

## Partitioned Global Address Space Approaches for Exascale Computing

Mirko Rahn, Fraunhofer, ITWM



Partitioned Global Address Space (PGAS) approaches have become a hot topic for the exascale computing domain. For instance, several exascale projects funded by the European Commission or the US Department of Energy extend current PGAS approaches. An example is the EC-funded [EPiGRAM](#) [1] project that has identified the gaps to be filled when attempting to master the Exascale challenge with PGAS.

Many PGAS approaches are relatively new when compared to MPI, the de-facto standard approach in HPC. The PGAS approaches are quite diverse and will further develop and improve in the future. In the EPiGRAM project, we have categorized the different PGAS development efforts.

As a first step, we have distinguished between languages and APIs. PGAS languages either extend a certain HPC programming language (e.g. UPC extends the C language and Fortran supports Coarray features) or are standalone languages (like Chapel). A non-experienced developer can prepare a parallel code with PGAS languages easily. The bottlenecks are the compilers rather than the implementation details.

APIs provide tools to implement the communication between the nodes explicitly. APIs can be used directly by application developers (like GPI) or by tool developers (like ARMCI which is the basis of GA and Gasnet which is the basis of a UPC implementation). Thus the scope of the APIs is quite different.

### Fault Tolerance

It is expected that fault tolerance will become more important issue when moving to exascale systems. Fault tolerance is a transversal issue. A minimum requirement for the PGAS programming model would be the ability to handle a node failure and to incorporate check-pointing information. A node failure could be handled in two ways. First, it is possible to redistribute the computation on fewer nodes. A second approach is to use spare nodes in addition. Some PGAS approaches like GPI possess the first failure tolerance mechanism. For most languages or APIs no statements about fault tolerance can be found. It is to be seen if fault tolerance will become as important as we think.

### Migration Path to PGAS

An incremental migration path, as proposed by most PGAS models analysed in the EPiGRAM project, requires a degree of interoperability between the programming models. There should also be a transparent and scalable way to map between the Processing Element (PE) identifiers (e.g. ranks in MPI, images in CAF and OpenSHMEM). At present there is no single standards body that can attempt to codify and enforce this interoperability between different programming models.

Typical large-scale MPI codes make an extensive use of sub-communicators to identify subgroups of ranks. Without a standardized and scalable mechanism for translating these sub-communicators into corresponding structures in PGAS programming models, incremental porting will be more difficult. Such a mechanism does not yet exist.

### Theoretical Limitations

Even simple concepts like having a globally valid rank could be an issue when it comes to high number of processes. This is explained with the concept of a rank identifying a process globally. Rank names are correlated with some machine or some connection or alike. The application uses the rank to address a remote process, thus the communication layer (either the language or the API) needs to resolve the connection. This is done by using a table in which a connection for each rank is mapped. The size of this table scales as  $O(P)$ , where  $P$  is the number of processes and  $O(P^2)$  summed up over all processes. Thus, the amount of memory that is needed to store the table is growing faster than the size of the memory that is available. In fact, there are reports that starting MPI or UPC programs on a large number of cores failed due to insufficient memory resources.

That said, one needs to implement the rank without a table of size  $O(P)$  but with local data of size  $O(1)$  only. This is possible, e.g. by storing nodes of a distributed tree. But then, to determine the connection that is associated with a given rank, one needs to traverse a distributed data structure thus increasing the latency by a factor of  $O(\log P)$ .

Altogether, one can choose between a non-scaling implementation that has low latency and a scaling implementation with high latency. This seems to be the case for each and every global property. And also the solution to trade latency for memory seems to be a generic one. There are two solutions to this theoretical limitation. First, one could imagine a system that does not have names for each entity. It is not clear whether or not such a system could be implemented for all the important applications. Second, one could write applications

## Partitioned Global Address Space Approaches for Exascale Computing

Published on Scientific Computing (<http://www.scientificcomputing.com>)

---

that can tolerate high latencies. In general these kind of theoretical limitations are not gaps in the programming model but they become important issues when the number of components grows to exascale.

*To learn more about EPIGRAM and the European exascale research, join the BoF titled "[Exascale Research: The European Approach \[2\]](#)," at 2.15pm - 3.15pm, Tuesday, June 24, 2014, in Hall 5. Alternatively you can talk to Rahn and team at their booth #833 in the exhibition hall.*

### **About the author:**

Dr Mirko Rahn is from Fraunhofer, ITWM. He studied Computer Science and Logics and received his Ph.D. in Theoretical Computer Science in 2008 from the University of Karlsruhe. In 2009 he was member of a team that set a new record in sorting huge amounts of data. In 2009 he joined the Fraunhofer ITWM in Kaiserslautern, Germany. He works on new concepts and tools to manage and process very large data sets, especially data sets from the seismic domain. His key interests are high performance parallel processing, compiler and language technology and workflow management.

*This blog was originally published on [ISC Events \[3\]](#) and is reproduced here with permission.*

### **Source URL (retrieved on 05/31/2016 - 3:46am):**

<http://www.scientificcomputing.com/blogs/2014/05/partitioned-global-address-space-approaches-exascale-computing>

### **Links:**

[1] <http://www.epigram-project.eu/>

[2] [http://www.isc-events.com/isc14\\_ap/sessiondetails.htm?t=session&o=108&a=select&ra=byday](http://www.isc-events.com/isc14_ap/sessiondetails.htm?t=session&o=108&a=select&ra=byday)

[3] <http://www.isc-events.com/isc14/blogs.html>