MPMD with Coarray Fortran (PGAS): Load Balancing – Example Program

by Michael Siehl

www.mpmd-with-coarray-fortran.de

May 2016 (160528)

Load Balancing – Example Program

We did modify our original MPMD example program slightly to illustrate a simple PGAS load balancing technique.

Our original MPMD example program did use 13 coarray images to execute:
- an InitialManager object on image 1
- 3 TeamManager objects on images 2, 6, and 10 respectively
- 9 TeamMember objects on images 3, 4, 5, 7, 8, 9, 11, 12, and 13 respectively.

This could easily be altered from outside the source code by modifying the TeamManagers.txt and the TeamMembersX.txt files. Nevertheless, the original example program did require that any of these objects got executed on its own coarray image.

By now, we want to illustrate (very basically) the ease of using coarrays (or PGAS memory in general) to develop parallel code that might be used equally well for the following both purposes, even at the same time:
(1) initiate distributed object execution on several distinct coarray images (in truly parallel), as well as
(2) initiate purely local (non-distributed) object execution on a single coarray image (sequentially, so to speak).

To achieve this, we did modify few subroutines of the original example program. The modifications are in the following subroutines (as well as in one emulated enumeration) and are marked with the date-stamp *160414* therein:
- Subroutine IIimma_SYNC_CheckActivityFlag (of the ImageManager ADT)
- Enumeration (emulated) OOOPimscEnum_ImageActivityFlag (of the ImageStatus coarray wrapper)
- Subroutine OOOPtmem_Start (of the TeamMember ADT)
- Subroutine OOOPimma_Start (of the InitialManager ADT)
- Subroutine OOOPtema_Start (of the TeamManager ADT)
- Subroutine OOOPimma_Start (of the ImageManager ADT)
- Subroutine IItema_ActivateTeamMember Image (of the TeamManager ADT).

Running the modified example program with the original TeamManagers.txt and TeamMembersX.txt files does lead to the same runtime behavior as with the original code (with the addition that the images will give an 'execution finished' message by now).

For illustration purposes, instead of the 13 coarray images of the original program, assume that we have only 11 coarray images available for doing the same computations. Our load balancing example program does allow to reuse the TeamManager images to execute a TeamMember object afterwards. It does also allow to pick up only a fraction of the TeamManager images for that purpose. Since we have 2 coarray images less available, we choose to reuse 2 TeamManager images. To do so, we did modify our TeamManagers.txt and TeamMembersX.txt files accordingly. Thus, the load balancing example program uses 11 coarray images to execute:
- an InitialManager object on image 1
- 3 TeamManager objects on images 2, 6, and 9 respectively
- 9 TeamMember objects on images 3, 4, 5, 6, 7, 8, 9, 10, and 11 respectively.

Effectively, the load balancing example program does execute the same computations, no matter if it uses 11 or 13 coarray images. To do so with only 11 images, it does reuse two TeamManager images (6 and 9) to execute TeamMember objects afterwards.

Nevertheless, the crucial point here is not just the simple reuse of coarray images. More importantly, the load balancing

example program does use the same PGAS parallel programming code to initiate both, the remote execution as well as the local execution of TeamMember objects. To achieve that it uses the ImageStatus_CA coarray wrapper for both, remote and local data transfer through PGAS memory (see subroutine IItema_ActivateTeamMemberImage of the TeamManager ADT). The key is also our simple synchronization technique (see the second do loop of subroutine IIimma_SYNC_CheckActivityFlag of the ImageManager ADT): We use a simple flag in PGAS memory (the ImageActivityFlag) to signal (and check for) synchronizations between objects (TeamManager and TeamMember) but not necessarily between distinct coarray images.

Of course, this load balancing example program is still very primitive, but yet using such a PGAS coding style routinely might be very helpful:

- to free the programmer (algorithm design as well) greatly from (early) decisions between purely local (non-distributed) object execution or remotely distributed object execution, and thus
- for easy development of dynamic load balancing code later on.