Implementing Flexible Threading Support in Open MPI

Jan Ciesko, Sandia National Laboratories, Albuquerque, NM

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

1: Motivation
2: Implementation
3: Evaluation
4: What’s next?
5: Conclusion
1. Motivation: Threading Implementations Differ

Threading libraries differ in performance.
- Pthreads vs ULTs

MPI implementations depend on the underlying threading implementation.

ThreadOps-bench

```c
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_threads = 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/janciesko/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 oob --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).
2. Implementation: Add new MCA component

MCA defines a set of useful APIs

```c
#define opal / 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_fn_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 }
```

Threads MCA implements those as

```c
#define opal / 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

```c
#define opal / 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,
13 }
```

Note: opal_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

- `argobots`
- `base`
- `pthreads`
- `qthreads`
- `Makefile.am`
- `README.md`
- `condition.h`
- `configure.m4`
- `mutex.h`
- `thread.h`
- `thread_usage.h`
- `threads.h`
- `tsd.h`
- `wait_sync.h`

Declarations

- `mca/threads: set THREAD_* flags in the component's root configure.m4`
- `opal/thread: New TSD API`
- `mca/threads: set THREAD_* flags in the component's root configure.m4`
- `Add threads framework`
- `mca/threads: remove libevent hack`
- `Add threads framework`
- `mca/threads: set THREAD_* flags in the component's root configure.m4`
- `Add threads framework`
- `Add threads framework`
- `Add threads framework`
- `mca/threads: remove libevent hack`
- `Fix renamed interface functions for argo, q, and pthreads`
- `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:**

```c
extern int opal_thread_start(opal_thread_t *);
extern int opal_thread_join(opal_thread_t *, void **thread_return);
extern bool opal_thread_self_compare(opal_thread_t *);
extern opal_thread_t *opal_thread_get_self(void);
extern void opal_thread_kill(opal_thread_t *, int sig);
extern void opal_thread_set_main(void);
```

**Mutexes (hot path):**

```c
static inline int opal_mutex.trylock(opal_mutex_t *mutex);
static inline void opal_mutex.lock(opal_mutex_t *mutex);
static inline void opal_mutex.unlock(opal_mutex_t *mutex);
static inline int opal_mutex.atomic.trylock(opal_mutex_t *mutex);
static inline void opal_mutex.atomic.lock(opal_mutex_t *mutex);
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:

```c
1... 2 static inline void opal_mutex_lock(opal_mutex_t *m)
3 { 4   if (OPAL_AB_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_AB_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:

```c
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 }
```
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-<threading_library>`. 
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

*Procedure Linkage Table
**IBM Power9 RDTSC counter data adjusted to reflect polling frequency
3. Evaluation: Function Call Overhead

**FNbench**

**As Static library:**

```c
void __attribute__((noinline)) noop() {
  asm volatile("fretstsc");
}

int main(void)
{
  7   unsigned long long start, finish;
  8   int i = 0;
  9   start = rdtsc();
 10   for(i = 0; i < ITERS; i++)
11     noopt();
12   finish = rdtsc();
13   return 1;
14 }
```

```
./objdump bin.static
```

```
00000100000790 <noop>:
1000790:  20 00 80 04 e1 blr
...
```

```
100000534:  40 8a bd 3b adi r29,r29,-30144
100000538:  78 f3 9e 7c or r30,r4,r30
10000053c:  00 00 e6 3b li r31,0
100000540:  51 02 00 48 bl 100000790 <noop>
100000544:  01 00 3f 39 addi r9,r31,1
100000548:  00 00 fd cb lfd f31,0(r29)
```

**As Shared library:**

```c
extern void noop(void);
```

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

```
./objdump bin.shared
```

```
00000100005c0 <00000039.plt_call.noop>:
100005c0:  13 00 41 f8 std r2,24(r1)
100005c4:  10 81 82 e9 ld r12,-32496(r2)
100005c8:  a6 03 89 7d mtcrr r12
100005cc:  20 04 80 04 bctr  
```

```
10000064c:  08 8b bd 3b adi r29,r29,-29944
100000650:  78 f3 9e 7c or r30,r4,r30
100000654:  00 00 e9 3b li r31,0
100000658:  00 00 49 f9 std r10,0(r9)
10000065c:  00 00 42 60 or r2,r2,0
100000660:  61 ff ff 4b bl 1000005c0 <00000039.plt_call.noop>
100000664:  18 00 41 e8 ld r2,24(r1)
100000668:  01 00 3f 39 addi r9,r31,1
10000066c:  00 00 9d c9 lfd f12,0(r29)
```

```bash
gcc -Wall -O3 src/testfn.c -c -o src/testfn.c.s
gcc -fPIC -shared src/testfn.c.s -o libtestfn.so
gcc -Wall -O3 -L ./src/fnbenchso.c -o main.exe -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 `extern`
- Using Pthreads

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

<table>
<thead>
<tr>
<th>Intel Haswell</th>
<th>Bowman.sandia.gov</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Xeon E5-2690</td>
<td>2.6 GHz</td>
</tr>
<tr>
<td>2x16 cores/node</td>
<td></td>
</tr>
<tr>
<td><strong>Cisco uN NIC</strong>: no</td>
<td><strong>Cray uNQI (Gemini/Aries)</strong>: no</td>
</tr>
<tr>
<td><strong>Intel OmniPath (PSM)</strong>: yes</td>
<td><strong>Intel OmniPath (PSM)</strong>: yes</td>
</tr>
<tr>
<td><strong>Open UCI</strong>: yes</td>
<td><strong>OpenUCCI</strong>: yes</td>
</tr>
<tr>
<td><strong>OpenFabric OFI LibFabric</strong>: yes</td>
<td><strong>OpenFabric OFI LibFabric</strong>: yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Intel Skylake</th>
<th>Blake.sandia.gov</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Xeon E7-8894 V4</td>
<td>2.1 GHz</td>
</tr>
<tr>
<td>Platinum (Skylake)</td>
<td>2x16 cores/node</td>
</tr>
<tr>
<td><strong>Cisco uN NIC</strong>: no</td>
<td><strong>Cray uNQI (Gemini/Aries)</strong>: no</td>
</tr>
<tr>
<td><strong>Intel OmniPath (PSM)</strong>: yes</td>
<td><strong>Intel OmniPath (PSM)</strong>: yes</td>
</tr>
<tr>
<td><strong>Open UCI</strong>: yes</td>
<td><strong>OpenUCCI</strong>: yes</td>
</tr>
<tr>
<td><strong>OpenFabric OFI LibFabric</strong>: yes</td>
<td><strong>OpenFabric OFI LibFabric</strong>: yes</td>
</tr>
</tbody>
</table>
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 `extern`
- Using Pthreads

Example command:
```
mpirun --np 2 --map-by ppr:<1,2>:node --bind-to socket rmamt_bwlat -x -t <num_threads> -o put -s fence
```

### Intel Xeon Phi

<table>
<thead>
<tr>
<th>Voltrino.sandia.gov</th>
<th>Cray uQMI (Gemini/Aries): yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Xeon Phi 7230, 68 cores</td>
<td>Intel OmniPath (PSM): no</td>
</tr>
<tr>
<td></td>
<td>Open UCI: no</td>
</tr>
<tr>
<td></td>
<td>OpenFabrics OFI Libfabric: no</td>
</tr>
</tbody>
</table>

### IBM Power9

<table>
<thead>
<tr>
<th>Lassen.bnl.gov</th>
<th>Cray uQMI (Gemini/Aries): yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM Power9, 2.3 GHz, 2 x 22 cores</td>
<td>Intel OmniPath (PSM): no</td>
</tr>
<tr>
<td></td>
<td>Intel TrueScale (PSM): no</td>
</tr>
<tr>
<td></td>
<td>Mellanox MOD: no</td>
</tr>
<tr>
<td></td>
<td>Open UCI: yes</td>
</tr>
</tbody>
</table>

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

---

Implement a MCA framework for threads #6578

Add a framework to support different types of threading models including user space thread packages such as Qthreads and argobots:

https://github.com/pmodels/argobots

https://github.com/Qthreads/qthreads

The default threading model is pthreads. Alternate thread models are specified at configure time using the --with-threads=X option.

The framework is static. The threading model to use is selected at Open MPI configure/build time.

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