Open source exascale multi-scale framework for the UK solid mechanics community ARCHER eCSE05-05 Report

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

28-FEB-2017

***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, defence 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 and TAU tools have been used for sampling and tracing analysis of the miniapps. Two routines with all-to-all communication structures have been identified as 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. In addition the code has been ported to Intel and GCC/OpenCoarrays platforms, which dramatically increased the number of users.**

***Keywords*-miniapps; Fortran; coarrays; MPI; cellular automata; CGPACK; finite elements; ParaFEM; CrayPAT; TAU; ARCHER;**

I. Introduction

Solid mechanics community is very poorly represented on ARCHER. This is partly due to lack of scalable codes suitable for complex solid mechanics problems: large scale geometrical or material non-linearity, extensive plastic flow, residual stress analysis or fracture. Many researchers are locked into using proprietary FE software limited to workstations, e.g. large shared memory 16-core computers. Some commercial FE codes scale to 64 or even to 1024 cores, according to the vendor literature. However, the vendors typically insist on charging a licence fee for each core. This makes analysis at scale prohibitively expensive, even with academic licences. Hence there is a big appetite for scalable, flexible, verified and feature reach simulation codes among the solid mechanics researchers.

This project has delivered a scalable tool that can satisfy this need, initially for fracture problems. A multi-scale engineering modelling framework for deformation and fracture in heterogeneous materials, e.g. polycrystals, porous graphite, concrete, metal matrix or carbon fibre composites has been demonstrated to have a good potential for HPC systems, such Tier-1 ARCHER, Tier-2 Hartree centre systems and smaller University and departmental systems.

This work delivered specific benefits to ARCHER users. The majority of the ARCHER community choose Fortran for their projects, due to many factors, some of which are performance, availability of optimising compilers and of international standards, and vast codebase including very high quality libraries. Coarrays are a standard Fortran 2008 feature. However, their uptake has been limited so far due to poor compiler support and lack of successful examples of coarray programs at scale. This project has delivered a scalable coarray/MPI modelling framework, which will serve as one such example. This, together with dissemination via Cray 2016 user group meeting [1], Supercomputing 2016 [2] and PARENG 2017 [3] conferences, might attract existing ARCHER users to try coarrays in their programs to improve scaling.

In addition to ARCHER, the multi-scale framework has been ported to Intel and GCC/OpenCoarray platforms which dramatically increased the number of potential users. The added portability opened many new optimisation possibilities because coarray implementations differ substantially across vendors. This also increases vendor competition which is always good news for users, including the ARCHER users.

Multiple bugs in Cray software were uncovered and fixed during this project, which resulted in much improved software environment for all ARCHER users.

Extensive profiling work with the TAU toolkit resulted in substantial improvement of TAU support for coarrays. As a result of this work, at present TAU is the most powerful tool for coarray/MPI hybrid programs on non-Cray systems. As a side benefit, a new collaboration between our team and the TAU team has been established.

II. ParaFEM/CGPACK (MPI/coarray) miniapps

We have developed a two-way hierarchical concurrent multi-scale cellular automata (CA) finite element (FE) fracture

framework (CAFE). FE represent the structural level using the ParaFEM MPI library. CA represent the microstructure using the using CGPACK Fortran coarray library. Both ParaFEM and CGPACK are being actively developed, including contributions from the UK Software Sustainability Institute (SSI) `http://software.ac.uk` [4]. Both ParaFEM and CGPACK libraries are distributed under BSD license.

ParaFEM is a highly scalable and portable MPI FE library written in Fortran 2003, `http://parafem.org.uk` [5]. It is the latest extension of the sequential FE libraries originally written by Professor Ian Smith and first published in the 1980s [6]. Interestingly, at that time, they were distributed as open source on tape by NAG Ltd. The software comprises modules, subroutines, functions and around 70 example mini-apps [7]. The mini-apps are typically 2-4 pages long and are used to solve a variety of common engineering problems. The mini-app philosophy enables customisation by engineers, a feature that has enabled the work presented herein to be carried out with a reasonable amount of software development effort.

The parallelisation strategy adopted in ParaFEM involves working element-by-element at each stage of the finite element process, including building element stiffness matrices, solving the system of equations and recovering stress values (post-processing). No global matrix is ever assembled and so domain decomposition is avoided. Each MPI process is allocated an equal number of finite elements, balancing both computational load and memory usage. Parallel element-by-element versions of different iterative solvers are used for different problem types. These work in essentially the same way as their sequential counterparts [8], with the only difference being the need to pass messages between MPI processes when operating on distributed data structures.

The approach has been successful in solving a variety of problem types, from nonlinear material behaviour [9] to coupled systems involving multiphysics, such as Biot consolidation and magneto-hydrodynamics [10]. The software has led to scientific advances in a range of disciplines such as Nuclear Engineering [11], [12], Biomechanics [13], [14], Geomechanics [15] and Palaeontology [16].

CGPACK is a scalable cellular automata library written in Fortran 2008 with extensive use of coarrays `http://cgpack.sf.net`. Work on CGPACK started in 2013 [17] on HECToR. The CA space is implemented as a 4D allocatable array coarray, with a 3D coindex set:

```
integer, allocatable ::      &
  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.

Although the idea of a multi-scale CAFE model is not new, [18], [19], [20], the current CAFE framework was designed specifically for HPC systems [21], [22]. 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.

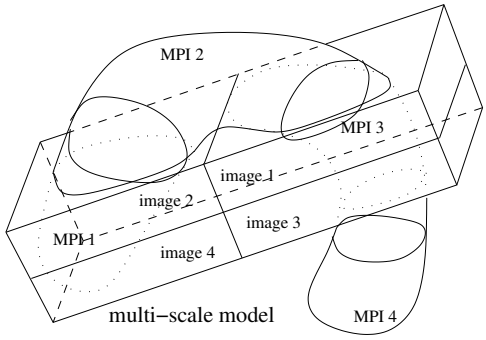

Fig. 1. Possible partition of the multi-scale model on 4 PEs.

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.

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. 1. 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.

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

```
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) )
```

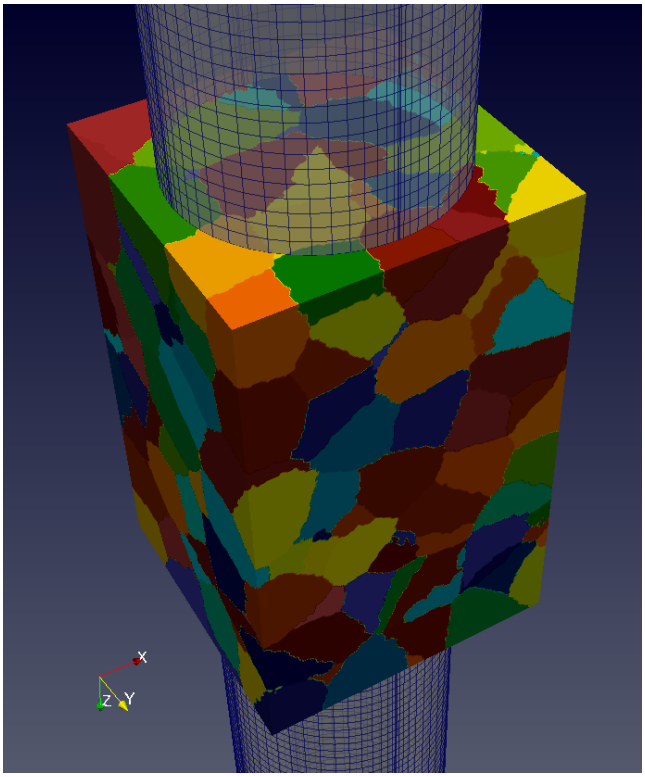

Fig. 2. CAFE modelling of a steel cylinder under tension showing the CA microstructure. The FE cylinder mesh is semi-transparent for clarity.

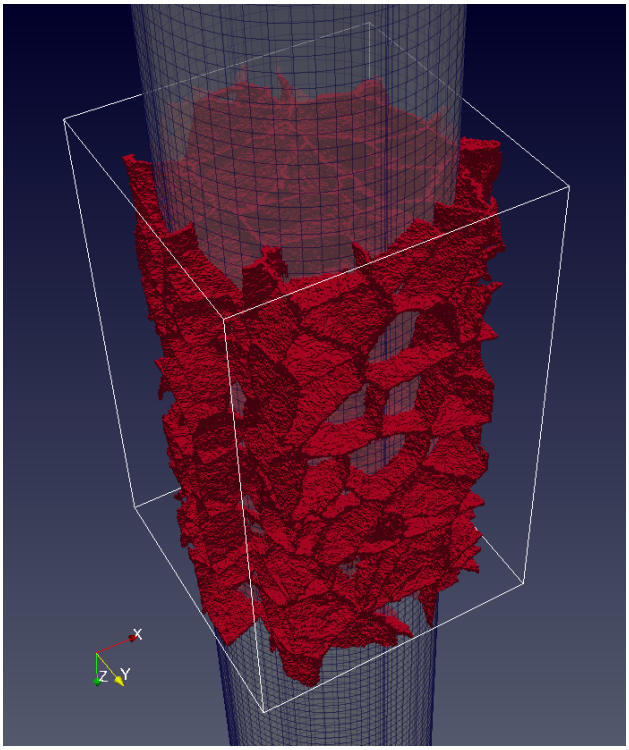

Fig. 3. CA microstructure grain boundaries, with inactive cells removed.

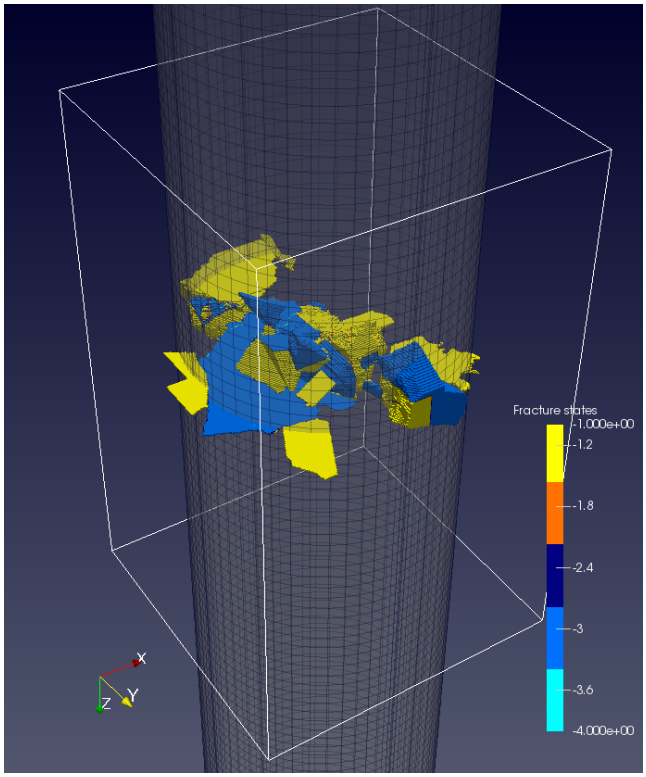

Fig. 4. Micro-cracks merging into a macro-crack.

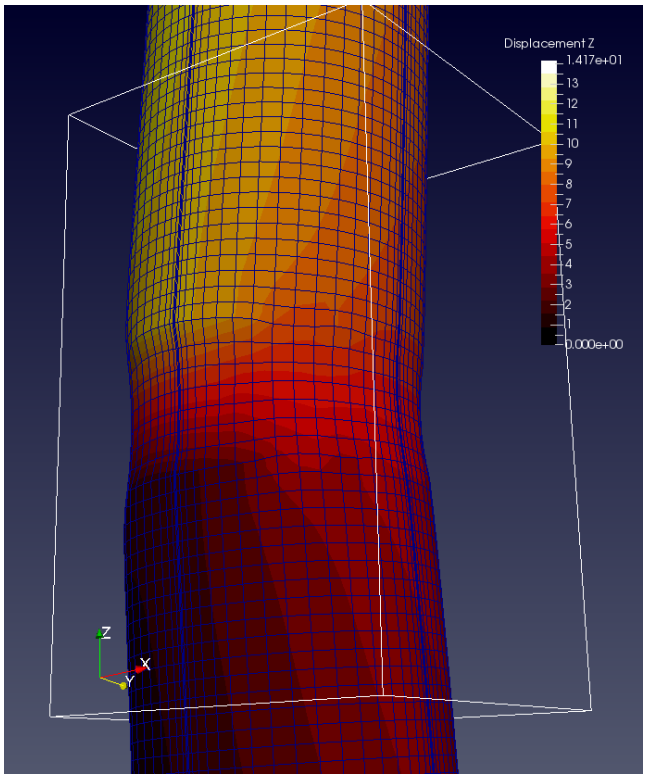

Fig. 5. The distorted FE mesh at the end of the CAFE simulation, with the axial displacement contours.

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 [23] 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 and on GCC/OpenCoarray platform as extension to the standard [24]. 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.

Several driver programs (miniapps) [25] were constructed using both ParaFEM and CGPACK libraries. CAFE cleavage simulation in a rod under tension is shown in Figs. 2 to 5. The FE model is a 140mm long mild steel cylinder of 10mm diameter and 100mm gauge length. One end of the cylinder is constrained and an axial force is applied to the other end. The FE elastic properties are the Young's modulus of 200GPa and the Poisson's ratio of 0.3. The CA block is positioned centrally on the cylinder, see Fig. 2.

Fig. 2 shows the polycrystalline microstructure layer of the `space` coarray. The colour of each grain (single crystal) encodes its rotation tensor. Fig. 3 shows the grain boundaries in the fracture layer of the `space` coarray.

Fig. 4 shows the macro-crack emerging from linking cracks on preferential cleavage planes in individual crystals. There are 4 cell fracture states in this model: -1, -2, -3 and -4. -1 (yellow) denotes crack flanks on 100 planes. -3 (light blue) denotes crack flanks on 110 planes. Both yellow and light blue regions are clearly visible in Fig. 4. -2 (dark blue) denotes crack edges on 100 planes. -4 (cyan) denotes crack edges on 110 planes.

Fig. 5 shows the FE mesh at the end of the simulation, when the macroscopic cleavage crack has propagated across nearly the whole of the cross section. The contour plot of the axial displacement is superimposed over the mesh. Note a high displacement gradient across the crack.

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 [26]. 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`.

III. Profiling and Optimisation (WP1 and WP2)

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. 6 and 7), 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. 8. It

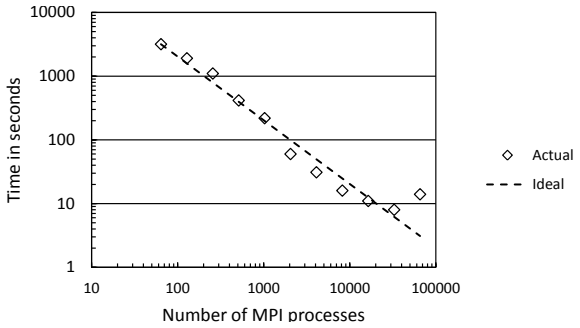

Fig. 6. ParaFEM scaling for a 3D transient flow explicit analysis. Reproduced from [7].

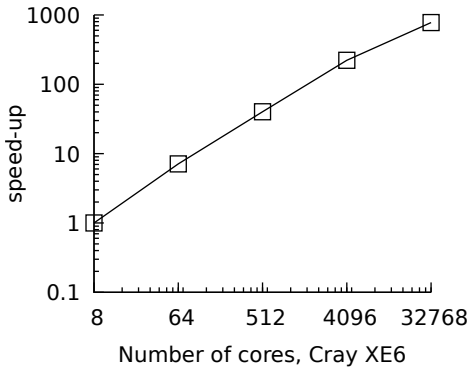

Fig. 7. CGPACK 3D solidification scaling with different synchronisation methods.

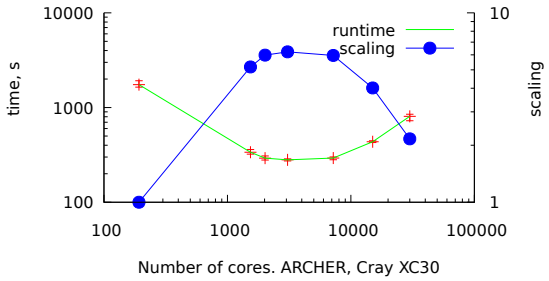

Fig. 8. ParaFEM/CGPACK MPI/coarray miniapp scaling on ARCHER XC30 for a 3D problem with 1M FE and 800M CA cells.

is clear that this miniapp scales well up to 2000 cores from where the scalability drops dramatically.

As shown in Figs. 9 and 10, 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.

```
integer :: gcupd(100,3)[*], rndint, j,&
  img, gcupd_local(100,3)
real :: rnd
:
call random_number( rnd )
```

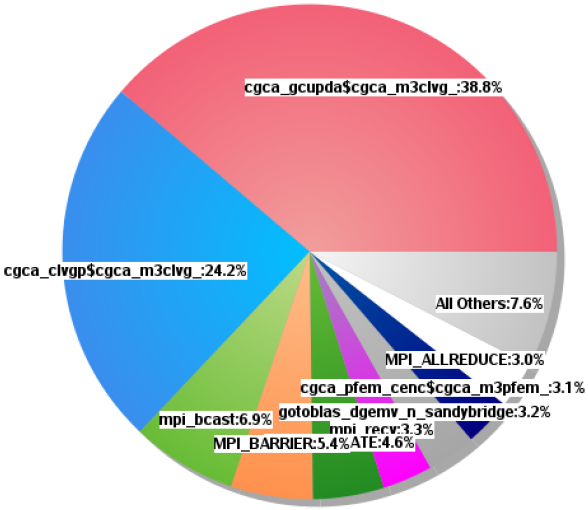

Fig. 9. Profiling function distribution for ParaFEM/CGPACK MPI/coarray miniapp with all-to-all routine `cgca_gcupda` at 7200 cores.

```
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
|==========================================================
```

Fig. 10. Raw profiling data for ParaFEM/CGPACK MPI/coarray miniapp with all-to-all routine `cgca_gcupda` at 7200 cores.

```
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]
  :
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
```

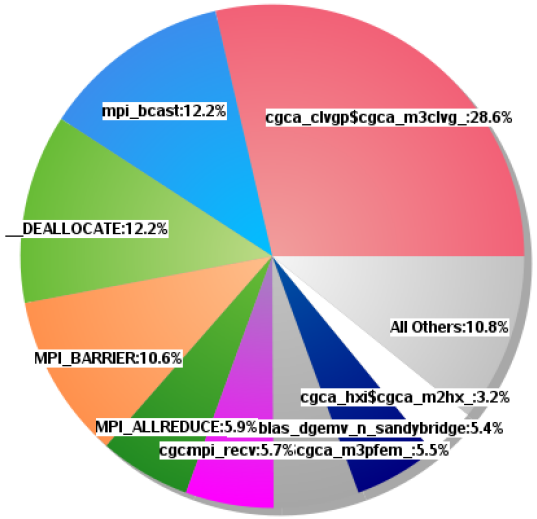

Fig. 11. Raw profiling data for ParaFEM/CGPACK MPI/coarray miniapp with the neareast neighbour routine `cgca_gcupdn` at 7200 cores.

```
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
|==========================================================
```

Fig. 12. Profiling function distribution for ParaFEM/CGPACK MPI/coarray miniapp with the neareast neighbour routine `cgca_gcupdn` at 7200 cores.

```
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

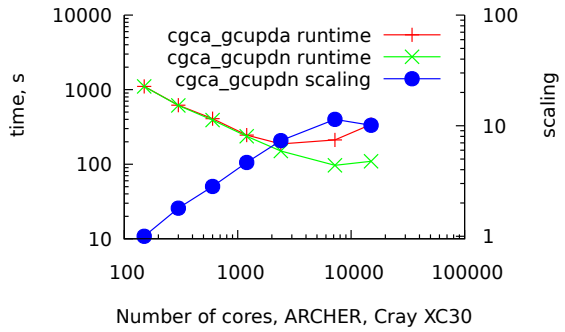

Fig. 13. Runtimes and scaling for ParaFEM/CGPACK MPI/coarray miniapp with the nearest neighbour, `cgca_gcupdn`, and all-to-all, `cgca_gcupda`, algorithms.

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 nearest neighbourgh algorithm clearly demonstrates a considerable reduction in the number of remote reads. This can be seen on Figs. 11 and 12 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. 13 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 from TS 18508 [23]. 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.

```
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 )
```

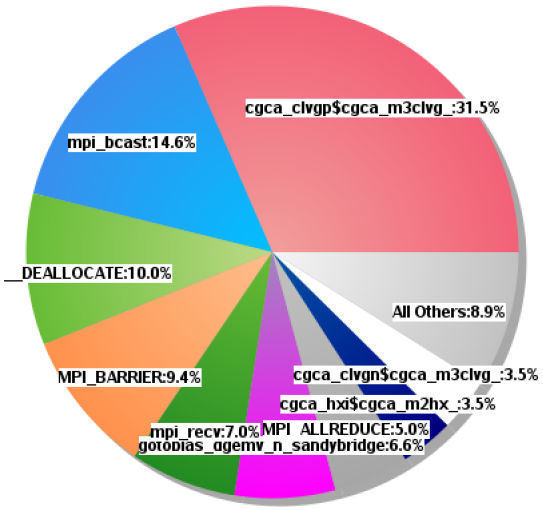

Fig. 14. Profiling function distribution with `cgca_gcupdn` and `cgca_pfem_map` at 7200 cores.

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 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.

Figs. 14 and 15 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 called only once during the execution of the miniapp, only a minor overall performance improvement is expected. This is shown in Fig. 16. 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.

```
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
|================================================================
```

Fig. 15. Raw profiling data for ParaFEM/CGPACK MPI/coarray miniapp `cgca_gcupdn` and `cgca_pfem_map` at 7200 cores.

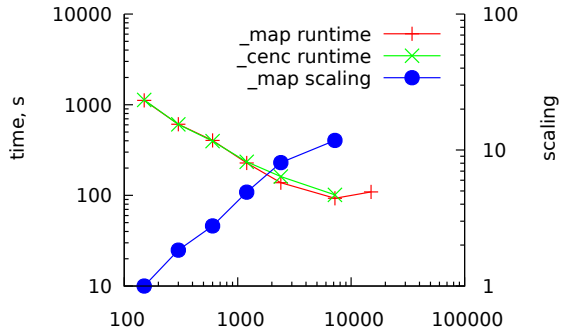

Fig. 16. Runtimes and scaling for ParaFEM/CGPACK MPI/coarray miniapp with `cgca_pfem_map` and `cgca_pfem_cenc`.

Tracing ParaFEM/CGPACK MPI/coarray has been reporting an inconsistent percentage of time for some user routines which were highlighted in sampling experiments. Figs. 17 and 18 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
```

```
|  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_
||===============================================================
```

Fig. 17. Sampling ParaFEM/CGPACK MPI/coarray miniapp with `cgca_gcupda` on 7200 cores.

```
|  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_
||=========================================================================
```

Fig. 18. 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
```

Fig. 19. Incorrect number of threads indentified by CrayPAT in a tracing experiment of ParaFEM/CGPACK MPI/coarray miniapp with `cgca_gcupda`.

```
  :
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`.

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. 19 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.

C. Profiling and tracing with TAU

TAU (Tuning and Analysis Utilities) `https://www.cs.uoregon.edu/research/tau` [27], is a popular open source set of tools for performance analysis, particularly on HPC systems. Recently TAU was shown to support coarray programs [28], [29]. We used TAU 2.25.2 to profile and trace

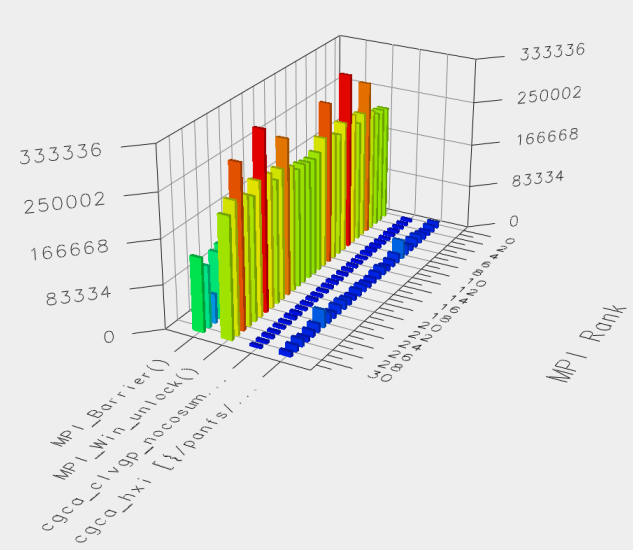

Fig. 20. Profile of CGPACK program `testABW`, showing only functions accounting for more than 1% of total exclusive time.

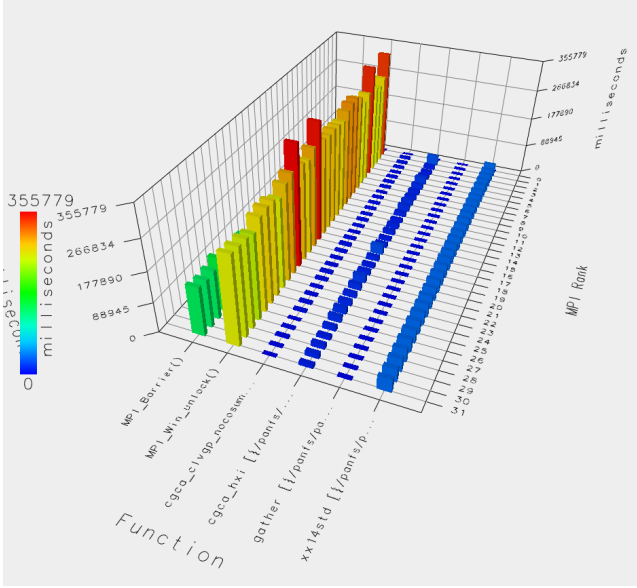

Fig. 21. 3D profiling bar chart of coarray/MPI miniapp `xx14std`.

coarray/MPI miniapps on non-Cray linux HPC platform using the Intel Fortran compiler and the Intel MPI library.

TAU is often used together with the Program Database Toolkit (PDT), `https://www.cs.uoregon.edu/research/pdt`, which is a framework for analysing source code. However, PDT 3.22 does not yet support coarrays. Hence compiler based instrumentation was used in this work, which is enabled with TAU flag `-optCompInst`.

The University of Bristol BlueCrystal phase 3 system was used for this work `http://www.acrc.bris.ac.uk`. Each node had a single 16-core 2.6 GHz SandyBridge CPU and 64GB RAM. Intel Cluster Studio XE, version 16.0.2, was used which uses the Intel MPI library 4.1.0. TAU was configured with

```
-mpi -c++=mpiicpc -cc=mpiicc \
```

```
-fortran=mpiifort
```

Exclusive TAU times were measured and reported in all cases.

Jumpshot-4, developed by the Argonne National Lab (ANL) `http://www.mcs.anl.gov/research/projects/perfvis/software/viewers/index.htm`, was used to view TAU traces.

Figs. 20 and 21 show profiling data of 2 CGPACK/ParaFEM coarray/MPI miniapps on 2 16-core nodes with `-O2` optimisation flag. The Intel implementation of coarrays uses MPI-2 RMA, with `MPI_Win_unlock` heavily dominating run times. Although MPI-2 RMA is supposed to be an optimal mapping of PGAS coarray communications, these results show that performance critically depends on the quality of optimisation, thus indicating a potential for optimisation of (MPI based) coarray remote operations, such as that from Intel.

D. Opportunities for thread parallelisation

Many CA routines contain triple nested loops over all cells on an image. An example below is taken from `cgca_clvgp`, the cleavage propagation routine. Each iteration of the `main` loop all cells in the CA on an image are processed.

```
main: do iter = 1,N
 do x3 = lbr(3), ubr(3)
 do x2 = lbr(2), ubr(2)
 do x1 = lbr(1), ubr(1)
  live: if ...
   ! scan only through undamaged cells
   call cgca_clvgn( clvgflag )
   if ( clvgflag ) call sub( space )
  end if live
 end do
 end do
 end do
 call co_sum( clvgglob )
 sync all
 call cgca_hxi( space )
 sync all
 call cgca_dacf( space )
end do main
```

Such nested loops might present good opportunities for thread parallelisation with either OpenMP or OpenACC (e.g. on GPUs or Xeon Phi), although the use of underpopulated nodes might be required. Fortran 2008 new intrinsic **`DO CONCURRENT`** should also be explored, although at present its performance portability is inferior to OpenMP. Recently, ParaFEM has been ported to Xeon Phi [30]. In order to make best use of the Xeon Phi architecture, the code needed some rewriting to use a mixed OpenMP/MPI parallelisation strategy. On standard x86 multicore processors, the addition of OpenMP provides no benefit. However, on the Xeon Phi, OpenMP using 4 threads per core provides an additional 4-fold speed-up in run times. Porting of CGPACK to Xeon Phi is planned for the future.

IV. Standard compliance and fault tolerance (WP3 and WP4)

The standard compliance (WP3) has been fully achieved. The framework has been ported to GCC/OpenCoarrays and Intel compilers with the use of only Fortran 2008 standard features. As these platforms are widely accessible, this work dramatically increased the number of users of the code, as measured by the number of downloads of CGPACK and by the number of registered ParaFEM users.

A separate grant from the Software Sustainability Institute (SSI, `http://software.ac.uk` [4]) funded ongoing work to implement an autotools build procedure, which will further improve portability and usability.

The fault tolerance objective was not achieved due to poor compiler support. Only a single feature, `FAILED_IMAGES`, has become available towards the end of this project, and is currently supported only by GCC/OpenCoarrays. Fortran 2015 text describing facilities for dealing with failed images have been finalised in JUN-2016 and it is expected that the fault tolerant features will gradually become available over the next 1-2 years in at least 3 compilers: Cray, Intel and GCC/OpenCoarrays. We will continue monitoring compiler support for these features and implement them in the multi-scale framework as they become available.

V. MPI/IO (WP5)

NetCDF and HDF5 filters have been written for cellular automata data structures. However, the performance of NetCDF and HDF5 writers has been disappointing compared with direct use of MPI/IO. This issue is currently being investigated.

Both NetCDF and HDF5 have been implemented in CGPACK. However, as Fig. 22 shows, at present maximum NetCDF IO rates are only about 1.2 GB/s [2], which is significantly lower than direct use of MPI/IO that gives rates of about 8 GB/s.

The xdmf wrapper scripts for CGPACK have been updated to support HDF5 data format.

With ParaFEM, a decision was made to implement a standard engineering binary format supported by the visualisation tool Paraview instead of NetCDF and HDF5 and use MPI/IO directly. There were two main reasons for this decision: (i) uptake is hindered by usability and our end users will be more comfortable using a standard engineering format rather than having the additional barrier of learning about NetCDF and HDF5 and (ii) the performance improvement using MPI/IO looks more promising than these libraries. We had an issue getting Paraview to read binary files written by ParaFEM. After significant debugging effort, we found that Paraview would not read binary files written by Fortran, only by C or C++. Our workaround was to use C data types in ParaFEM (provided by the ISO C Binding in Fortran 2003 and later) and write out C binary files from the Fortran program. So for ParaFEM, the eCSE has provided us with a big push in the right direction, but we ran out of resource before completion. We will complete this objective ourselves.

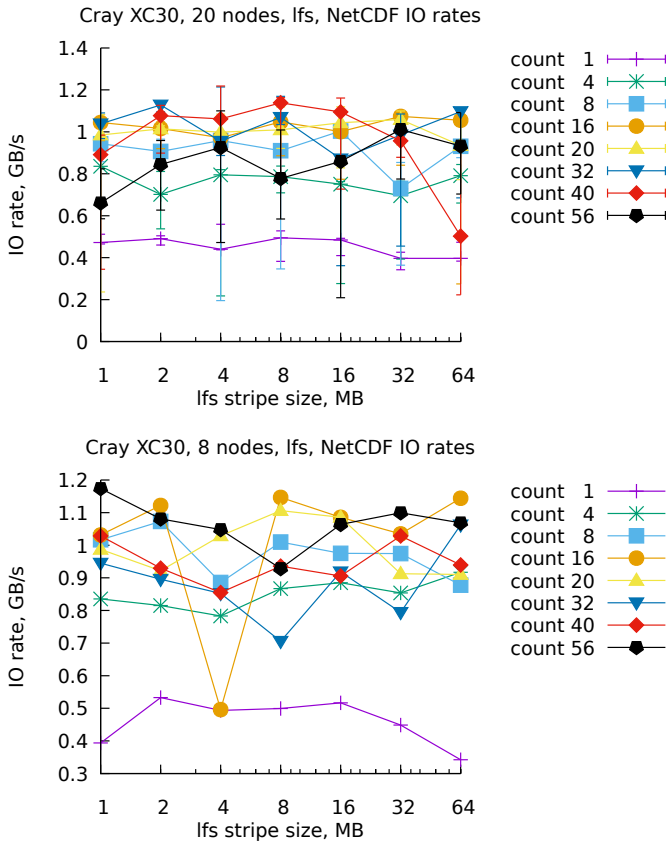

Fig. 22. CGPACK IO rates showing the influence of the Lustre file system stripe size and stripe count settings

VI. Conclusion

The technical objectives of this project have been achieved. The scaling limit for hybrid coarray/MPI miniapps has been increased by over 3 times, from 2k to 7k cores. This enabled new high resolution deformation and fracture results to be achieved. TAU profiling and tracing tools were proven useful on Intel platforms in exposing poor optimisation of coarray operations in Intel 16 Fortran compiler. The addition of NetCDF and HDF5 routines improved the usability of CGPACK. Together with successful porting of the multi-scale framework to GCC/OpenCoarrays platform, this dramatically increased the usage of the model. The hybrid MPI/coarray miniapps stress tested CrayPAT tools and identified several issues in these tools.

Acknowledgement

This work was funded under the embedded CSE programme of the ARCHER UK National Supercomputing Service (`http://www.archer.ac.uk`). This work was carried out using the computational facilities of the Advanced Computing Research Centre, The University of Bristol, UK (`http://www.bris.ac.uk/acrc`). We would like to acknowledge the assistance of the Software Sustainability Institute, UK (`https://www.software.ac.uk`).

References

[1] L. Cebamanos, A. Shterenlikht, D. Arregui-Mena, and L. Margetts, "Scaling hybid coarray/mpi miniapps on archer," in *Cray User Group 2016 meeting (CUG2016), London, 8-12-MAY-2016*, 2016. [Online]. Available: https://cug.org/proceedings/cug2016_proceedings/includes/files/pap120.pdf

[2] A. Shterenlikht, L. Margetts, L. Cebamanos, and J. D. Arregui-Mena, "Multi-scale CAFE framework for simulating fracture in heterogeneous materials implemented in Fortran coarrays and MPI ," in *PGAS Application Workshop (PAW), Supercomputing 2016, USA* , 2016. [Online]. Available: http://conferences.computer.org/paw/2016/papers/5214a001.pdf

[3] A. Shterenlikht, L. Margetts, and L. Cebamanos, "Fortran coarray/MPI multi-scale CAFE for fracture in heterogeneous materials," in *PARENG2017, Int. Conf. Parallel, Distributed, Grid and Cloud Computing for Engineering, Pécs, Hungary*, 2017.

[4] S. Crouch, N. C. Hong, S. Hettrick, M. Jackson, A. Pawlik, S. Sufi, L. Carr, D. D. Roure, C. Goble, and M. Parsons, "The Software Sustainability Institute: Changing Research Software Attitudes and Practices," *Computing in Science & Engineering*, vol. 15, pp. 74–80, 2013. [Online]. Available: http://dx.doi.org/10.1109/MCSE.2013.133

[5] L. Margetts, "Parallel finite element analysis," Ph.D. dissertation, University of Manchester, 2002.

[6] I. M. Smith, *Programming the Finite Element Method*. Wiley, 1982.

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

[8] I. M. Smith and L. Margetts, "The convergence variability of parallel iterative solvers," *Eng. Computations*, vol. 23, pp. 154–165, 2006. [Online]. Available: http://dx.doi.org/10.1108/02644400610644522

[9] ——, "Portable parallel processing for nonlinear problems," in *Proc. VII Int. Conf. Computational Plasticity, Barcelona, Spain*, 2003.

[10] L. Margetts, I. M. Smith, and J. M. Leng, "Parallel 3d finite element analysis of coupled problems," in *Proc. III European Conf. Comp. Mechanics in Solids, Structures and Coupled Problems in Engineering, Lisbon, Portugal*, 2006.

[11] 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. [Online]. Available: http://dx.doi.org/10.1016/j.fusengdes.2015.04.048

[12] 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. [Online]. Available: http://dx.doi.org/10.1016/j.jnucmat.2015.05.058

[13] F. Levrero-Florencio, L. Margetts, E. Sales, S. Xie, K. Manda, and P. Pankaj, "Evaluating the macroscopic yield behaviour of trabecular bone using a nonlinear homogenisation approach," *J. Mech. Behavior Biomed. Mater.*, vol. 61, pp. 384–96, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.jmbbm.2016.04.008

[14] 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. [Online]. Available: http://dx.doi.org/10.1007/s10237-014-0593-5

[15] L. Margetts, I. M. Smith, L. M. Lever, and D. V. Griffiths, "Parallel processing of excavation in soils with randomly generated material properties," in *Proc. 8th European Conf. Numer. Methods in Geotechnical Engineering, Delft, Netherlands*, 2014, pp. 265–270.

[16] L. Margetts, J. M. Leng, I. M. Smith, and P. L. Manning, "Parallel three dimensional analysis of dinosaur trackway formation," in *Proc. 6th European Conf. Numer. Methods in Geotechnical Engineering, Graz, Austria*, 2006, pp. 743–749.

[17] 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

[18] 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. [Online]. Available: http://dx.doi.org/10.1111/j.1460-2695.2006.01031.x

[19] 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. [Online]. Available: http://dx.doi.org/10.1098/rspa.2006.1681

[20] S. J. Wu, C. L. Davis, A. Shterenlikht, and I. C. Howard, "Modeling the ductile-brittle transition behavior in thermomechanically contolled rolled steels," *Met. Mater. Trans. A*, vol. 36, pp. 989–997, 2005. [Online]. Available: http://dx.doi.org/10.1007/s11661-005-0292-z

[21] A. Shterenlikht, L. Margetts, S. A. McDonald, and N. K. Bourne, "Towards mechanism-based simulation of impact damage using exascale computing," *AIP Conf. Proc.*, vol. 1793, p. 080009, 2017. [Online]. Available: http://dx.doi.org/10.1063/1.4971615

[22] A. Shterenlikht, L. Margetts, L. Cebamanos, and D. Henty, "Fortran 2008 coarrays," *ACM Fortran Forum*, vol. 34, pp. 10–30, 2015. [Online]. Available: http://dx.doi.org/10.1145/2754942.2754944

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

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

[25] 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

[26] 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://dx.doi.org/10.1098/rspa.2015.0039

[27] S. Shende and A. D. Malony, "The TAU parallel performance system," *Int. J. High Perf. Comp. Appl.*, vol. 20, pp. 287–311, 2006. [Online]. Available: http://dx.doi.org/10.1177/1094342006064482

[28] H. Radhakrishnan, D. W. I. Rouson, K. Morris, S. Shende, and S. C. Kassinos, "Using coarrays to parallelize legacy Fortran applications: Strategy and case study," *Sci. Prog.*, vol. 2015, p. 904983, 2015. [Online]. Available: http://dx.doi.org/10.1155/2015/904983

[29] M. Haveraaen, K. Morris, D. Rouson, H. Radhakrishnan, and C. Carson, "High-performance design patterns for modern Fortran," *Sci. Prog.*, vol. 2015, p. 942059, 2015. [Online]. Available: http://dx.doi.org/10.1155/2015/942059

[30] L. Margetts, J. D. Arregui Mena, T. Hewitt, and L. Mason, "Parallel finite element analysis using the Intel Xeon Phi," in *Proc. Emerging Technology Conf. (EMiT 2016), ISBN 978-0-9933426-3-9*, 2016.