



# Treating threads as MPI processes thru registration/deregistration

Alexander Supalov  
Intel Corporation

June 5, 2009



# Contents

- Rationale
- Proposal
- Bindings
- Example
  - Sample code
  - Timeline
  - Notes
- Extensions

# Rationale

- Increasing prevalence of multi- and manycore processors calls for extended MPI facilities for dealing with threads as first class MPI entities
- So far the MPI standard treats threads as second-class entities that cannot be directly addressed and whose presence is merely tolerated
- This leads to issues like the Probe/Recv consistency issue described in <https://svn.mpi-forum.org/trac/mpi-forum-web/ticket/38> and the partial solution proposed therein
- This proposal seeks to introduce a powerful and convenient way of direct addressing of the threads as MPI processes, thus resolving all related issues

# Proposal

- A new collective routine **MPI\_Comm\_thread\_register()** is introduced to create a communicator in which existing threads become MPI “processes” with unique ranks
  - The way in which the existing threads were created or are going to be terminated is out of the scope of this proposal. See Posix threads, OpenMP, etc.
- The existing routines **MPI\_Comm\_free()** is extended to terminate the resulting communicators
- All power of MPI is retained. For example:
  - All communicator and group manipulation routines can work on the resulting communicators recursively, combining and rearranging processes as needed to the user
  - All communication routines (pt2pt, collectives, 1-sided, file I/O) can work on the new communicators, and may optionally take advantage of the fact that the data should not cross the process boundary
- Prerequisite: MPI\_THREAD\_MULTIPLE thread support level

# MPI\_Comm\_thread\_register

## Language independent binding

```
MPI_Comm_thread_register(comm,  
    local_thread_index, local_num_threads, newcomm)
```

IN comm	original communicator
IN local_thread_index	index of the calling thread (0 to local_num_threads – 1) on the current MPI process in comm
IN local_num_threads	total number of threads issuing this call on the current MPI process in comm
OUT newcomm	new communicator based on threads treated as MPI processes

# MPI\_Comm\_thread\_register

## Basic language bindings

**C:**

```
int MPI_Comm_thread_register(MPI_Comm comm,  
    int local_thread_index, int local_num_threads,  
    MPI_Comm *newcomm)
```

**Fortran:**

```
MPI_COMM_THREAD_REGISTER(COMM,  
    LOCAL_THREAD_INDEX, LOCAL_NUM_THREADS,  
    NEWCOMM, IERROR)
```

```
INTEGER COMM, LOCAL_THREAD_INDEX,  
    LOCAL_NUM_THREADS, NEWCOMM, IERROR
```

# Example: OpenMP parallel section

! Any other thread creation method can go here

!\$OMP parallel num\_threads(4)

```
call MPI_COMM_THREAD_REGISTER(&
    MPI_COMM_WORLD, OMP_GET_THREAD_NUM(), &
    OMP_GET_NUM_THREADS(), NEWCOMM, IERROR)
```

!

!       Whatever MPI operations in NEWCOMM

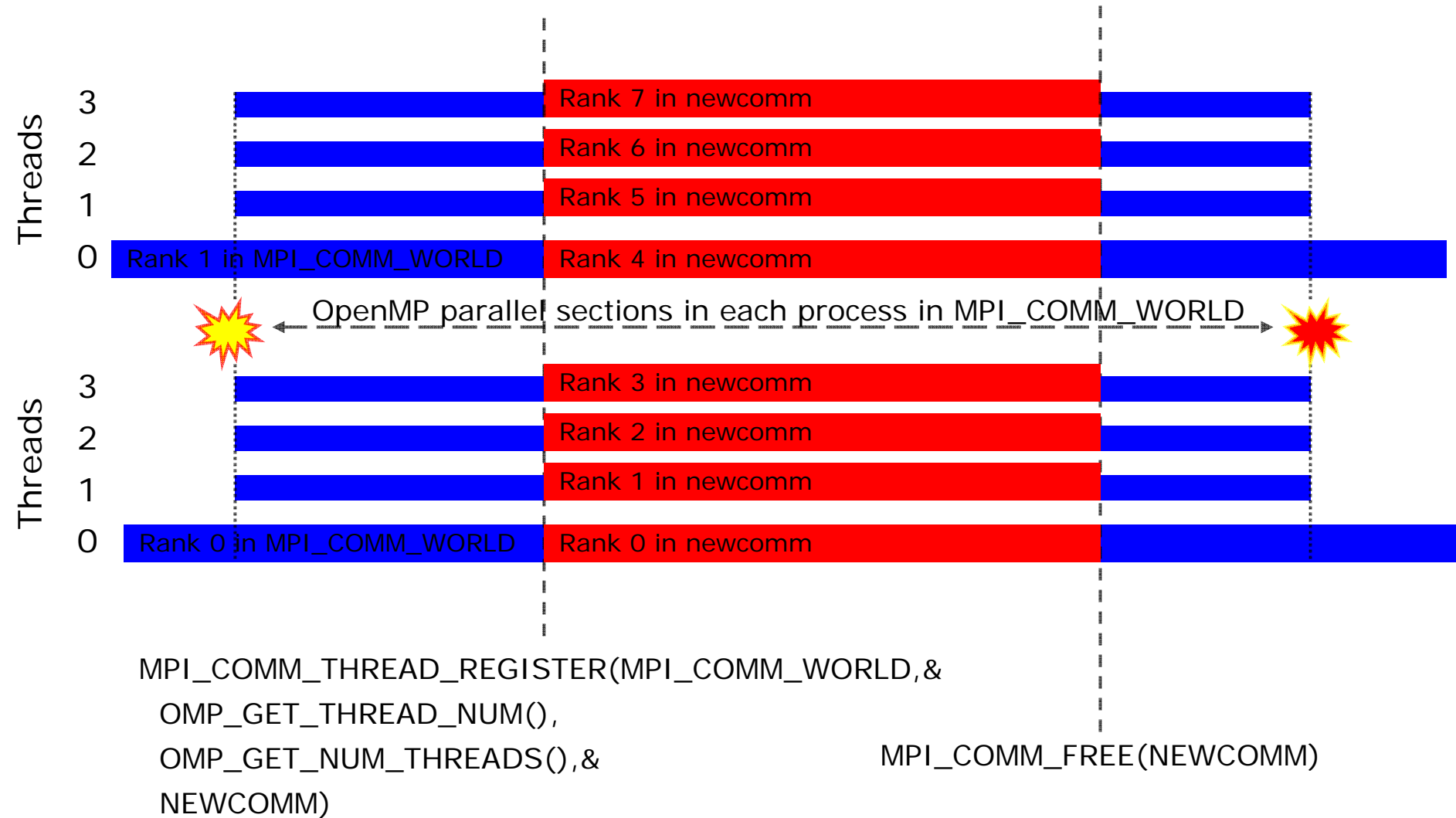
!

```
call MPI_COMM_FREE(NEWCOMM, IERROR)
```

! Any suitable thread termination method can go here

!\$OMP parallel end

# Example: Event timeline



# Example: Notes

- The method of thread creation/termination is out of the scope of this proposal
  - OpenMP is but one possibility
- The number of threads per process can differ from process to process depending on the application logic
  - The value of the `local_num_threads` argument will facilitate synchronization
- While `newcomm` exists, both MPI and thread communication and synchronization mechanisms can be used within their respective scopes
  - It is user responsibility to make sure that those mechanisms do not conflict

# Extensions

- One can imagine not all threads calling the `MPI_Comm_thread_register`
  - The minimum of 1 thread per MPI process in the comm is however necessary. Use another comm if you need to skip some processes.
- One can think of using a `global_thread_index` or `color` instead of the `local_thread_index` in the `MPI_Comm_thread_register`
  - This would allow global thread reordering, resulting in arbitrary newcomm configurations, e.g., all odd threads grouped in round robin fashion across all MPI processes in comm, etc.
  - This is not really necessary as the usual communicator management calls can also be used on the newcomm