Wikiprint Book

Title: Integrating TAMPI and ParaStationMPI NAM windows

Subject: DEEP - Public/User_Guide/TAMPI_NAM

Version: 25

Date: 01.05.2024 21:47:48

Table of Contents

Integrating TAMPI and ParaStationMPI NAM windows

Table of contents:

Quick Overview

Heat Benchmark

In this section, we exemplify the use of TAMPI and NAM windows through the Heat benchmark. We use an iterative Gauss-Seidel method to solve the Heat equation, which is a parabolic partial differential equation that describes the distribution of heat in a given region over time. This benchmark simulates the heat diffusion on a 2D matrix of floating-point elements during multiple timesteps. The 2D matrix is logically divided into 2D blocks and may have multiple rows and columns of blocks. The computation of an element at position `M[r][c]` in the timestep `t` depends on the value of the top and left elements (`M[r-1][c]` and `M[r][c-1]`) computed in the current timestep `t`, and the right and bottom elements (`M[r][c+1]` and `M[r+1][c]`) from the previous timestep `t-1`. We can extrapolate this logic in the context of blocks so that a block has a dependency on the computation of its adjacent blocks. Notice that the computation of blocks in a diagonal is fully concurrent because there is no dependency between them.

There are three different MPI versions, and all of them distribute the 2D matrix across ranks assigning consecutive rows of blocks to each MPI rank. Note that the matrix is distributed by blocks vertically but not horizontally. Therefore, an MPI rank has two neighboring ranks: one above and another below. The exceptions are the first and last ranks since they have a single neighbor. This distribution requires the neighboring ranks to exchange the external rows (halos) from their boundary blocks in order to compute their local blocks in each timestep.

This benchmark is publicly available in the ?https://pm.bsc.es/gitlab/DEEP-EST/apps/Heat repository. The first version is based on an MPI-only parallelization, while the other two are hybrid MPI+OmpSs-2 leveraging tasks and the TAMPI library. We briefly describe each one below:

- `01.heat_mpi.bin`: A straightforward **MPI-only** implementation using **blocking MPI primitives** (MPI_Send and MPI_Recv) to send and receive the halo rows. The computation of blocks and exchange of halos inside each rank is completely sequential.
- `02.heat_itampi_ompss2_tasks.bin`: A hybrid **MPI+OmpSs-2** version leveraging **TAMPI** that performs both computation and communications using **tasks** with **data dependencies**. It instantiates a task to compute each of the blocks inside each rank and for each of the timesteps. It also creates a sending and receiving tasks to exchange the block halo rows for each of the boundary blocks. The execution of tasks follows a **data-flow model** because tasks declare the dependencies on the data they read/modify. Moreover, communication tasks call **non-blocking MPI primitives** and leverage the **non-blocking mechanism of TAMPI** (`TAMPI_Iwait`), so communications are fully non-blocking and **asynchronous** from the user point of view. Communication tasks issue non-blocking communications that are transparently managed and periodically checked by TAMPI. These tasks do not explicitly wait for their communication, but they delay their completion (asynchronously) until their MPI communications finish.
- `03.heat_tampirma_ompss2_tasks.bin`: An implementation similar to `02.heat_itampi_ompss2_tasks.bin` but using **MPI RMA operations** (`MPI_Put`) to exchange the block halo rows. This program leverages the MPI active target RMA communication using the **MPI window fences** to open/close RMA access epochs. It uses the **TAMPI** library and the new integration for the `MPI_Win_ifence` synchronization function. In this way, we use `TAMPI_Iwait` to bind the completion of a communication task to the finalization of a `MPI_Win_ifence`. Therefore, the opening/closing of RMA access epochs is completely non-blocking and asynchronous from the user point of view. We assume the calls to `MPI_Put` are non-blocking. Finally, as an optimization, we register **multiple MPI RMA** windows for each rank to allow **concurrent** communications through the different RMA windows. Each RMA window holds a part of the halo row that may belong to multiple logical blocks. Each communication task exchanges the part of the halo row assigned to a single MPI window.

Requirements

The requirements of this application are shown in the following lists. The main requirements are:

- The **GNU** or **Intel®** Compiler Collection.
- The **ParaStationMPI** installation supporting **multi-threading** and featuring the **libNAM** integration that allows access to NAM memory regions through MPI RMA windows.
- The **Task-Aware MPI (TAMPI)** library which defines a clean **interoperability** mechanism for MPI and OpenMP/OmpSs-2 tasks. It supports both blocking and non-blocking MPI operations by providing two different interoperability mechanisms. Downloads and more information at ?https://github.com/bsc-pm/tampi.
- The **OmpSs-2** model which is the second generation of the **OmpSs** programming model. It is a **task-based** programming model originated from the ideas of the OpenMP and StarSs programming models. The specification and user-guide are available at ?https://pm.bsc.es/ompss-2-docs/spec/ and ?https://pm.bsc.es/ompss-2-docs/user-guide/, respectively. OmpSs-2 requires both **Mercurium** and **Nanos6** tools. Mercurium is a source-to-source compiler which provides the necessary support for transforming the high-level directives into a parallelized version of the application. The Nanos6

runtime system provides the services to manage all the parallelism in the application (e.g., task creation, synchronization, scheduling, etc.). Downloads at ?https://github.com/bsc-pm.

- The NAM software allowing access to NAM memory.

In this benchmark, we use the NAM memory to perform checkpointing of the matrix that we are computing. During the execution of the application and every a few timesteps, the benchmark instantiate multiple tasks that save the whole matrix into a NAM region. There is a task for saving the data of each block into that region and may run in parallel.

Building & Executing on DEEP

The instructions to build and execute the Heat benchmark with NAM checkpointing will appear here soon.

References

- ?https://pm.bsc.es/ompss-2
- ?https://github.com/bsc-pm
- ?https://github.com/bsc-pm/tampi
- ?https://en.wikipedia.org/wiki/Gauss-Seidel_method
- ?https://pm.bsc.es/gitlab/DEEP-EST/apps/Heat