



**Barcelona
Supercomputing
Center**
Centro Nacional de Supercomputación



EXCELENCIA
SEVERO
OCHOA

Introducing the Task-Aware MPI (TAMPI) library for hybrid programming

PRACE-6IP WP8

LoSync - Synchronisation reducing programming
techniques and runtime support

Mark Bull (EPCC) and Vicenç Beltran(BSC)
(vbeltran@bsc.es)



OmpSs-2

Programming Model

<https://pm.bsc.es/ompss-2>

<https://github.com/bsc-pm/tampi>

18/06/2019

PRACE 6-IP, Frankfurt, ISC2018

Outline

- **Overview LoSync project**
- Task-Aware MPI (TAMPI)
 - Support for blocking MPI operations (OmpSs-2)
 - Support for non-blocking MPI operations (OpenMP & OmpSs-2)
 - Performance evaluation

Overview LoSync project

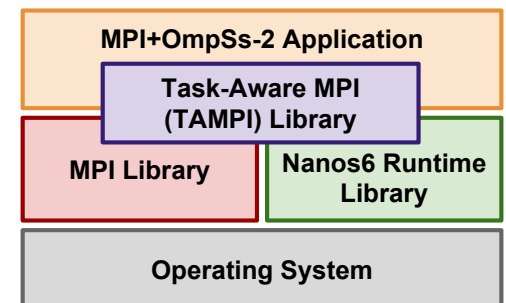
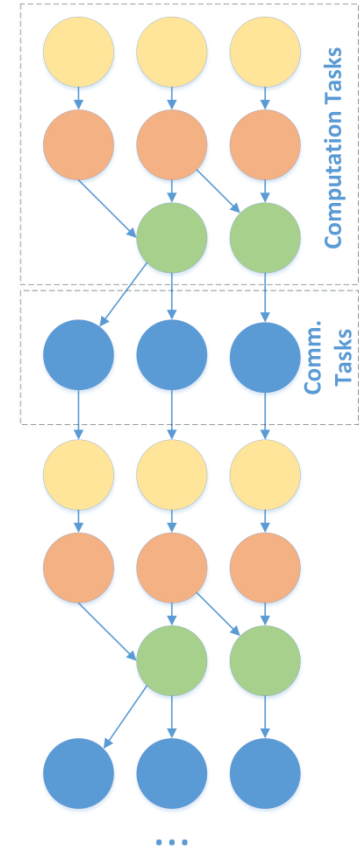
- LoSync: Synchronisation reducing programming techniques and runtime support
 - Improve the scalability of applications by removing unnecessary synchronisation and serialisation, and fully realising opportunities to overlap calculation and communication.
 - Make use of modern features of well-standardised APIs, to ensure portability and relevance
 - Hybrid programming approach, but not “traditional” MPI + OpenMP

Outline

- Overview LoSync project
- **Task-Aware MPI (TAMPI)**
 - Support for blocking MPI operations (OmpSs-2)
 - Support for non-blocking MPI operations (OpenMP & OmpSs-2)
 - Performance evaluation

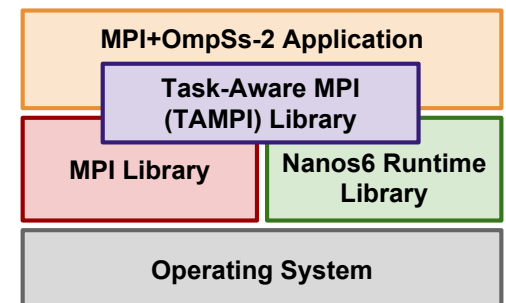
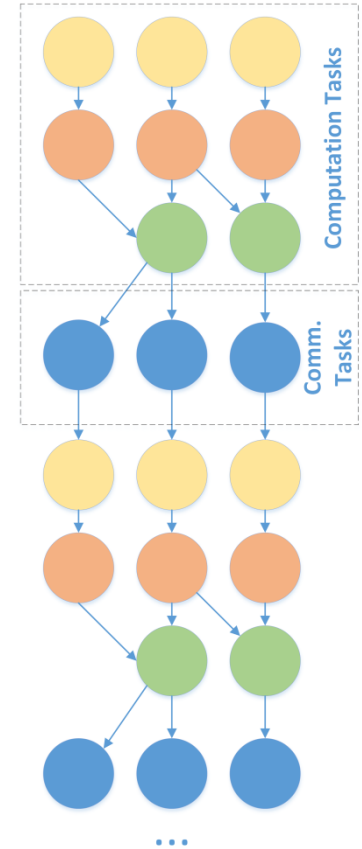
Task-Aware MPI (TAMPI)

- TAMPI is a library to ease hybrid MPI+ OpenMP programming
- TAMPI main benefits
 - Enables **straightforward** use of task with communication primitives
 - No deadlocks! **Safe** and **parallel** execution of **communication** tasks
 - Exposes **more parallelism** (removing artificial deps.)
 - Natural **overlap** of communications and computations phases
- Compatible with **mainstream MPI** implementations
- Currently supported by
 - **OmpSs-2** (v. 2018.11 or greater)
 - **Modified** implementation of LLVM Intel **OpenMP** runtime
 - Not released yet
 - Based on OpenMP 5.0 **detach** clause
- Available on GitHub!
 - <https://github.com/bsc-pm/tampi>



Task-Aware MPI (TAMPI)

- TAMPI provides two modes
 - **Blocking Mode (MPI_TASK_MULTIPLE):**
 - Safe & efficient support for **blocking MPI operations** called from inside tasks (*MPI_Recv*, *MPI_Bcast*...)
 - Supported just by **OmpSs-2**
 - **Non-Blocking Mode (TAMPI_lwait):**
 - Efficient support for **non-blocking MPI operations** from inside tasks (*MPI_Issend*, *MPI_Igather*...)
 - Supported by both **OpenMP** and **OmpSs-2** 😊
- But let's explain them one at a time...

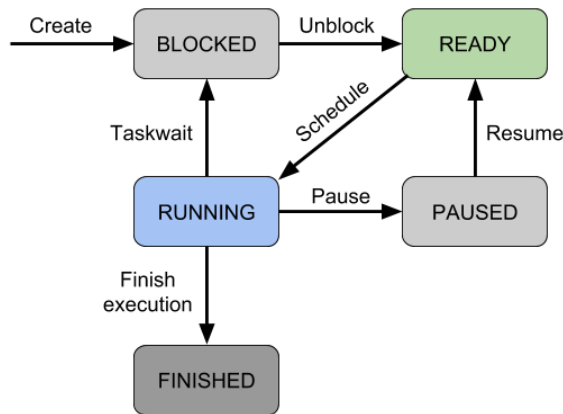


Outline

- Overview LoSync project
- Task-Aware MPI (TAMPI)
 - **Support for blocking MPI operations (OmpSs-2)**
 - Support for non-blocking MPI operations (OpenMP & OmpSs-2)
 - Performance evaluation

Blocking Mode (OmpSs-2)

- Targets the “deadlock-free” & efficient execution of **blocking MPI operations** from inside tasks
- **Virtualizes** the **execution resources** (e.g., hardware threads) of the system when tasks call blocking MPI functions
 - The calling tasks are “blocked” but...
 - The underlying cores running them are **not blocked inside MPI!**
 - So these cores can **execute other ready** tasks while MPI operations are pending!
- Applications can **progress** although many communication tasks are executing blocking MPI calls



```
#include <TAMPI.h>
// ...
#pragma oss task in(senddata[i]) out(recvdata[i])
{
    MPI_Send(&senddata[i], 1, MPI_INT, 1, tag,
            MPI_COMM_WORLD);
    // The send() operation has already been completed

    MPI_Recv(&recvdata[i], 1, MPI_INT, 0, tag,
            MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    // The recv() operation has already been completed
    printf("%d", recvdata[i]);

    // Now we call a 3rd party library that uses
    // MPI inside => OK!
    external_library_function(&recvdata[i]);
}
```

Blocking Mode (OmpSs-2)

- Blocking MPI functions maintain their **blocking semantics**
 - They return once the operation completes
- These **blocking** operations are supported
 - **Primitives:** *MPI_Recv*, *MPI_Send*, *MPI_Ssend*, *MPI_Rsend* and *MPI_Bsend*
 - **Collectives:** *MPI_Bcast*, *MPI_Barrier*, *MPI_Gatherv*, etc.
 - **MPI_Wait** and **MPI_Waitall**
- **MPI_TASK_MULTIPLE**
 - The blocking mode is activated when requesting this new threading level through *MPI_Init_thread()*
 - Defined in “*TAMPI.h*”

```
#include <TAMPI.h>
// ...
#pragma omp task in(senddata[i]) out(recvdata[i])
{
    MPI_Send(&senddata[i], 1, MPI_INT, 1, tag,
             MPI_COMM_WORLD);
    // The send() operation has already been completed

    MPI_Recv(&recvdata[i], 1, MPI_INT, 0, tag,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    // The recv() operation has already been completed
    printf("%d", recvdata[i]);

    // Now we call a 3rd party library that uses
    // MPI inside => OK!
    external_library_function(&recvdata[i]);
}
```

Outline

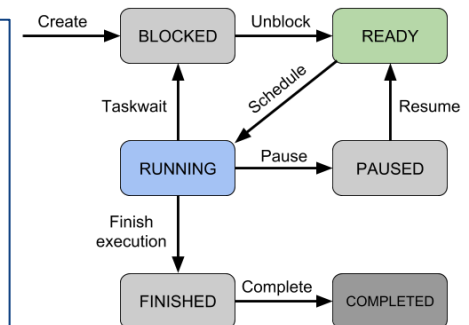
- Task-Aware MPI (TAMPI)
 - Support for blocking MPI operations (OmpSs-2)
 - **Support for non-blocking MPI operations (OpenMP & OmpSs-2)**
 - Performance evaluation

Non-Blocking Mode (OpenMP & OmpSs-2)

- Targets the efficient execution of **non-blocking MPI operations** from inside tasks
- TAMPI allows to **bind** non-blocking MPI **requests** with tasks, so the task will not release its dependencies until the MPI requests have been completed
- **TAMPI_Iwait** and **TAMPI_Iwaitall**
 - New **asynchronous** and **non-blocking** functions that bind the completion of the calling task to the finalization of the passed MPI requests
 - The calling task will release its dependencies when (1) it finishes its execution AND (2) all bound MPI requests complete
- Requires the correct declaration of task **dependencies** over the comm. **data buffers!**

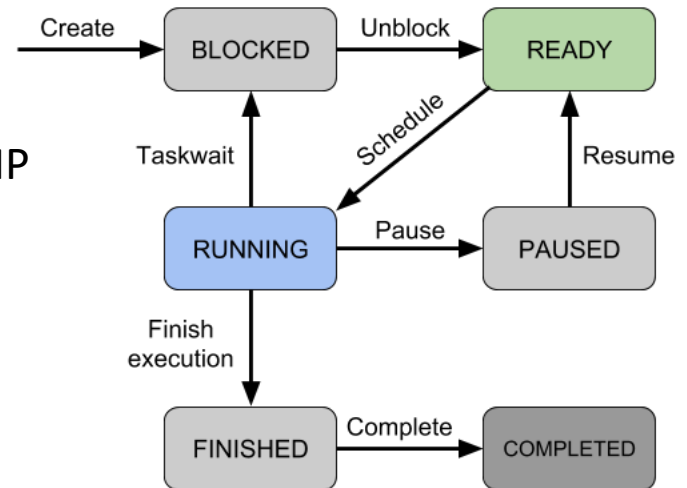
```
#pragma omp task depend(out: recvdata[i], status)
{
    MPI_Request request;
    MPI_Irecv(&recvdata[i], 1, MPI_INT, 0, tag, MPI_COMM_WORLD, &request);
    TAMPI_Iwaitall(1, &request, &status);
    // recvdata and status cannot be accessed yet!
}
#pragma omp task depend(in: recvdata[i], status)
{
    check_status(&status);
    process_data(&recvdata[i]);
}
```

```
int TAMPI_Iwait(MPI_Request *request, MPI_Status *status);
int TAMPI_Iwaitall(int count, MPI_Request *requests,
                  MPI_Status *statuses);
```



Non-Blocking Mode (OpenMP & OmpSs-2)

- This mode is automatically activated when initializing MPI with the standard `MPI_THREAD_MULTIPLE`
- Orthogonal to the previous mode
- Supported by both OmpSs-2 and deriv. OpenMP



```
#pragma omp task depend(out: recvdata[i], status)
{
    MPI_Request request;
    MPI_Irecv(&recvdata[i], 1, MPI_INT, 0, tag, MPI_COMM_WORLD, &request);
    TAMPI_Iwaitall(1, &request, &status);
    // recvdata and status cannot be accessed yet!
}
#pragma omp task depend(in: recvdata[i], status)
{
    check_status(&status);
    process_data(&recvdata[i]);
}
```

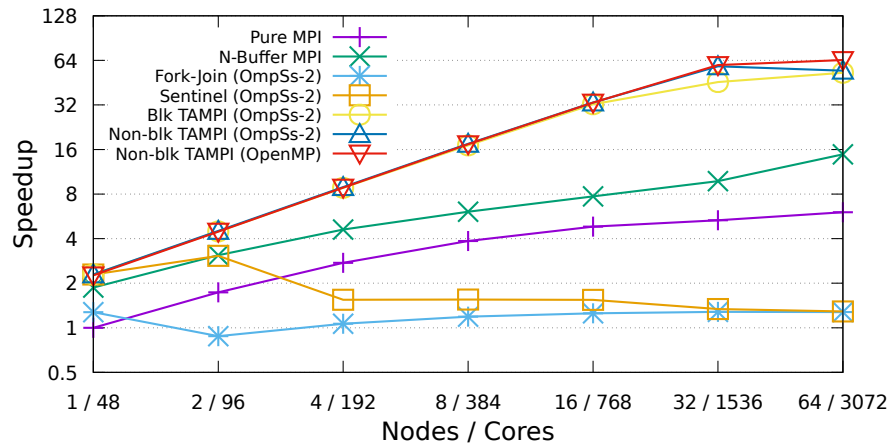
```
int TAMPI_Iwait(MPI_Request *request, MPI_Status *status);
int TAMPI_Iwaitall(int count, MPI_Request *requests,
                  MPI_Status *statuses);
```

Outline

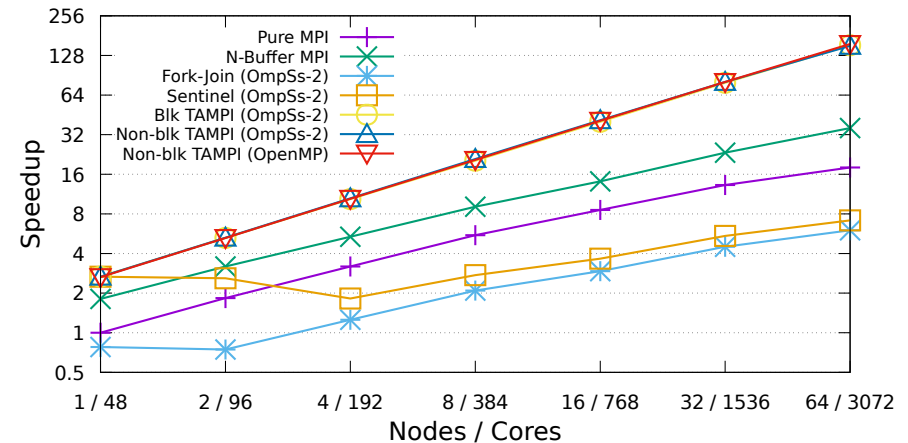
- Task-Aware MPI (TAMPI)
 - Support for blocking MPI operations (OmpSs-2)
 - Support for non-blocking MPI operations (OpenMP & OmpSs-2)
 - **Performance evaluation**

Tiled Gauss-Seidel (MN4, 48 cores x Node)

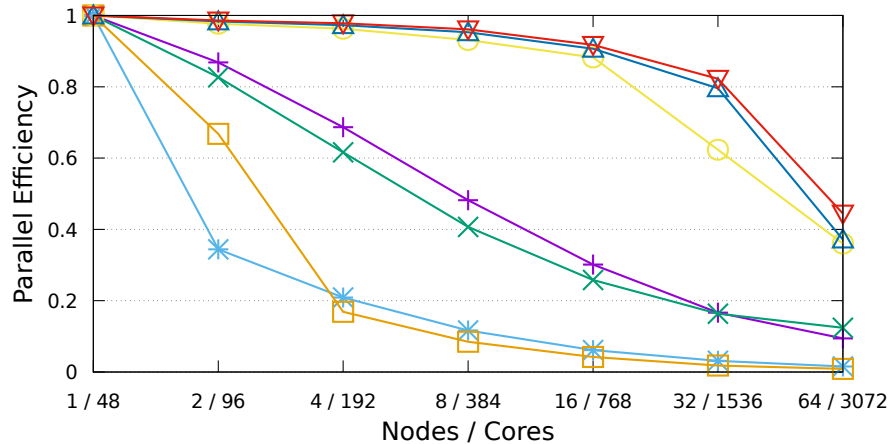
Gauss-Seidel Strong Scaling (64K² matrix, 1000 its)



Gauss-Seidel Weak Scaling (32K² elem/node, 1000 its)



Gauss-Seidel Strong Scaling (64K² matrix, 1000 its)



Gauss-Seidel Weak Scaling (32K² elem/node, 1000 its)

