Scaling hybrid coarray/MPI miniapps on Archer

Luis Cebamanos*, Anton Shterenlikht†, Jose D. Arregui-Mena‡ and Lee Margetts‡
*Edinburgh Parallel Computing Centre (EPCC), The University of Edinburgh, King's Buildings, Edinburgh EH9 3FD, UK
Email: l.cebamanos@epcc.ed.ac.uk
†Department of Mechanical Engineering, The University of Bristol, Bristol BS8 1TR, UK, Email: mexas@bris.ac.uk
‡School of Mechanical, Aero and Civil Engineering, The University of Manchester, Manchester M13 9PL, UK
Email: jose.arregui-mena@manchester.ac.uk, Lee.Margetts@manchester.ac.uk

***Abstract*—We have developed miniapps from MPI finite element library ParaFEM and Fortran 2008 coarray cellular automata library CGPACK. The miniapps represent multi-scale fracture models of polycrystalline solids. The software from which these miniapps have been derived will improve predictive modelling in the automotive, aerospace, power generation, defense and manufacturing sectors. The libraries and miniapps are distributed under BSD license, so these can be used by computer scientists and hardware vendors to test various tools including compilers and performance monitoring applications. CrayPAT tools have been used for sampling and tracing analysis of the miniapps. Two routines with all-to-all communication structures have been identified a primary candidates for optimisation. New routines have been written implementing the nearest neighbour algorithm and using coarray collectives. Scaling limit for miniapps has been increased by a factor of 3, from about 2k to over 7k cores. The miniapps uncovered several issues in CrayPAT and Cray implementation of Fortran coarrays. We are working with Cray engineers to resolve these. Hybrid coarray/MPI programming is uniquely enabled on Cray systems. This work is of particular interest to Cray developers, because it details real experiences of using hybrid Fortran coarray/MPI programming for scientific computing in an area of cutting edge research.**

***Keywords*-miniapps; Fortran; coarrays; MPI; library; profiling; CrayPAT, ARCHER;**

I. Introduction

Fortran coarrays are a feature of Fortran 2008 standard published in 2010 [1], [2], [3]. To date there are very few mixed coarray/MPI programs, or platforms which support this. Cray is perhaps still the only such platform, and ECMWF codes are probably the best example [4].

We have developed a two-way hierarchical concurrent multi-scale fracture model [5]. The structural level is represented via finite element (FE) approximation using the ParaFEM MPI library, `http://parafem.org.uk`, [6] The microstructural level is represented by cellular automata (CA) [7] using CGPACK Fortran coarray library, `https://sf.net/p/cgpack`, [8]. Both libraries are written in modern Fortran and distributed under 2-clause BSD license.

ParaFEM is a highly scalable MPI finite element library written in Fortran 95 [9]. It has been used in large scale simulations, e.g. in nuclear fusion research [10], [11] and biomechanics [12], [13].

CGPACK is a scalable cellular automata library written in Fortran 2008 with extensive use of coarrays. Work on CGPACK started in 2013 [14] on HECToR. It has since been ported to Intel and OpenCoarray/GCC platforms, `http://www.opencoarrays.org`, [15], [16]. CGPACK is being actively developed, including contributions from the Software Sustainability Institute, `www.software.ac.uk`, and a grant from the embedded CSE programme of ARCHER.

Although the idea of a multi-scale CAFE model is not new, [17], [18], [19], the current CAFE framework was designed specifically for HPC systems. Fortran coarrays is a PGAS language, which some HPC experts predict to be better suited to exa-scale hardware [20], [21]. Coarrays exploit the power and simple syntax of multi-dimensional Fortran arrays, while MPI allows for very fine tuning of parallelisation, thus hybrid MPI/coarray algorithms can deliver highly optimised parallel code.

A. ParaFEM/CGPACK (MPI/coarray) interface

The CA space is implemented as a 4D array coarray, with a 3D coindex set: `space(:,:,:,:)[:,:,:]`. The first 3 array dimensions are used to store each cell's Cartesian coordinates. The fourth array dimension allows for multiple types (layers) of microstructural information to be stored and processed simultaneously. At present two layers are used - grains and fracture surfaces. In contrast to the "box-shaped" CA space, the FE model can be of arbitrary shape. In other words CA uses a structured grid, whereas FE uses an unstructured grid. In addition, partition of the FE model into MPI chunks is totally independent of coarray images. This presents severe problems for linking coarray CA part of the model with the MPI FE part as described below.

In general the FE domain in solid mechanics can be very irregular. A schematic example is shown in Fig. 1. The microstructural CA space is always a structured grid, as shown schematically in Fig. 2. Sometimes, the CA space will be fully inside the FE model, but in general, the CA space can be of arbitrary size and orientation with respect to the FE domain, depending on what deformation and/or fracture

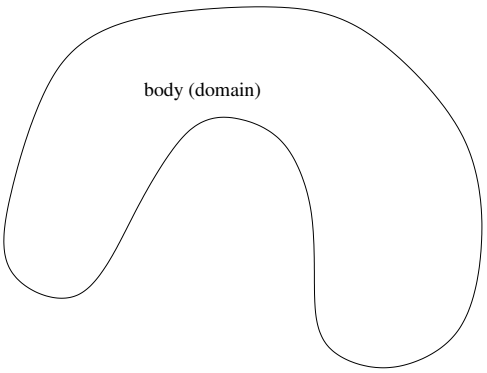

Figure 1. Schematic of the FE domain.

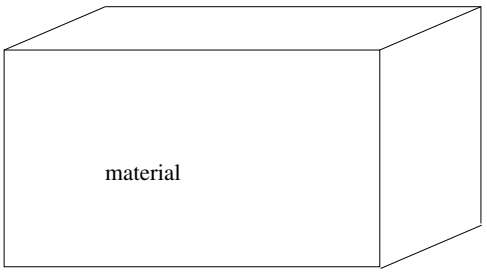

Figure 2. Schematic of the CA space.

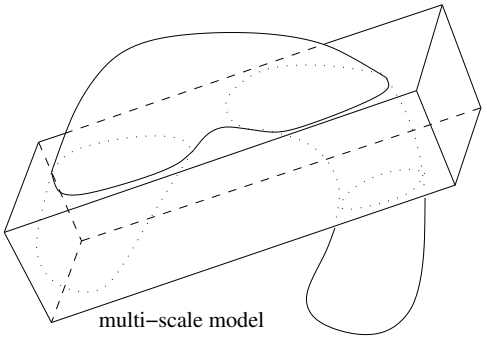

Figure 3. Schematic of a multi-scale CAFE model composed of the FE domain superimposed with the CA material space.

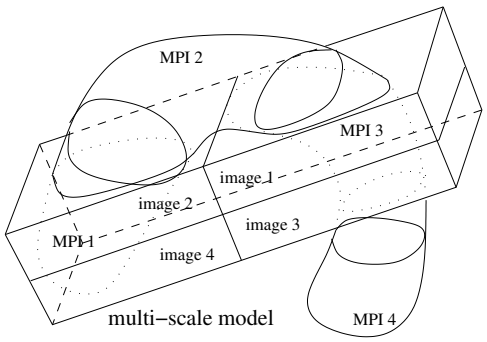

Figure 4. Possible partition of the multi-scale model on 4 PEs.

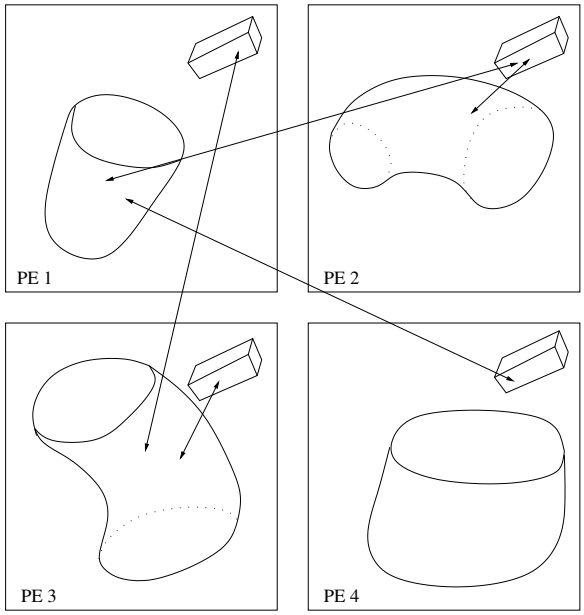

Figure 5. Schematic of communications between the MPI (FE) and the coarray (CA) parts of the coarray/MPI (CAFE) hybrid model on 4 PEs.

phenomena are to be studied with it, as shown in Fig. 3. Some FEs will occupy the same physical space as some CA cells. These FEs and cells form a two-way macro/micro multi-scale CAFE model. However, as indicated in Fig. 3, in general, there will be cells occupying physical space outside of the body, and there will be FEs occupying physical space not covered by the CA material. These FEs and cells must be excluded from the analysis.

It is assumed in the following that in a hybrid coarray/MPI program there is always identical number of MPI processes and coarray images, which on Cray is just a number of processing elements, PEs.

A schematic partition of the CAFE model on 4 PEs is shown in Fig. 4. The labels denote on which PE the corresponding parts of the model are stored. For example, "image 1" and "MPI 1" parts of the model are stored on PE 1. However, these FEs do not share physical space with these CA cells. Instead cells on image 1 share physical space with FEs on PE 3, labelled "MPI 3". This is important because information transfer is required only between CA and FE which occupy the same physical space. So in this example MPI part of the model stored on PE 3 will have to communicate with coarray part of the model stored on PEs 1 and 3.

Communications between the MPI (FE) and the coarray (CA) parts of the coarray/MPI (CAFE) hybrid model are shown schematically with arrows in Fig. 5. The imbalance in the communication pattern is clear. The FE part of the model stored on PE 4 will not communicate with CA at all. However, the FE part of the model stored on PE 1 will need to communicate with CA coarrays stored on PEs 2 and 4.

The mapping of FE to CA is established via a private allocatable array of derived type:

```
type mcen
  integer :: image
  integer :: elnum
  real :: centr(3)
end type mcen
type( mcen ), allocatable :: lcentr(:)
```

based on coordinates of FE centroids calculated by each MPI process and stored in a coarray of derived type with allocatable array component:

```
type rca
  real, allocatable :: r(:,:)
end type rca
type( rca ) :: centroid_tmp[*]
```

which is allocated as

```
allocate( centroid_tmp%r(3, nels_pp) )
```

where `nels_pp` is the number of FE stored on this PE.

There are two different routines which establish `lcentr` on each image from `centroid_tmp`. Subroutine `cgca_pfem_cenc` implements an all-to-all communication pattern, i.e. each images reads `centroid_tmp` from every image. Subroutine `cgca_pfem_map` uses temporary arrays and coarray collectives `CO_SUM` and `CO_MAX`, which are described in TS18508 [22] and will be included in the next revision of the Fortran standard, Fortran 2015. At the time of writing coarray collectives are available on Cray systems as extension to the standard [1]. The two routines differ in their use of remote communications. However, both routines implement the same algorithm for establishing `lcentr` - if the centroid of an FE on any image is within the coarray CA array of this image, then this FE is added to `lcentr` on this image.

Fig. 6 schematically shows `lcentr` arrays established on two images P and Q. In this example finite element n, stored on image Q, has centroid coordinates r, which identify a physical location within CA coarray on image P. So this element is stored in `lcentr` array on image P. Finite element m, also stored on image Q, has centroid coordinates u, which identify a physical location within CA coarray also on image Q. So this element is stored in `lcentr` array on image Q. FEs with centroids outside of the CA space are not entered in `lcentr`.

`lcentr` plays the key role in information transfer between the FE and the CA parts of the multi-scale CAFE model.

After `lcentr` has been established, the second important mapping issue can be resolved. CA cells which are outside of the FE model must not be processed. This means no fracture propagation can occur in such cells. However, since there is finite resolution in FE model and in CA, this problem cannot

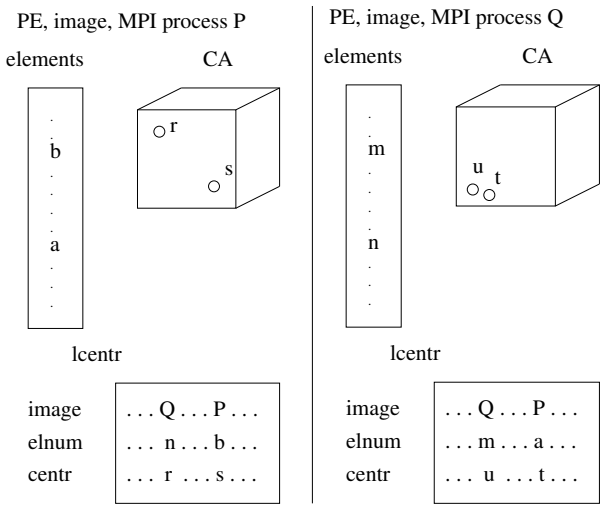

Figure 6. `lcentr` arrays on two images P and Q.

be posed precisely. Depending on the FE size and the CA cell size, a cell can be deemed to lie inside the FE model or out. The algorithm implemented in the ParaFEM/CGPACK interface uses some characteristic distance measure, L_c. The criterion is this - if the distance between a cell and the centroid of any FE in `lcentr` is less than L_c, then this cell is considered to lie inside the FE model, otherwise it is considered to lie outside of the FE model. Cell which lie outside of the FE model are not processed at all in any of the fracture routines. Although these cells represent microstructure in the material layer, this microstructure is simply ignored in all fracture calculations.

This mapping is established with a divide and conquer approach. The algorithm starts by checking boxes of CA cells the size of the whole coarray on each image. If a box in partially in and partially out, it is split into two smaller boxes and the process continues until each box is either fully in, or fully out. If necessary, CA boxes are divided down to single CA cells. This algorithm is implemented in the routines `cgca_pfem_partin`, `cgca_pfem_boxin` and `cgca_pfem_cellin`.

B. Separate scaling of ParaFEM and CGPACK

Individually both ParaFEM and CGPACK libraries showed the potential to scale well into tens of thousands of cores on HECToR, as seen in Figs. 7 and 8, although parallel efficiency of CGPACK is only about 25%.

However, scaling of a library can be understood only as scaling of particular programs using various routines from the library. Scaling might change dramatically depending on which library routines are invoked in a particular executable and in which order. Too many permutations exist for a comprehensive scaling analysis of a library.

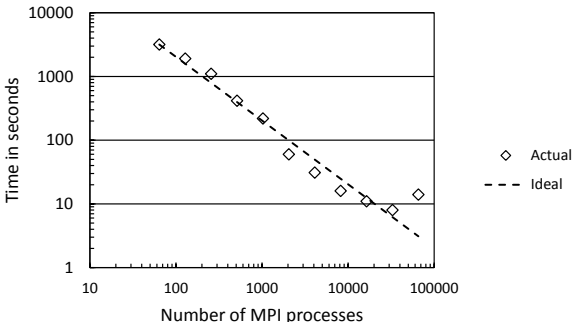

Figure 7. ParaFEM scaling for a 3D transient flow explicit analysis. Reproduced from [6].

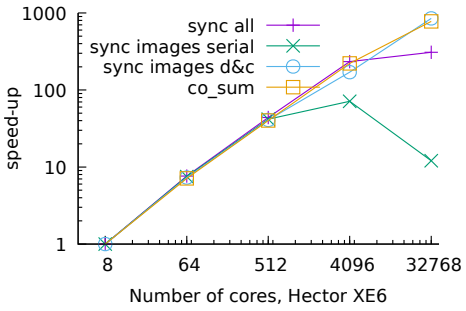

Figure 8. CGPACK 3D solidification scaling with different synchronisation methods. Reproduced from [14].

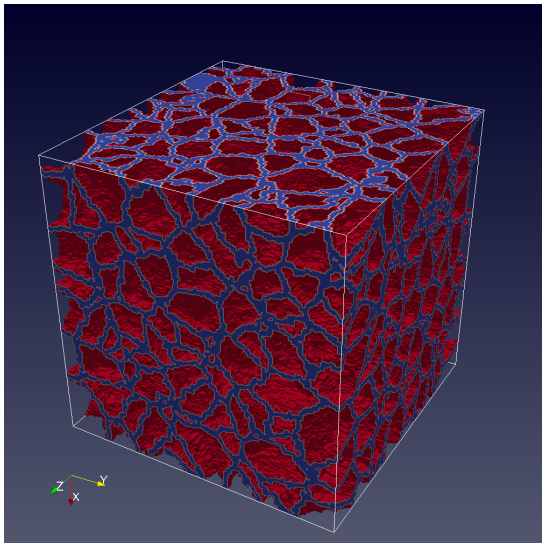

Figure 9. CGPACK simulated crystal boundaries in a polycrystalline material.

II. ParaFEM/CGPACK miniapps

Several driver programs (miniapps) [23] were constructed using both ParaFEM and CGPACK libraries. The programs simulate progressive cleavage propagation in polycrystalline iron. Examples of the miniapp results are shown in Figs. 9-11.

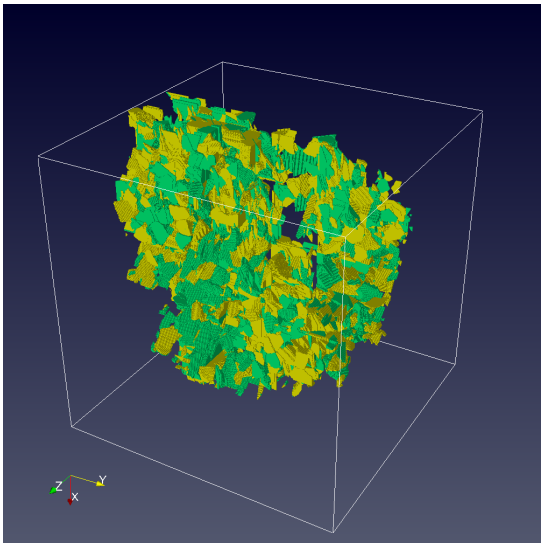

Figure 10. Example of the ParaFEM/CGPACK miniapp simulation. Cracks on $\{110\}$ planes are shown in green and on $\{100\}$ planes are shown in yellow.

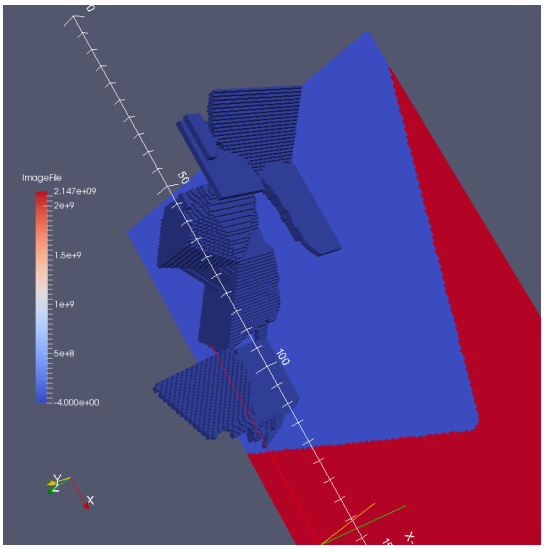

Figure 11. Example of the ParaFEM/CGPACK miniapp simulation. The CA space (blue) intersects the FE domain (red). The cleavage cracks (dark blue) are confined to the CA space.

Crack propagation across crystal boundaries is of particular interest in structural integrity analysis of polycrystal structures and components. A typical CA simulation of equiaxed crystal boundaries is shown in Fig. 9.

Fig. 10 shows the emerging macro-crack, visualised on the CA scale. In this example cracks in individual iron crystals merge to form a macro-crack. The process is driven by the FE stress field.

Fig. 11 shows results from a miniapp where the CA space intersected with the FE domain. Mapping routines `cgca_pfem_partin`, `cgca_pfem_boxin` and

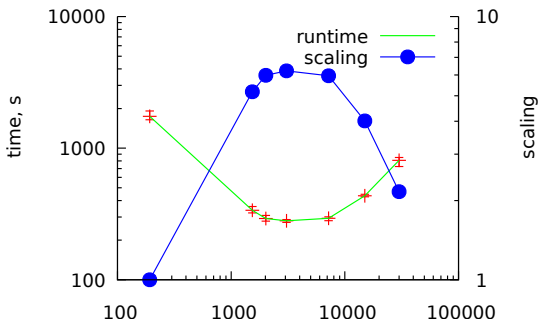

Figure 12. ParaFEM/CGPACK MPI/coarray miniapp scaling on ARCHER XC30 for a 3D problem with 1M FE and 800M CA cells.

`cgca_pfem_cellin`, described in Sec. I-A, were used in this miniapp.

When a crystal boundary is crossed by a crack in the CA coarray on any image, it is important that all other images are notified as soon as possible [8]. If the crack propagation speed is relatively low, and the model resolution is high enough, then it is sufficient to inform the nearest neighbouring images. Subroutine `cgca_gcupdn` implements the nearest neighbour algorithm. In cases when crack propagation speed can be so high that the CA changes can propagate more than the length of the CA coarray in one model iteration an all-to-all algorithm has to be used. It is implemented in subroutine `cgca_gcupda`. Profiling of the miniapps with both these routines, as well as with the nearest neighbour `cgca_pfem_map` and all-to-all `cgca_pfem_cenc` FE to CA mapping routines, see Sec. I, is presented in the next section.

III. Profiling

CrayPat on ARCHER (Cray XC30 system) was used for profiling work. The model with 1M FE and 800M CA cells was used.

Although both ParaFEM and CGPACK independently can scale well to tens of thousands of cores (see Figs. 7 and 8), the initial profiling study showed limited scalability mainly due to an all-to-all remote read routine `cgca_gcupda`. Strong scaling of one of our miniapps which simulates a 3D trans-granular cleavage in polycrystalline iron is shown in Fig. 12. It is clear that this miniapp scales well up to 2000 cores from where the scalability drops dramatically.

As shown in Figs. 13 and 14, a large portion of the total time (over 38%) is spent on `cgca_gcupda` subroutine which indicates it is a clear candidate for further optimization.

The key fragment from this all-to-all routine is shown below. In this routine each coarray image reads a coarray value from all the other images which becomes a communication problem at large number of images. The outer loop starting counter (remote image number) is chosen at random to even out communication load.

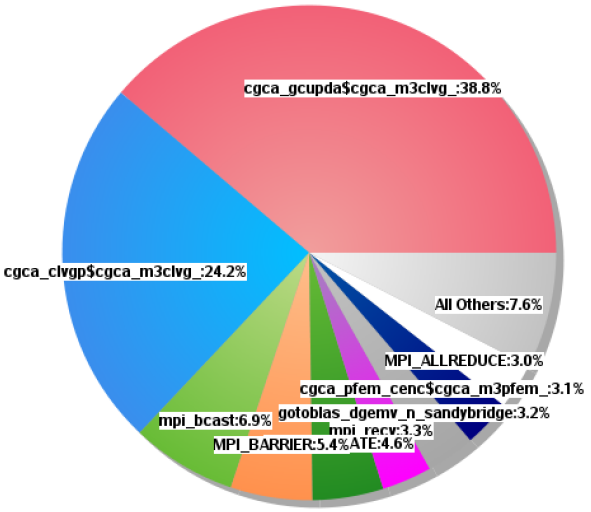

Figure 13. Profiling function distribution for ParaFEM/CGPACK MPI/coarray miniapp with all-to-all routine `cgca_gcupda` at 7200 cores.

```
100.0% | 20,520.4 |    -- |    -- |Total
|-----------------------------------------------------------
|  71.4% | 14,649.9 |    -- |    -- |USER
||----------------------------------------------------------
||  38.7% |  7,950.6 | 913.4 | 10.3% |cgca_gcupda$cgca_m3clvg_
||  24.1% |  4,951.2 | 940.8 | 16.0% |cgca_clvgp$cgca_m3clvg_
||   3.1% |    638.0 |  70.0 |  9.9% |cgca_pfem_cenc$cgca_m3pfem_
||   1.8% |    367.5 | 578.5 | 61.2% |cgca_hxi$cgca_m2hx_
||   1.7% |    346.0 | 196.0 | 36.2% |cgca_clvgn$cgca_m3clvg_
||==========================================================
|  19.8% |  4,061.4 |    -- |    -- |MPI
||----------------------------------------------------------
||   6.9% |  1,413.5 | 356.5 | 20.1% |mpi_bcast
||   5.4% |  1,098.3 | 419.7 | 27.7% |MPI_BARRIER
||   3.3% |    670.0 | 322.0 | 32.5% |mpi_recv
||   3.0% |    615.3 |  61.7 |  9.1% |MPI_ALLREDUCE
||==========================================================
|   8.8% |  1,797.2 |    -- |    -- |ETC
||----------------------------------------------------------
||   4.6% |    950.5 |   5.5 |  0.6% |__DEALLOCATE
||   3.2% |    654.2 | 110.8 | 14.5% |gotoblas_dgemv_n_sandybridge
|==========================================================
```

Figure 14. Raw profiling data for ParaFEM/CGPACK MPI/coarray miniapp with all-to-all routine `cgca_gcupda` at 7200 cores.

```
integer :: gcupd(100,3)[*], rndint, j,&
  img, gcupd_local(100,3)
real :: rnd
:
call random_number( rnd )
rndint = int( rnd*num_images() )+1
do j=rndint, rndint+num_images()-1
 img = j
 if (img .gt. num_images()) &
      img = img - num_images()
  if (img .eq. this_image()) cycle
  :
  gcupd_local(:,:) = gcupd(:,:)[img]
```

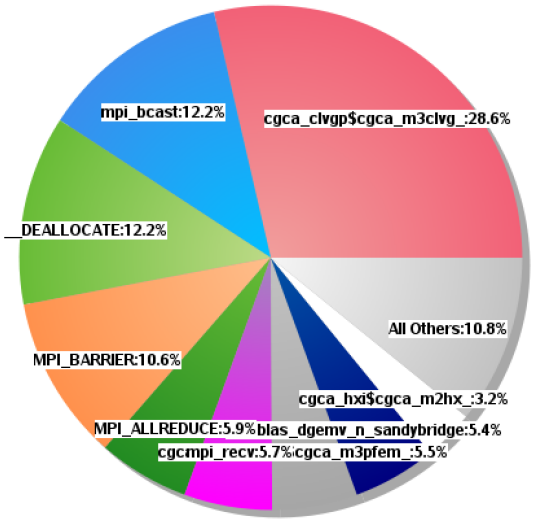

Figure 15. Raw profiling data for ParaFEM/CGPACK MPI/coarray miniapp with the neareast neighbour routine `cgca_gcupdn` at 7200 cores.

```
  :
end do
```

An alternative to an all-to-all algorithm is the nearest neighbour algorithm. As mentioned before, this has been implemented in subroutine `cgca_gcupdn`. The key fragment is shown below.

```
do i = -1 , 1
do j = -1 , 1
do k = -1 , 1
 ! Get the coindex set of the neighbour
 ncod = mycod + (/ i, j, k /)
 :
 gcupd_local(:,:) =     &
 gcupd(:,:)[ncod(1),ncod(2),ncod(3)]
 :
end do
end do
end do
```

It must be emphasised that the nearest neighbour and all-to-all are not identical. In the nearest neighbour case the information is propagated only one image away from the current image. Multiple invocations of the nearest neighbour algorithm are required for changes on any image to reach all images. However, because the nearest neighbour algorithm is known to scale well, it might still outperform all-to-all at high core counts, even if multiple invocations are used. Moreover, for some fracture propagation problems a single invocation of the nearest neighbour will suffice, if crack propagation rates are such that no crack is likely to cross the whole of CA array on an image in one CA iteration.

The execution of the mentioned miniapp execising the

```
100.0% | 12,199.5 |    -- |    -- |Total
|-----------------------------------------------------------
|  44.8% |  5,459.7 |    -- |    -- |USER
||----------------------------------------------------------
||  28.6% |  3,484.0 | 582.0 | 14.3% |cgca_clvgp$cgca_m3clvg_
||   5.5% |    666.1 |  93.9 | 12.4% |cgca_pfem_cenc$cgca_m3pfem_
||   3.2% |    393.1 | 752.9 | 65.7% |cgca_hxi$cgca_m2hx_
||   2.8% |    346.0 | 176.0 | 33.7% |cgca_clvgn$cgca_m3clvg_
||   1.4% |    165.2 |  37.8 | 18.6% |cgca_sld$cgca_m3sld_
||   1.0% |    126.0 |  82.0 | 39.4% |xx14_
||==========================================================
|  36.7% |  4,472.1 |    -- |    -- |MPI
||----------------------------------------------------------
||  12.2% |  1,484.4 | 380.6 | 20.4% |mpi_bcast
||  10.6% |  1,287.9 | 389.1 | 23.2% |MPI_BARRIER
||   5.9% |    714.9 |  90.1 | 11.2% |MPI_ALLREDUCE
||   5.7% |    689.4 | 338.6 | 32.9% |mpi_recv
||   1.5% |    179.1 | 417.9 | 70.0% |MPI_REDUCE
||==========================================================
|  18.5% |  2,256.1 |    -- |    -- |ETC
||----------------------------------------------------------
||  12.1% |  1,480.9 |   4.1 |  0.3% |__DEALLOCATE
||   5.4% |    653.8 |  95.2 | 12.7% |gotoblas_dgemv_n_sandybridge
|==========================================================
```

Figure 16. Profiling function distribution for ParaFEM/CGPACK MPI/coarray miniapp with the neareast neighbour routine `cgca_gcupdn` at 7200 cores.

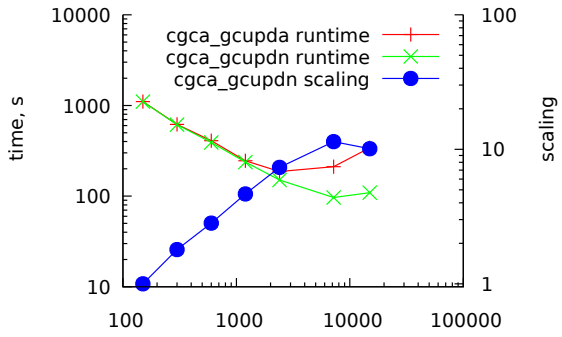

Figure 17. Runtimes and scaling for ParaFEM/CGPACK MPI/coarray miniapp with the nearest neighbour, `cgca_gcupdn`, and all-to-all, `cgca_gcupda`, algorithms.

nearest neighbourgh algorithm clearly demonstrates a considerable reduction in the number of remote reads. This can be seen on Figs. 15 and 16 where the user time is no longer dominated by remote reads between images with the exception of subroutine `cgca_pfem_cenc`.

This optimisation should also be reflected in the miniapp performance since now it will be able to scale to much larger of core counts. Fig. 17 shows that the scaling limit has been increased from 2k, when using all-to-all `cgca_gcupda`, to 7k cores, when `cgca_gcupdn` is used.

As previously mentioned, subroutine `cgca_pfem_cenc` also implements an all-to-all communication pattern which could be replaced with subroutine `cgca_pfem_map`. Subroutine `cgca_pfem_map` relies on the use of temporary arrays and coarray collectives `CO_SUM` and `CO_MAX`.

At present Cray collectives specification differs slightly

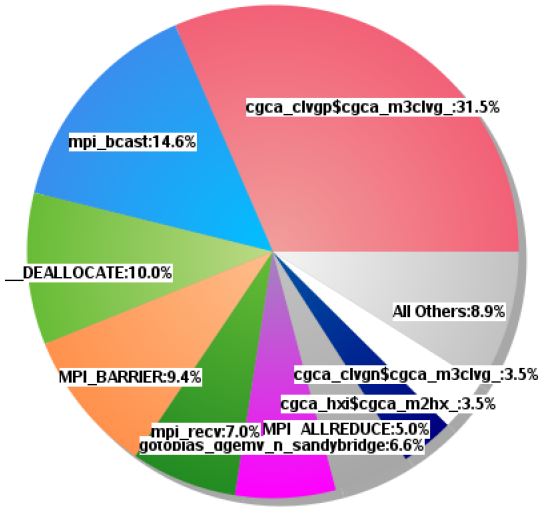

Figure 18. Profiling function distribution with `cgca_gcupdn` and `cgca_pfem_map` at 7200 cores.

from TS 18508 [22]. However, the differences are immaterial for this work, so in the following we just use "collectives" to mean the features of both Cray implementation and TS 18508.

The key fragment of `cgca_pfem_map` is shown below. See Sec. I for details of `centroid_tmp` coarray.

```
integer :: maxfe, pos_start, pos_end, &
 ctmpsize
real, allocatable :: tmp(:,:)
! Calculate the max number of FE
! stored on this image
maxfe = size( centroid_tmp%r, dim=2 )
ctmpsize = maxfe
call co_max( source = maxfe )
allocate(tmp( maxfe*num_images(), 5), &
 source=0.0 )
! Each image writes its data in a unique
! portion of tmp.
pos_start = (this_image() - 1)*maxfe + 1
pos_end = pos_start + ctmpsize - 1
tmp( pos_start : pos_end, 1 ) = &
 real( this_image(), kind=4 )
! Write element number *as real*
tmp( pos_start : pos_end, 2 ) = &
 real((/ (j, j = 1, ctmpsize) /), kind=4)
! Write centroid coord
tmp( pos_start : pos_end, 3:5 ) = &
 transpose( centroid_tmp%r(:,:) )
call co_sum( source = tmp )
```

A large temporary array, of length in the order of the maximum number of FE on any image times the number of images, is required. This approach might prove problematic

```
Table 1:  Profile by Function
  Samp% |    Samp |    Imb. |  Imb. |Group
        |         |    Samp | Samp% | Function
        |         |         |       |  PE=HIDE
        |         |         |       |   Thread=HIDE
 100.0% | 9,903.4 |      -- |    -- |Total
|-------------------------------------------------------------
|  43.6% | 4,321.6 |      -- |    -- |USER
||------------------------------------------------------------
||  31.4% | 3,110.7 |   589.3 | 15.9% |cgca_clvgp$cgca_m3clvg_
||   3.5% |   346.0 |   513.0 | 59.7% |cgca_hxi$cgca_m2hx_
||   3.5% |   342.0 |   175.0 | 33.8% |cgca_clvgn$cgca_m3clvg_
||   1.2% |   116.3 |     4.7 |  3.9% |cgca_pfem_map$cgca_m3pfem_
||   1.1% |   106.8 | 1,537.2 | 93.5% |cgca_clvgsd$cgca_m3clvg_
||   1.0% |    99.9 |    24.1 | 19.5% |cgca_sld$cgca_m3sld_
||============================================================
|  38.4% | 3,803.6 |      -- |    -- |MPI
||------------------------------------------------------------
||  14.6% | 1,446.6 |   350.4 | 19.5% |mpi_bcast
||   9.4% |   932.4 |   473.6 | 33.7% |MPI_BARRIER
||   7.0% |   689.5 |   371.5 | 35.0% |mpi_recv
||   4.9% |   489.3 |    76.7 | 13.6% |MPI_ALLREDUCE
||   1.5% |   145.4 |   314.6 | 68.4% |MPI_REDUCE
||============================================================
|  17.8% | 1,766.8 |      -- |    -- |ETC
||------------------------------------------------------------
||   9.9% |   983.9 |     8.1 |  0.8% |__DEALLOCATE
||   6.6% |   652.3 |    93.7 | 12.6% |gotoblas_dgemv_n_sandybridge
|============================================================
```

Figure 19. Raw profiling data for ParaFEM/CGPACK MPI/coarray miniapp `cgca_gcupdn` and `cgca_pfem_map` at 7200 cores.

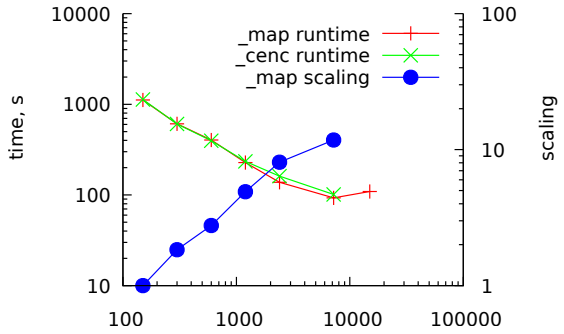

Figure 20. Runtimes and scaling for ParaFEM/CGPACK MPI/coarray miniapp with `cgca_pfem_map` and `cgca_pfem_cenc`.

at very high core counts due to memory limitations.

In addition, all variables have to be recast in the same real kind before writing to `tmp` array. This is because `CO_SUM` and `CO_MAX` work only with numeric types.

Finally we note that `CO_SUM` and `CO_MAX` allow non-coarray `SOURCE` argument. This is an interesting feature of coarray design.

Figs. 18 and 19 show that although the total percentage of time spent on `cgca_clvgp` has slightly increased, the time corresponding to the user group functions has been reduced. As a result, this miniapp manages to improve the performance achieved by miniapps running `cgca_pfem_cenc`.

Since `cgca_pfem_map` or `cgca_pfem_cenc` are

```
|  71.4% | 14,649.9 |    -- |    -- |USER
||-----------------------------------------------------------
||  38.7% |  7,950.6 | 913.4 | 10.3% |cgca_gcupda$cgca_m3clvg_
||  24.1% |  4,951.2 | 940.8 | 16.0% |cgca_clvgp$cgca_m3clvg_
||   3.1% |    638.0 |  70.0 |  9.9% |cgca_pfem_cenc$cgca_m3pfem_
||   1.8% |    367.5 | 578.5 | 61.2% |cgca_hxi$cgca_m2hx_
||   1.7% |    346.0 | 196.0 | 36.2% |cgca_clvgn$cgca_m3clvg_
||===========================================================
```

Figure 21. Sampling ParaFEM/CGPACK MPI/coarray miniapp with `cgca_gcupda` on 7200 cores.

called only once during the execution of the miniapp, only a minor overall performance improvement is expected. This is shown in Fig. 20. The miniapp implementing `cgca_pfem_map` shows some performance improvement only from around 1000 cores, where the overhead of remote read statements becomes noticeable.

A. CrayPat issues

During the course of this investigation, a small number of issues have been identified with CrayPat. Theses issues have been already submitted to Cray developers for further investigation.

Tracing ParaFEM/CGPACK MPI/coarray has been reporting an inconsistent percentage of time for some user routines which were highlighted in sampling experiments. Figs. 21 and 22 illustrate this effect on subroutine `cgca_gcupda`. Although the sampling report indicates that this subroutine is the most time consuming user function, this routine is not present at all in the tracing report, even when `cgca_gcupda` was specifically traced. However, it the miniapp, `cgca_gcupda` and `cgca_hxi` are called exactly the same number of times. Indeed a call to `cgca_gcupd` is immediately followed by a call to `cgca_hxi` in `cgca_clvgp`. The key fragment is shown below.

```
module subroutine cgca_clvgp( coarray,  &
         ... gcus, ... )
 integer, allocatable, intent(inout) :: &
   coarray(:,:,:,:)[:,:,:]
 procedure( gcupd_abstract ) :: gcus
  :
end subroutine cgca_clvgp

module procedure cgca_clvgp
 :
! update all local GC arrays using
! the given subroutine
 call gcus( periodicbc )
! halo exchange after a cleavage
! propagation step
 call cgca_hxi( coarray )
 :
end procedure cgca_clvgp
```

where `cgca_gcupda` is passed for dummy `gcus`.

```
|  29.7% |  99.743118 |       -- |    -- |  5,226,813.1 |USER
||-------------------------------------------------------------------------------
||  17.4% |  58.326659 | 36.082315 | 38.2% |          5.0 |cgca_clvgp$cgca_m3clvg_
||   5.6% |  18.876152 |  5.062089 | 21.1% |          1.0 |cgca_pfem_cenc$cgca_m3pfem_
||   3.3% |  11.145318 | 15.328335 | 57.9% |          1.0 |xx14_
||   1.7% |   5.705317 |  8.788733 | 60.6% |  5,224,771.1 |cgca_clvgn$cgca_m3clvg_
||   1.7% |   5.689672 |  1.910819 | 25.1% |      2,035.0 |cgca_hxi$cgca_m2hx_
||===============================================================================
```

Figure 22. Tracing ParaFEM/CGPACK MPI/coarray miniapp with `cgca_gcupda` on 7200 cores.

```
CrayPat/X:  Version 6.2.2 Revision 13378 (xf 13240)  11/20/14 14:32:58
Number of PEs (MPI ranks):   480

Numbers of PEs per Node:      24  PEs on each of  20  Nodes

Numbers of Threads per PE:     3

Number of Cores per Socket:   12
Execution start time:  Thu Mar  3 13:40:17 2016
System name and speed:  tdsmom 2701 MHz
```

Figure 23. Incorrect number of threads indentified by CrayPAT in a tracing experiment of ParaFEM/CGPACK MPI/coarray miniapp with `cgca_gcupda`.

Another issue is the number of threads reported by CrayPat when profiling our different ParaFEM/CGPACK MPI/coarray miniapps. Although profiling has always been carried out using a single thread, CrayPat indicates otherwise. An example is shown in Fig. 23 where in this case CrayPat reports that the miniapp has been run with 3 threads.

B. Cray Reveal

Fortran 2008 standard says that allocatable coarrays must be allocated with the same dimensions and codimensions on all images. Moreover this allocation must be done at the same time on all images, because coarray allocation involves synchronisation between all images. When more flexible data structures are desired, e.g. coarray arrays of different length, as in this work, the only solution currently is to use coarray variables of derived type with allocatable components. Such variables provide excellent flexibility and potential for extension. However, the price for such flexibility is poor optimisation of remote operations with such variables. This problem was highlighted with Cray Reveal.

The following fragment is from the CGPACK/ParaFEM interface module `cgca_m3pfem`, subroutine `cgca_pfem_yum`, which updates the FE Young's modulus based on the accummulated fracture.

```
! how many elements
ndims=size(      &
 centroid_tmp[ img_curr ] % r, dim=1 )
nelements=size( &
 centroid_tmp[ img_curr ] % r, dim=2 )
! use a temp array to pull
! all centroids data in one call
allocate( tmp( ndims, nelements ) )
tmp = centroid_tmp[ img_curr ] % r
```

where `img_curr` is the neighbouring image number.

Here an attempt is made to minimise the number of remote operations by reading the whole array from another image into a local array `tmp`. However, the size of the array on the remote image is not known. Hence the first two remote operations are reading just the two dimensions of the array. Then the local array is allocated accordingly. Finally the remote array can be read into the local array.

The last statement in the above fragment is at line 571. It attracted five separate messages from Cray Reveal, which are reproduced below.

- "A loop starting at line 571 is flat (contains no external calls)."
- "A loop starting at line 571 was not vectorized because it contains a definition of reference to a coarray variable on line 571."
- "A loop starting at line 571 was unrolled 8 times."
- "An implicit non-blocking operation was used for this statement."
- "The coarray assignment in the loop starting at line 571 was not replaced by a block remote data transfer operation because it is not recognizable as a supported pattern."

More information is obtained about the last message with `explain ftn-6239`: "The list of patterns recognizable by the compiler is as follows: memory copy, get, or put with loop invariant strides." This indicates that coarray support in CCE is still maturing and there is scope for further optimisation.

IV. Future work

The focus of this work was optimisation of user routines. Another important area with a significant scope for optimisation is synchronisation.

The Fortran standard [1] contains very strict rules on coarray synchronisation. The rules are designed to ensure that a standard conforming Fortran coarray program will neither deadlock nor suffer from race conditions. The price of this guarantee of coarray integrity is potentially excessive synchronisation. This is particularly pertinent to a coarray library such as CGPACK, where multiple synchronisation paradigms are possible.

At one extreme, all synchronisation is removed from the library routines and is left as the responsibility of the library user. This approach might be the most efficient but is definitely the most error-prone. At the other extreme synchronisation is used inside all, or most, library routines. This approach is most likely to guarantee coarray integrity but is very inefficient.

The search of an efficient coarray synchronisation paradigm, for the CGPACK library and for ParaFEM/CGPACK miniapps, which will ensure full standard conformance and coarray integrity, is left for future work. We are keen to test new coarray features of TS 18508, as soon as these are available on Cray. In particular, coarray `EVENTS` might be useful for implementing more flexible synchronisation between images.

V. Conclusion

Successful usage of Fortran coarray and MPI parallel libraries on ARCHER (Cray XC30 system), to create what may be the first multi-scale cellular automata finite element (CAFE) "miniapp" for proof-of-concept modelling is novel. It is uniquely enabled on a Cray system. CrayPAT sampling and tracing led to identification of optimisation hotspots. Replacement of all-to-all algorithms by the nearest neighbour algorithms and coarray collectives increased the scaling limit by a factor of 3. The hybrid MPI/coarray miniapps stress tested CrayPAT tools and identified several issues in these tools. This highlights the usefulness of MPI/coarray miniapps to Cray developers, because these small driver programs represent complexity of real engineering codes for structural integrity and solid mechanics calculations. The libraries and miniapps are BSD-licensed, so they can be used by computer scientists and hardware vendors to test compilers, performance monitoring applications, etc.

Acknowledgment

This work was funded under the embedded CSE programme of the ARCHER UK National Supercomputing Service (http://www.archer.ac.uk)

References

[1] ISO/IEC 1539-1:2010, *Fortran – Part 1: Base language, International Standard*, 2010.

[2] A. Shterenlikht, L. Margetts, L. Cebamanos, and D. Henty, "Fortran 2008 coarrays," *ACM Fortran Forum*, vol. 34, pp. 10–30, 2015. [Online]. Available: http://eis.bris.ac.uk/~mexas/pub/2015acmff.pdf

[3] A. Shterenlikht, "Writing parallel programs with Fortran 2008 coarrays," *Institute of Physics, Computational Physics Group Newsletter*, pp. 12–20, Summer 2014. [Online]. Available: http://www.iop.org/activity/groups/subject/comp/news/file_64129.pdf

[4] G. Mozdzynski, M. Hamrud, and N. Wedi, "A partitioned global address space implementation of the European centre for medium range weather forecasts integrated forecasting system," *Int. J. High Perf. Comp. Appl.*, vol. 29, pp. 261–273, 2015.

[5] V. Kouznetsova, W. A. M. Brekelmans, and F. P. T. Baaijens, "An approach to micro-macro modeling of heterogeneous materials," *Comp. Mech.*, vol. 27, pp. 37–48, 2001.

[6] I. M. Smith, D. V. Griffiths, and L. Margetts, *Programming the Finite Element Method*, 5th ed. Wiley, 2014.

[7] J. Phillips, A. Shterenlikht, and M. J. Pavier, "Cellular automata modelling of nano-crystalline instability," in *Proc. 20th UK ACME Conf. 27-28 March 2012, The University of Manchester, UK*, 2012. [Online]. Available: http://eis.bris.ac.uk/~mexas/pub/2012_ACME.pdf

[8] A. Shterenlikht and L. Margetts, "Three-dimensional cellular automata modelling of cleavage propagation across crystal boundaries in polycrystalline microstructures," *Proc. Roy. Soc. A*, vol. 471, p. 20150039, 2015. [Online]. Available: http://eis.bris.ac.uk/~mexas/pub/2015prsa.pdf

[9] I. M. Smith and L. Margetts, "The convergence variability of parallel iterative solvers," *Eng. Computations*, vol. 23, pp. 154–165, 2006.

[10] L. M. Evans, L. Margetts, V. Casalegno, L. M. Lever, J. Bushell, T. Lowe, A. Wallwork, P. Young, A. Lindemann, M. Schmidt, and P. M. Mummery, "Transient thermal finite element analysis of CFC-Cu ITER monoblock using x-ray tomography data," *Fusion Eng. Des.*, vol. 100, pp. 100–111, 2015.

[11] J. D. Arregui-Mena, L. Margetts, D. V. Griffiths, L. Lever, G. Hall, and P. M. Mummery, "Spatial variability in the coefficient of thermal expansion induces pre-service stresses in computer models of virgin gilsocarbon bricks," *J. Nuclear Materials*, vol. 465, pp. 793–804, 2015.

[12] S. D. Rawson, L. Margetts, J. K. F. Wong, and S. H. Cartmell, "Sutured tendon repair; a multi-scale finite element model," *Biomechanics Modelling Mechanobiology*, vol. 14, pp. 123–133, 2015.

[13] W. I. Sellers, L. Margetts, R. A. Coria, and P. L. Manning, "March of the titans: The locomotor capabilities of sauropod dinosaurs," *Plos one*, vol. 8, p. e78733, 2013.

[14] A. Shterenlikht, "Fortran coarray library for 3D cellular automata microstructure simulation," in *Proc. 7th PGAS Conf., 3-4 October 2013, Edinburgh, Scotland, UK*, M. Weiland, A. Jackson, and N. Johnson, Eds. The University of Edinburgh, 2014, pp. 16–24. [Online]. Available: http://www.pgas2013.org.uk/sites/default/files/pgas2013proceedings.pdf

[15] A. Fanfarillo, "Parallel programming techniques for heterogeneous exascale computing platforms," Ph.D. dissertation, Università degli Studi di Roma Tor Vergata, 2016. [Online]. Available: http://www.opencoarrays.org/uploads/6/9/7/4/69747895/fanfarillophd.pdf

[16] A. Fanfarillo, T. Burnus, S. Filippone, V. Cardellini, D. Nagle, and D. Rouson, "OpenCoarrays: open-source transport layers supporting coarray Fortran compilers," in *Proc. PGAS 2014, Eugene, Oregon, USA*, 2014. [Online]. Available: http://www.opencoarrays.org/uploads/6/9/7/4/69747895/pgas14_submission_7-2.pdf

[17] A. Shterenlikht and I. C. Howard, "The CAFE model of fracture – application to a TMCR steel," *Fatigue Fract. Eng. Mater. Struct.*, vol. 29, pp. 770–787, 2006.

[18] S. Das, A. Shterenlikht, I. C. Howard, and E. J. Palmiere, "A general method for coupling microstructural response with structural performance," *Proc. Roy. Soc. A*, vol. 462, pp. 2085–2096, 2006.

[19] S. J. Wu, C. L. Davis, A. Shterenlikht, and I. C. Howard, "Modeling the ductile-brittle transition behavior in thermo-mechanically contolled rolled steels," *Met. Mater. Trans. A*, vol. 36, pp. 989–997, 2005.

[20] A. Vishnu, S. Song, A. Marquez, K. Barker, D. Kerbyson, K. Cameron, and P. Balaji, "Designing energy efficient communication runtime systems: a view from PGAS models," *J. Supercomputing*, vol. 63, pp. 691–709, 2013.

[21] C. Teijeiro, G. Sutmann, G. L. Taboada, and J. Touriño, "Parallel Brownian dynamics simulations with the message-passing and PGAS programming models," *Comp. Physics Comms*, vol. 184, pp. 1191–1202, 2013.

[22] ISO/IEC JTC1/SC22/WG5 N2074, *TS 18508 Additional Parallel Features in Fortran*, 2015.

[23] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, "Improving performance via mini-applications," Sandia National Laboratories, Albuquerque, New Mexico 87185 and Livermore, California 94550, Tech. Rep. SAND2009-5574, 2009. [Online]. Available: https://mantevo.org/MantevoOverview.pdf