HPCLatAm 2013, pp. 210-215 (short paper)
Session: Prospective and Ongoing Projects
C. Garcia Garino and M. Printista (Eds.)
Mendoza, Argentina, July 29-30, 2013

Use of the PGAS model for high performance computing in Beowulf Clusters

Jorge D'Elía[⋆], Lisandro Dalcin, Sofía Sarraf, Ezequiel López,
Laura Battaglia, Gustavo Ríos Rodríguez, and Victorio Sonzogni

Centro Internacional de Métodos Computacionales en Ingeniería (CIMEC)
Instituto de Desarrollo Tecnológico para la Industria Química (INTEC)
Universidad Nacional del Litoral - CONICET Güemes 3450, 3000-Santa Fe, Argentina
{jdelia,dalcin,sonzogni}@intec.unl.edu.ar,
{sssarraf,ezequiel.jose.lopez}@gmail.com,
lbattaglia@santafe-conicet.gob.ar,
gusadrr@yahoo.com.ar
http://www.cimec.org.ar

Abstract. The Partitioned Global Address Space (PGAS) is a parallel programming model that has been developed for distributed memory computers. Furthermore, it can be used in High Performance Computing (HPC) on Beowulf clusters oriented to scientific and engineering applications through computational mechanics. As it is known, the PGAS model is the basis for, among others, some multi-paradigm programming languages such as the UPC (Unified Parallel C) and the Coarray Fortran (CAF or Fortran 2008), as well as the library Global Arrays (GA). All these resources are extensions to provide one-side communication. This work summarizes some of the activities carried out in one of the clusters available in CIMEC, as well as some ideas for a Message Passing Interface (MPI) implementation of coarrays on a fortran compiler.

Keywords: Partitioned Global Address Space, Unified Parallel C, Coarray Fortran, High Performance Computing, computational mechanics

1 Introduction

The Partitioned Global Address Space (PGAS) [1, 2] is a programming model for parallel computing on distributed memory computers, as well as on "traditional" Beowulf clusters.

1.1 PGAS memory model

The PGAS memory model assumes a global address space along with an explicitly Single Program and Multiple Data (SMPD) programming model, and it

[⋆] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET, Argentina): grant PIP 112-20111-00978, Universidad Nacional del Litoral (UNL, Argentina): grant CAI+D 2009–III–4–2, Agencia Nacional de Promoción Científica y Tecnológica (ANPCyT, Argentina): grants PICT 2492–10, PICT 1141–07, PICT-PRH 2009-0147, EU-IRSES, grant PIRSES-GA-2009-246977 can seen as share memory model in a distributed memory computer in which the locality of the cache data is taken into account. In this model, a distintion between local and remote memory references is made to take advantage of the local data, with the ultimate goal of increasing the performance in distributed memory computers. There are SPMD processes that share a part of their address space, while a part of the share space is local to each process. Data structures are allocated either globally or privately, where the first ones are distributed across the adress space. Remote global data are accesible to any process with simple assignament or dereference operations, while the compiler and runtimes convert these operations into messages among processes on a distribute memory computer. Although programs based in this model work mainly with their local data and requering communication through global data structures only, most of them also provide Application Programming Interfaces (APIs) for bulk communication and syncronization.

1.2 PGAS runtimes and libraries

Three of the PGAS runtimes and libraries are the following:

1. Global-Address Space Networking (GASNet) runtimes [3]: it is a language-indpendent and low-level networking layer that provides network-indpendent primitives for one-side communication, and it used as tool for runtime libraries. Inside GASNet there are two layers, the lower one is a general interface implemented directly on the top of several network architecture, while the upper layer provides remote memory access as well as collective operations in a high level mode.
2. Aggregate Remote Memory Copy Interface (ARMCI) runtimes [4]: it is a library that allows remote memory copy functions, one-sided put and get, as well as mutual exclusive operations, handling regular and irregular distributed data. It has compatibility with MPI for hybrid shared-distributed memory computers, and it has blocking and noblocking APIs, where the last ones can be used in some cases to overlap computations and communications. It is worth to say that the official site website is out-to-date, although there are plans to release the 1.5 version, according to its developers, as well as there is the completly rewritten implementation ARMC-MPI [5].
3. Global Arrays (GA) library [6]: it provides a shared memory style programming environment in distributed array data structures. From the user perspective, a global array is used as if it is stored in the shared memory. The primary target architectures are the massively-parallel distributed-memory and scalable shared-memory systems. It divides shared data structures into local and remote portions, where the local portion of the shared memory is assumed to be faster to access, while the remote portion is considered slower to access. Also, it allows to combine shared-memory and message-passing styles of programming in a same program, where it inherits the execution environment from a MPI library that started the parallel program (see Global Arrays Manual).

1.3 PGAS programming languages

Two programming languages that support the PGAS model are the following:

1. Unified Parallel C (UPC) [7]: it is an extension of the C programming language for HPC on distributed memory computers, combining the shared memory and the control over the data layout of the message-passing models, using a SMPD computation pattern. There is a single shared and partitioned address space, where the variables my be directly read and written by any processor, although each variable is physically associated with a single processor, while the amount of parallelis is fixed at the program startup.
2. Coarray Fortran [8] (or Fortran 2008 [9]): the coarray model is included in the lastest Fortran 2008 standard as a part of the language definition in order to handling the work and data distribution in a parallel program. A SMPD programming model is emplyed for the work distribution, where a single prgram is replicated a fixed number of times. Each replication is called an imagen, and it has its own set of data, and executes mostly asynchronously, while a syncronization can be requested thorough specific statements. The coarrays allow to define the data distribution among the memory images, they are like ordinary variables although they have their indices in square brackets for memory access across the images, i.e. reference without square brackets involve local data, while a square bracket reference imply a communication across the images.

2 Some software based on the PGAS model

The CIMEC clusters are based on the free software available for Linux Operative Systems (OS), particularly Fedora distributions, where the following runtimes, libraries and programming languages based on the PGAS model were tested:

- PGAS runtimes and libraries:
  - Global-Address Space Networking (GASNet) runtimes [3];
  - Aggregate Remote Memory Copy Interface (ARMCI) runtimes [4];
  - Global Arrays (GA) library [6];
  - UPCBLAS library (for parallel matrix computations in UPC) [10].
- PGAS programming languages:
  - Unified Parallel C (UPC) [7]: the Berkeley UPC [11] and GUPC [12] compilers;
  - Coarray Fortran [8] (or Fortran 2008 [9]): the OpenUH [13] and G95 [14] compilers. The first one without restrictions, while G95 only until 4 computer nodes.

The Berkeley UPC and Open-UH compilers and GA were built on the CIMEC Coyote cluster [15], and they are currently in an experimental stage. These were built over the GASNet and ARMCI layers, in addition to the ubiquitous MPI layer, through the OpenMPI [16] distribution. The currently OS is the Fedora 17 distribution.

3 A MPI based library for the coarray model

Nowdays, the coarray model is included in the lastest Fortran 2008 standard. However, since several fortran compilers are built on the basis of the C/C++ ones, e.g. the GNU–GFortran [17] and Rose [18] compilers, there is not a native way to add the coarray model. One option is to build a MPI based library for supporting the coarray model. In this case, it should be ensured that the final library is as neutral as possible regards to an end user. However, due to the fact that the MPI binaries are not yet compatibles among the available MPI implementations, e.g. OpenMPI [16] or MPICH [19], this issue should be overcomed. Among other possibilities, the following alternatives can be considered, from the simplest case to more elaborated choices:

1. A static library *libcaf_mpi.a* which uses a specific MPI implementation when the library is built (e.g. OpenMPI or MPICH). However, it only works with the MPI distribution available when the library is built;
2. A dynamic library *libcaf_mpi.so* that uses a specific MPI implementation (e.g. OpenMPI or MPICH) when the library is built. However, again, it only works with the MPI implementation available when the library is built. From an usability point of view, this option is not very different than the previous one;
3. A symbolic link *libcaf_mpi.so* that points to dynamics libraries *libcaf_mpich.so* or *libcaf_openmpi.so.* The sysadmin can manage the link using system tools like "alternatives". Linux distributions usually manage this infrastructure by themselves without additional work required from the fortran compiler side. However, regular (non-root) users cannot switch the backend MPI. This is only available on POSIX systems;
4. Different dynamic libraries named *libcaf_mpi.so* are built for each MPI implementation, each of them installed in different directories, e.g.

   ```
   <some-prefix>/mpich/libcaf_mpi.so
   <some-prefix>/openmpi/libcaf_mpi.so
   ```

   By using the `modules` tool, users can select the preferred MPI implementation, e.g. `module load mpich2-x86_64` in Fedora. This works by adding entries in the LD_LIBRARY_PATH environment variable. Linux distributions usually manage this infrastructure by themselves and do not require additional work from the fortran compiler side. Regular users are able to choose the preferred MPI implementation, and the dynamic linker loads the appropriate *libcaf_mpi.so*;
5. A dynamic library "libcaf_mpi.so" built by the dlopen() tool. This option could be practical if the number of MPI functions were not too large (BTW, how many?), and it can be built on both Linux and Windows OS;
6. A dynamic library "libcaf_mpi.so" that is not linked with the MPI library, but uses the dlopen() and dlsym() to load and access the contents of a specific "MPI-linked" dynamic library, e.g. "libcaf_mpi_{mpich2|openmpi|other}.so".

In this way, "libcaf_mpi.so" does not depend on any MPI implementation, but acts as a thin wrapper to the specific CAF + MPI library. By using environment variables or `rc` configuration files, the user can choose the preferred library to open at runtime with dlopen(). Although dlopen() is specific to POSIX systems, similar mechanisms are available on Windows OS.

On the one hand, from all the previous options, the last one is considered the more robust and convenient since regular users can switch MPI library (as opposed to option 3), and environment variables are not strictly required (as in option 4), since sysadmin could choose a default MPI by editing a general config file (e.g. located in /etc) and regular users could make a different choice by editing a config file at $HOME. On the other hand, the last option can be seen as "over-engineered" since having a simple libcaf_mpi.a in the path, e.g. in the $MPI/lib directory, could be the best option for most typical usage. Having a "libcaf_mpi.so" at the same location, will also allow dynamic switching. Nevertheless, a potential usage for a dynamically linked version can be useful for closed-source software.
For instance, regarding the current state of the coarray in the GNU–GFortran compiler given in http://gcc.gnu.org/wiki/CoarrayLib#WARNING, two GNU Fortran coarray communication libraries are available:

– A single-image library consisting of stubs, which allows to use a single image without recompiling and is also useful for debugging. However, using -fcoarray=single will produce a faster code.
– A MPI version which is a wrapper that calls to a library implementing the MPI library, and it is planned as a purely MPI 1.x version. The current rough version also uses few MPI 2.x features.

Nevertheless, the GNU–GFortran implementation of coarrays based in MPI is currently in an embryonic state mainly due to other priorities in the implementation. For example, the mpi.c file of the official repo of the GNU–GFortran is a stub, where there are calls to MPI_Init, MPI_Barrier, or similar. An implementation can be developed using the MPI 1.x, although it would also be possible with MPI 2.x or MPI 3.x, because MPICH already support MPI 3.x. The use of MPI 3.x would make an easier implementation based on *windows*, a.k.a *remote memory access* or *one-sided communication*, all in the MPI_Win_XXX API, that were included into the MPI2 to support the programming model of a shared memory on a distributed memory environment. Therefore, a plausibe option could be to assess if it is reasonably easy to start with MPI3 instead MPI2.

4 Conclusions

The use of PGAS models in Beowulf clusters through, for instance, the Berkeley UPC+GUPC and OpenUH compilers for the UPC and CAF programming language extensions, respectively, are another resource for parallel computing on distribute memory systems with the advantage of being available with free software resources on Unix-like operative systems. Additionally, some alternatives for building a MPI based library for the coarray model on fortran compilers were considered, from the simplest to more elaborated, and they could be used as a basis for a work project.

Acknowledgments. This work has received financial support from Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET, Argentina, grant PIP 112-20111-00978), Universidad Nacional del Litoral (UNL, Argentina, grant CAI+D 2009–III-4–2), Agencia Nacional de Promoción Científica y Tecnológica (ANPCyT, Argentina, grants PICT 2010-2492/16, PICT 2009-1141/07, PICT-PRH 2009-0147), EU-IRSES (PIRSES-GA-2009-246977), and was performed with the *Free Software Foundation /GNU-Project* resources such as GNU–Linux-OS, GNU–GFortran, GNU–Octave, GNU–Git, GNU–Doxygen, and GNU–GIMP, as well as other Open Source resources as LaTeX.

References

1. Partitioned Global Address Space: http://www.pgas.org (2013)
2. Diaz, J., Muñoz Caro, C., Niño, A.: A survey of parallel programming models and tools in the multi and many-core era. IEEE Trans. Parallel Distr. Syst. **23**(8) (2012) 1369–1388
3. Global-Address space networking: http://gasnet.cs.berkeley.edu (2013)
4. Aggregate Remote Memory Copy Interface (ARMCI): http://www.emsl.pnl.gov/docs/parsoft/armci (2013)
5. ARMCI-MPI: http://wiki.mpich.org/armci-mpi/index.php/Main_Page (2013)
6. Global Arrays (GA): http://www.emsl.pnl.gov/docs/global (2013)
7. Unified Parallel C: http://upc.gwu.edu (2013)
8. Coarray Fortran: http://www.co-array.org (2013)
9. Fortran 2008: http://www.nag.co.uk/sc22wg5 (2013)
10. González-Domínguez, J., Martín, M.J., Taboada, G.L., Touriño, J., Doallo, R., Mallón, D.A., Wibecan, B.: UPCBLAS: A library for parallel matrix computations in unified parallel C. Concurrency Comput. Pract. Ex. **24**(14) (2012) 1645–1667
11. Berkeley UPC - Unified Parallel C: http://upc.lbl.gov (2013)
12. GNU Unified Parallel C (GNU UPC): http://www.gccupc.org (2013)
13. OpenUH compiler: http://www2.cs.uh.edu/~openuh/index.shtml (2013)
14. G95 Fortran compiler: http://www.g95.org (2013)
15. Coyote cluster: http://www.cimec.org.ar/coyote (2013)
16. Open MPI: http://www.open-mpi.org (2013)
17. GNU Gfortran compiler: http://gcc.gnu.org/wiki/GFortran (2013)
18. ROSE compiler infrastructure: http://www.rosecompiler.org (2013)
19. MPICH: http://www.mpich.org (2013)