PGAS for OP2

Nat Manley, Ben Metzger,
Janek Piskorski, and Nicholas Walker

April 2023

1 Introduction

Partitioned Global Address Space(PGAS) systems provide an alternative memory backend to the industry standard MPI implementations. PGAS systems boast higher throughput and lower latency by avoiding remote host involvement with technologies such as RDMA. This document outlines how to use a PGAS port of OP2, aptly named PGAS-OP2.

During a fourth year project at the University of Warwick, four students have implemented an alternative memory backend for OP2 using the PGAS library GPI-2.

As stated in GPI-2's documentation: GPI-2 implements the GASPI specification (www.gaspi.de), an API specification which originates from the ideas and concepts GPI. GPI-2 is an API for asynchronous communication. It provides a flexible, scalable and fault tolerant interface for parallel applications. [1]

It must be noted, however, that the current incarnation of PGAS-OP2 is bug prone. Difficulties in using GPI-2 have propagated and caused unreliability in some OP2 apps. Thus, it is not recommended to use PGAS-OP2 for anything beyond experimentation and research.

The bugs and their likely causes are outlined later on in the document, after introducing how to use PGAS-OP2.

1.1 MPI Interoperability

Currently, OP2 uses MPI entirely for its distributed computing functionality. Importantly, GPI optionally supports MPI to enable a smooth transition to PGAS systems and one-sided communication.

As such, PGAS-OP2 relies heavily on MPI for the setup and initialisation of OP2, along with the cool-down and performance reports. Leaving the primary bulk of network communication during the compute loops to GPI.

Importantly, PGAS-OP2 can be executed with `mpirun`, much in the same way as normal OP2.

We believe this implementation uses MPI and GPI to their respective strengths.

2 Building OP2 With GPI

Similarly to other build options for OP2, it is necessary to install external tooling that links into the OP2 codebase. Luckily, this process is not dissimilar to the manual compilation and installs of other libraries.

2.1 GPI-MPI Interop

GPI-2 in MPI interoperability mode can be obtained via the following steps:

1. Ensure an install of MPI
2. Clone GPI-2 from their github repository (`https://github.com/cc-hpc-itwm/GPI-2`.

3. Generate the congiuration script with `./autogen.sh`

4. Configure the Makefiles using `./configure --with-mpi --with-ethernet --prefix=$INSTALL_PREFIX`. Consult the GPI `README` if GPI is unable to find the MPI install. Altneratively, modify `--with-ethernet` to `--with-infiniband` should your system support it.

5. Finally, build GPI-2 using the MPI compiler: `CC=mpicc make` and `CC=mpicc make install`

2.2 Application Makefiles

In order to use PGAS-OP2 in your OP2 application, it is necessary to provide additional flags during the build process.

To build OP2, please use the `GPI_INSTALL_PATH` variable to define the install location for GPI's shared objects (i.e. the path defined in the configuration step).

Next, your OP2 application must now be linked with the GPI-2 shared objects stored in the installation directory defined in the configuration previously. This involves adding `-lGPI2` to the compiler flags.

Then, it is necessary to add or modify the previous OP2 library flags to now state `-lop2_gpi` with the OP2 library install path if necessary (i.e. if not part of `LD_LIBRARY_PATH`.

2.3 Example Makefile

PGAS-OP2 supports automatic GPI codegeneration. If you instead prefer manual code translation, an extract from the MG-CFD Makefile[1] has been included as an example in Listing 1.

```
## GPI
$(OBJ_DIR)/mgcfd_gpi_main.o: $(OP2_MAIN_SRC)
   mkdir -p $(OBJ_DIR)
   $(MPICPP) $(CPPFLAGS) $(OPTIMISE) $(MGCFD_INCS) $(OP2_INC) $(HDF5_INC) $(GPI_INC) \
        -DMPI_ON -c -o $@ $^
$(OBJ_DIR)/mgcfd_gpi_kernels.o: $(SRC_DIR)/../gpi_seq/_gpiseqkernels.cpp $(GPISEQ_KERNELS)
   mkdir -p $(OBJ_DIR)
   $(MPICPP) $(CPPFLAGS) $(OPTIMISE) $(MGCFD_INCS) $(OP2_INC) $(HDF5_INC) $(GPI_INC) \
        -DMPI_ON -c -o $@ $(SRC_DIR)/../gpi_seq/_gpiseqkernels.cpp
$(BIN_DIR)/mgcfd_gpi: $(OP2_GPI_OBJECTS)
   mkdir -p $(BIN_DIR)
   $(MPICPP) $(CPPFLAGS) $(OPTIMISE) $^ $(MGCFD_LIBS) \
      -lm $(OP2_LIB) -lop2_gpi $(PARMETIS_LIB) $(KAHIP_LIB) $(PTSCOTCH_LIB) $(HDF5_LIB) $(
     GPI_LIB)\
      -o $@
```

Listing 1: Example codegeneration Makefile

3 Limitations

Potential for Timeouts

This is the primary bug preventing greater use of PGAS-OP2.

There is currently potential for timeouts (resulting in an ungraceful abort process) when performing back to back exchanges of the same dat between nodes. Currently, this issue can be fixed with the use of `op_gpi_barrier()` to force synchronisation between `op_par_loop`s, and while this is clearly not ideal for optimal performance, tested applications have demonstrated comparable performance to the MPI back-end for smaller node counts.

Static Temp *dat* Segment Size

Temporary dats have been given their own set of four segments (one for each halo type), which function as heaps when creating and releasing these temporary dats. However, the segments have a static size defined at compile time, with a default size of `4MB`. If the total size of

[1] https://github.com/Highly-Predictable-Calamity/MG-CFD-app-OP2/blob/master/Makefile

temporary dats is likely to be greater than this value, the default value can be changed in `op2/include/op_lib_gpi.h`.

```
44 #define GPI_HEAP_SIZE (1024 * 4096) /* 1024 x 4K pages ~ 4MiB Pages */
```

Listing 2: Line defining temp dat segments size in `op2/include/op_lib_gpi.h`.

Limited Number of Ranks and *dats*

Due to the way we use GPI notifications to encode information about the sending rank and the *dat* data it is sending, there are a limited number of ranks and dats possible. The notification ID field of a notification is 16 bits, and currently 8 of the bits are used for rank, and 8 for dat, resulting in a limit of 256 ranks or dats.

Currently these proportions could be adjusted; a higher number of dats and a lower number of ranks for example.

This adjustment can be performed before compile time by adjusting `NOTIF_SHIFT` in `op2/include/op_gpi_core.h`.

One *InfiniBand* Process per Node

There is a limit of one *InfiniBand* process per node; any more than one would try to reuse ports. Furthermore, anymore than one InfiniBand process per node is not likely to be desirable; optimisations using OpenMP and CUDA are more likely to provide better performance on one node. This is a GPI-2 level problem, and out of the scope of PGAS-OP2.

For on-node parallelism, please refer to OpenMP. Currently PGAS-OP2 has not been tested with OpenMP optimisations also.

Custom Ports

The GPI back-end currently utilises a specific port range. Should these need to be changed, manual GPI source edits will be necessary.

This is also out of the scope of PGAS-OP2.

References

[1] GPI-2, *Gpi-2 webpage*. [Online]. Available: `http://gpi-site.com.www488.your-server.de/docs/`.