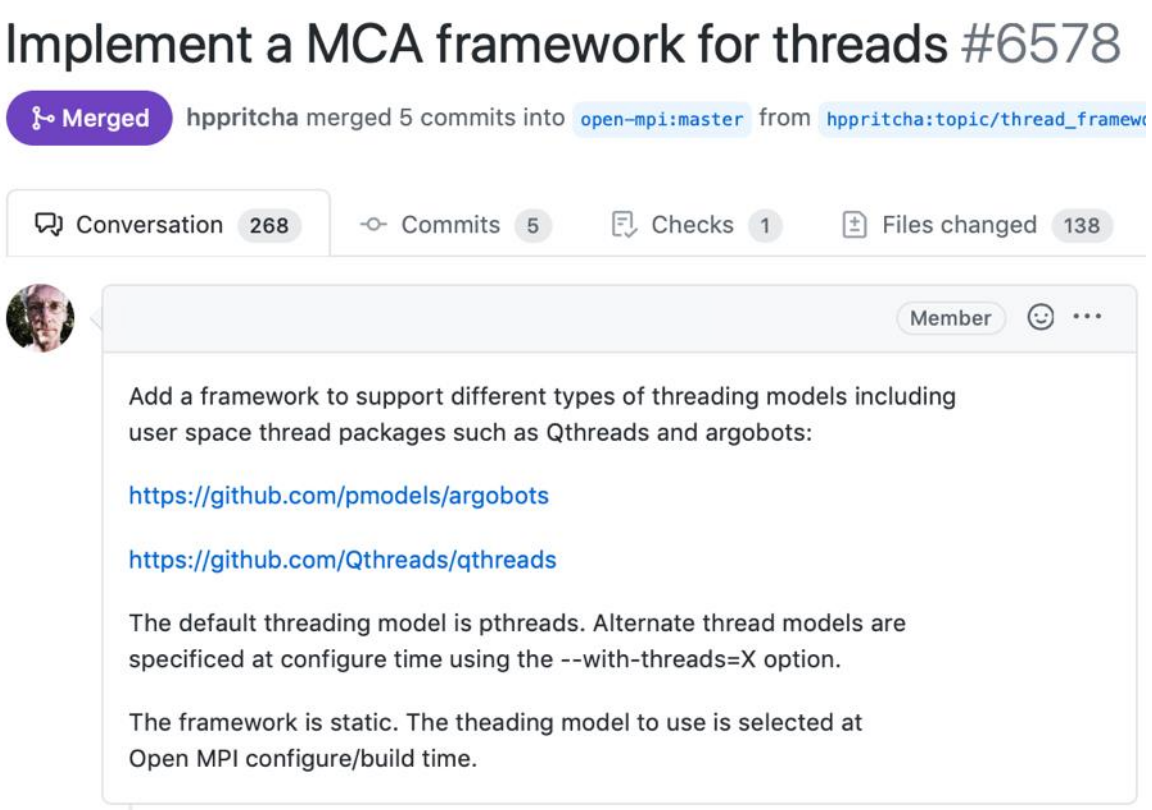
Show and quantify benefits of ULTs in Open MPI and MPI+X

Optimize the use of ULT in the Open MPI threading API

Investigate usefulness and implications of an "at run-time" selection of threading library (no static linking or static variables)

Resolve correctness issue in regard to ULT-to-Pthread mapping (deadlock)

Libevent support for ULTs



https://github.com/open-mpi/ompi/pull/6578

5. Summary

Threading support in Open MPI is a hybrid approach.

Management functionality implemented following MCA .

Hot-path functionality statically defined and in-lined.

Use --with-threads=<threading model>.

Evaluation shows no significant performance differences.

Base work for a lot of interesting future work.

Find us on Slack!

slack

https://qthreads.slack.com/

GitHub

https://github.com/open-mpi

https://github.com/qthreads

https://github.com/argobots