Profiling coarray+MPI miniapps with TAU and the Intel compiler

Anton Shterenlikht
Mech Eng Dept
Queen's Building, University Walk
Bristol BS8 1TR, UK
mexas@bris.ac.uk

Sameer Shende
Performance Research Lab
5294 University of Oregon
Eugene, OR 97403-5294, USA
sameer@cs.uoregon.edu

Luis Cebamanos
Edinburgh Parallel Computing Centre (EPCC)
The University of Edinburgh, King's Buildings
Edinburgh EH9 3FD, UK
l.cebamanos@epcc.ed.ac.uk

Lee Margetts
School of Mechanical, Aero and Civil Engineering
The University of Manchester
Manchester M13 9PL, UK
lee.margetts@manchester.ac.uk

Jose D. Arregui-Mena
School of Mechanical, Aero and Civil Engineering
The University of Manchester
Manchester M13 9PL, UK
jose.arregui-mena@manchester.ac.uk

ABSTRACT

Coarrays are a native Fortran means for SPMD parallel programming, implementing a single-sided communications model. Coarray Fortran belongs to the PGAS class of parallel languages. TAU (Tuning and Analysis Utilities) toolkit is used in this work for profiling and tracing of pure coarray programs and mixed coarray+MPI miniapps. We show that the Intel MPI compiler maps coarray remote operations and synchronisation routines onto MPI-2 RMA calls. Four coarray programs, of progressively increasing complexity, are studied. It is shown that in iterative programs, where some remote data access and image synchronisation are performed each iteration, performance is heavily dominated by `MPI_Win_unlock` routine. These results might open a possibility for Intel developers to improve and optimise their MPI based coarray implementation.

CCS Concepts

•**Computing methodologies** → **Parallel programming languages; Massively parallel and high-performance simulations;** *Multiscale systems;*

Keywords

Fortran coarrays; MPI; RMA; TAU; profiling; tracing; Intel

1. INTRODUCTION

Coarrays are a native Fortran means for SPMD parallel programming [16]. Although coarrays (or co-arrrays, Co-Array Fortran (CAF), as they were originally known) have been used, particularly on Cray systems, as an extension, for nearly 20 years [10], they became part of the Fortran standard only in 2010 [6]. Coarray capabilities will be substantially expanded in the Fortran 2015 standard [7]. Coarrays offer simple syntax and portability for SPMD standard conforming Fortran programs. Coarrays can be added gradually to existing Fortran projects and can co-exist with other popular parallel technologies, primarily OpenMP and MPI [9]. In an attempt to ensure coarray integrity, the standard writers introduced very strict image ordering and synchronisation rules. As a result, a standard conforming coarray program will not deadlock or suffer races.

However, coarray performance can vary significantly. This is partly because coarrays represent a very high level of abstraction, and Fortran compilers can use different transport libraries to map coarrays data and communications to hardware. Cray systems use DMAPP, the Intel implementation uses MPI and the OpenCoarrays implementation is designed to support MPI and GASNet [4].

In our previous work we successfully used proprietary Cray-PAT profiling and sampling tools to optimise performance of mixed coarray+MPI miniapps on Cray systems [1]. Recently TAU (Tuning and Analysis Utilities) was shown to support coarray programs [11, 5]. In this work we use TAU to profile and trace several pure coarray and mixed coarray+MPI programs using the Intel Fortran compiler.

2. PROFILING AND TRACING WITH TAU

TAU[1] [12], is a popular open source set of tools for performance analysis, particularly on HPC systems. In this work TAU 2.25.1 was used.

TAU is often used together with the Program Database Toolkit (PDT), [2], which is a framework for analysing source code. However, PDT 3.22 does not yet support coarrays.

[1]https://www.cs.uoregon.edu/research/tau
[2]https://www.cs.uoregon.edu/research/pdt

A progress report, JUN-2016, The University of Bristol, UK
ACM ISBN X-XXXXX-XX-X/XX/XX.
DOI: 0

Hence compiler based instrumentation was used in this work, which is enabled with TAU flag `-optCompInst`.

Jumpshot-4, developed by the Argonne National Lab (ANL)[3], was used to view TAU traces.

The University of Bristol BlueCrystal phase 3 system was used for this work[4]. Each node has a single 16-core 2.6 GHz SandyBridge CPU and 64GB RAM. Intel Cluster Studio XE, version 16.0.2, was used. TAU was configured with

```
-mpi -c++=mpiicpc -cc=mpiicc -fortran=mpiifort
```

2.1 Case studies

2.1.1 Calculation of π using Gregory-Leibniz series

This example is taken from the University of Bristol coarrays course[5], folder `examples/prof/tau/5pi`.

π can be calculated using the Gregory-Leibniz as follows:

$$\pi = 4 \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{2n-1} \quad (1)$$

This is a classical parallel problem - all workers calculate their own partial sums in parallel. The total sum is calculated via a collective routine or by a master process. The series limit was set arbitrarily at 2^{35}. The key fragment is shown below.

```
real :: pi[*]
do i = this_image(), 2**35, num_images()
 pi = pi + (-1)**(i+1) / real( 2*i-1 )
end do
sync all ! all images synchronise here
if ( this_image() .eq. 1 ) then
  do i = 2, num_images()
    pi = pi + pi[i]
  end do
  pi = pi * 4.0
end if
```

Coarray collectives are not yet in the Fortran standard. They are described in [7] and will become a part of Fortran 2015 standard. Cray and OpenCoarrays already support coarray collectives. Intel Fortran 16 does not. Hence image 1 is calculating the total sum by pulling partial sums from all other images with `pi = pi + pi[i]`.

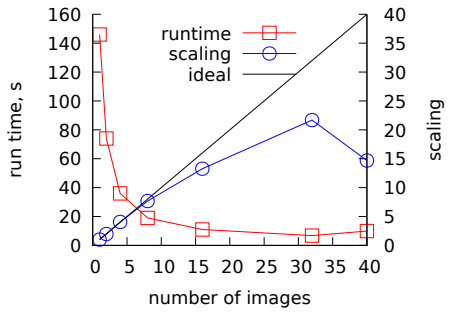

Figure 1: Calculating π on 2 nodes - scaling.

[3]http://www.mcs.anl.gov/research/projects/perfvis/software/viewers/index.htm

[4]http://www.acrc.bris.ac.uk

[5]http://coarrays.sourceforge.net

Fig. 1 shows scaling of this code on two nodes with Intel `-fast` optimisation. As expected, performance drops when the number of images exceeds the number of cores. We profile this code on 32 cores with 32 images.

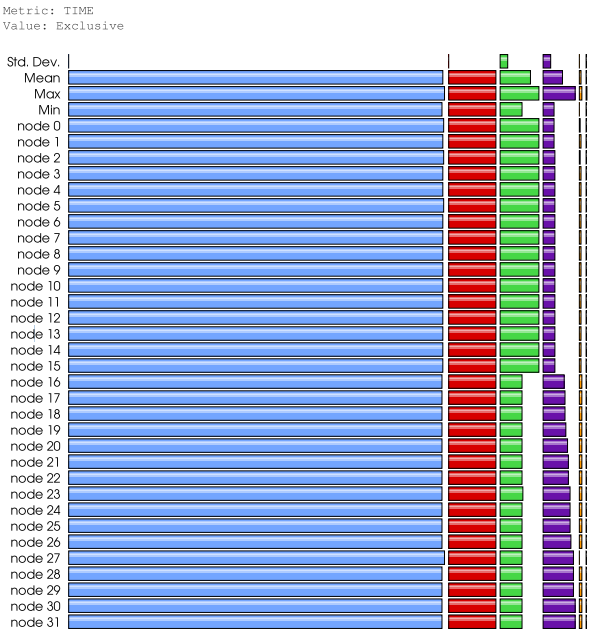

Figure 2: Calculating π with 32 images - profiling.

Fig. 2 shows an even spread of load over all nodes, sorted by exclusive time. Note that TAU reports nodes starting from 0, so node n means image $n+1$. `MPI_Init` (green) and `MPI_Finalize` (purple) times differ between nodes 0-15 and nodes 16-31, highlighting the boundary between the physical CPUs. Time spent in other MPI routines is negligible.

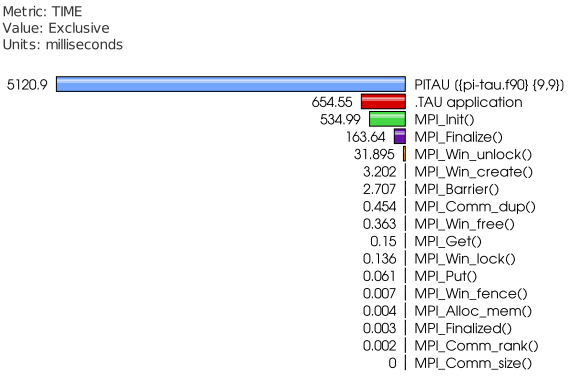

Figure 3: Calculating π - single image profile.

Fig. 3 shows exclusive time profile on node 13 (image 14). It is clear that Intel implementation of coarrays uses remote memory access (RMA) one-sided communications of MPI-2, where a typical sequence of calls for a lock/unlock synchronisation is `MPI_Win_create`, `MPI_Win_lock`, `MPI_Put`, `MPI_Get`, `MPI_Win_unlock` and `MPI_Win_free` [18]. Note that Fig. 3

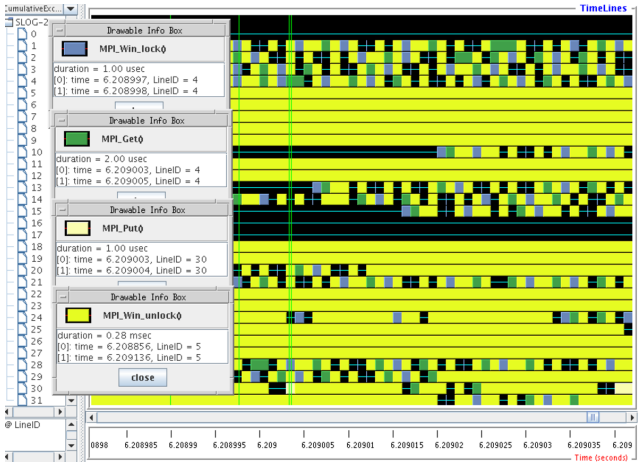

Figure 4: Calculating π with 32 images - a trace fragment.

shows that of these calls, nearly 90% of time is spent in `MPI_Win_unlock`. A fragment of the trace for this program, taken after `sync all`, is shown in Fig. 4. It highlights the dominance of `MPI_Win_unlock` calls.

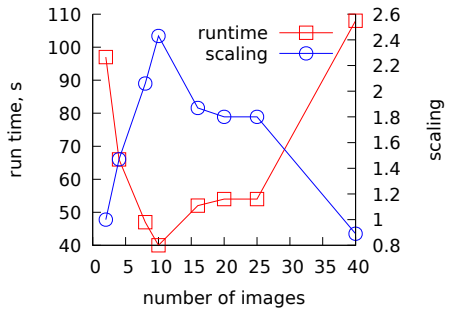

Figure 5: Scaling of the Laplacian code `coback1-tau` on 2 nodes.

2.1.2 *A Laplacian solver*

This example is taken from the University of Bristol coarrays course[6], folder `examples/prof/tau/9laplace`.

Program `coback1-tau.f90` iteratively reconstructs a 2D array `p` from previously calculated 2D edge array `edge`. Halo exchange and `sync images` synchronisation is used every iteration. The key fragment is show below.

```
img = this_image() ; nimgs = num_images()
outer: do iter = 1, niter
  if (img .ne. 1) op(:, 0) = op(:, size2) [img-1]
  if (img .ne. nimgs) op(:, size2+1)=op(:, 1)[img+1]
  do j = 1, size2 ; do i = 1, width
      p(i,j) = 0.25 * ( op(i-1,j) + op(i+1,j) + &
           op(i,j-1) + op(i,j+1) - edge(i,j) )
    end do ; end do
  op = p
  if ( img .ne. 1 ) sync images( img-1 )
  if ( img .ne. nimgs ) sync images( img+1 )
end do outer
```

[6]http://coarrays.sourceforge.net

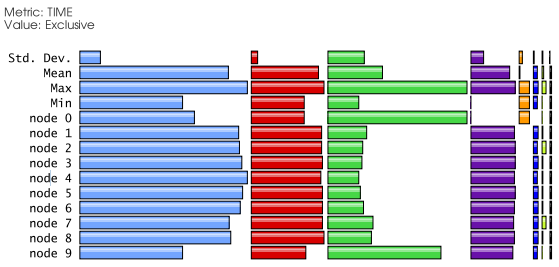

Figure 6: Profile of `coback1-tau` on 10 images.

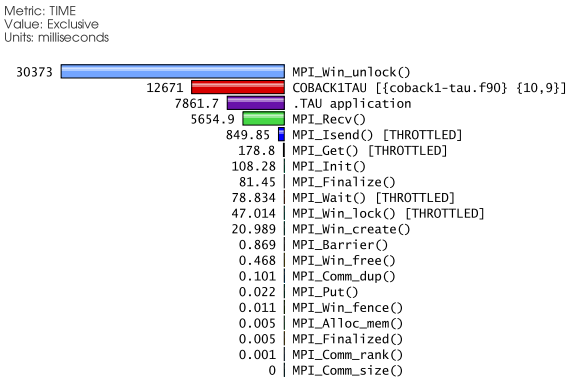

Figure 7: Profile of `coback1-tau` on a single image.

Fig. 5 shows scaling of program `coback1-tau` on two nodes. Best performance is achieved on 10 images. Profiling results with 10 images are shown in Figs. 6 and 7.

`MPI_Win_unlock` now dominates the run time, and the program itself, `cobacktau`, accounts for less than half of time. Significant load imbalance is seen in `MPI_Recv`.

Fig. 8 shows the data transfer pattern and dominance of `MPI_Win_unlock` calls in the program at the exit from the `outer` loop. Note also an emerging pattern in remote calls, when no pattern is present in the source code.

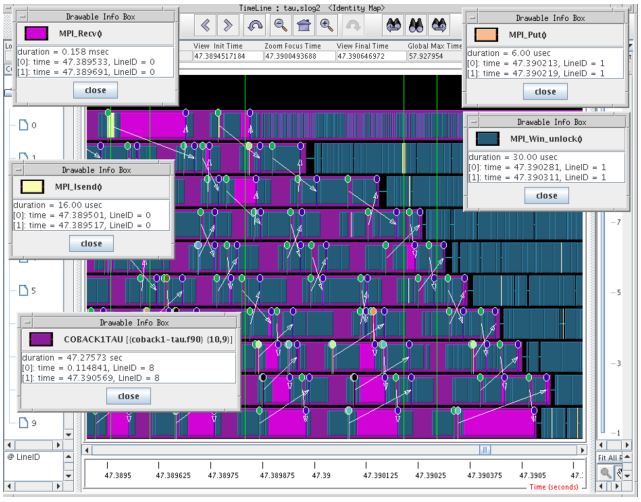

Figure 8: A fragment of the trace of the Laplacian code `coback1-tau` on 10 images.

2.1.3 *Cellular automata microstructure simulation code*

CGPACK,[7] is a BSD licensed Fortran coarray library for microstructure simulation [15, 13]. It uses a cellular automata approach, where a 3D space is represented as a structured grid of cells. Cell states are updated iteratively. At every iteration the state of each cell is determined by the states of its neighbouring cells and, possibly, by a superimposed field, such as temperature or strain.

The CGPACK distribution includes a number of test programs, e.g. `tests/testABW.f90`, studied here. This program simulates formation and cleavage fracture of polycrystalline microstructure. It calls multiple routines to manipulate this coarray:

```
integer, allocatable :: space(:,:,:,:) [:,:,:]
```

Intel `-O2` optimisation was used. Figs. 9 and 10 show that the time is dominated by `MPI_Win_unlock` even more than in the previous examples. `MPI_Barrier` is in the second place. Only 2 CGPACK routines, the halo exchange, `cgca_hxi`, and the cleavage fracture propagation, `cgca_clvgp_nocosum`, exceed the threshold of 1% of the total time.

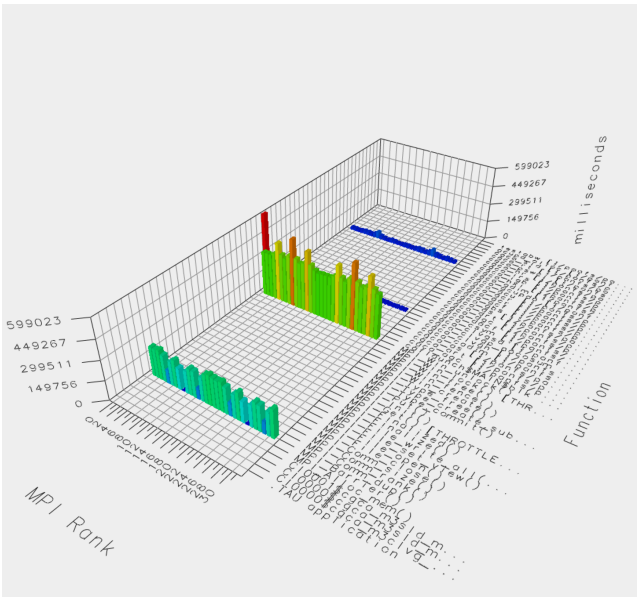

Figure 9: Profile of CGPACK program `testABW`.

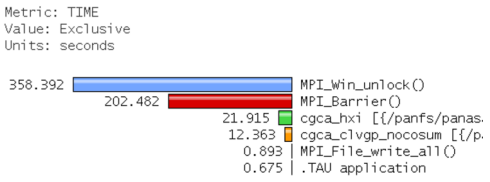

Figure 10: Profile of `testABW` on a single image.

Fig. 11 shows the trace of `testABW`, while the program is inside the halo exchange routine `cgca_hxi`. `MPI_Win_unlock` is run on all images except 17, which is in `MPI_Barrier`. The source code points to no obvious explanation of why only a single image would call a global barrier at this point.

[7] http://cgpack.sourceforge.net

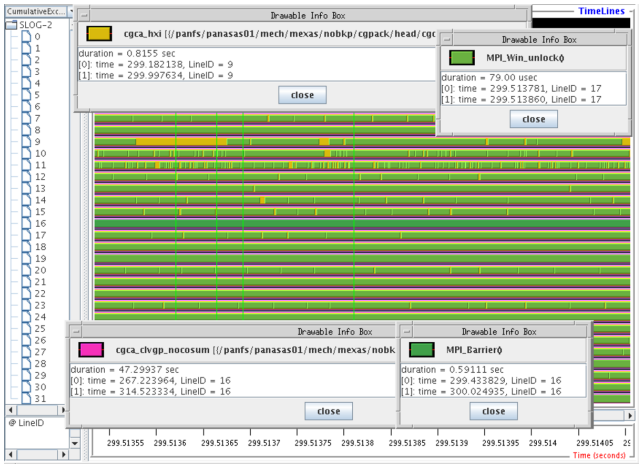

Figure 11: A fragment of trace of `testABW`.

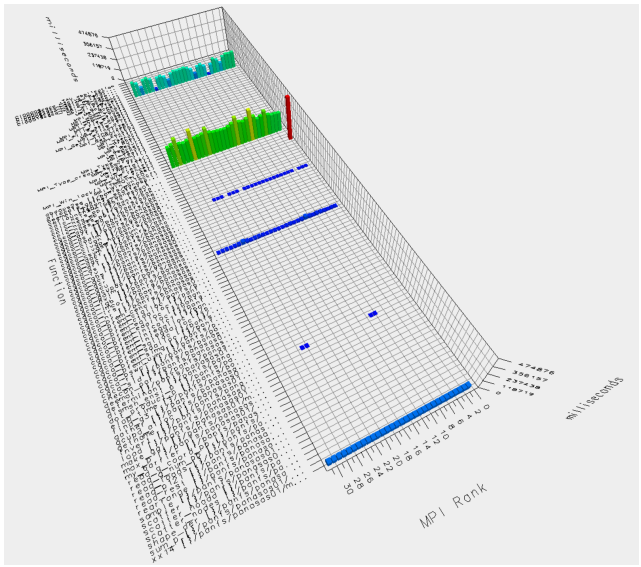

Figure 12: 3D profiling bar chart of program `xx14std`, coarray+MPI, on 2 nodes with 32 images.

2.1.4 *Multiscale coarray+MPI fracture model*

ParaFEM,[8] is a BSD licensed[9] highly scalable Fortran MPI finite element library [17]. It has recently been used in nuclear fusion research [3] and biomechanics [8].

By linking ParaFEM with CGPACK a multi-scale cellular automata finite element (CAFE) framework was created [15]. In CAFE approach structural scale is represented with FE and material microstructure evolution is modelled with CA. A concurrent two-way information transfer is established between the FE and the CA layers [14, 2].

ParaFEM includes several CAFE miniapps, as a set of developer programs, under `src/programs/dev/xx14`. In this work we profile `xx14std.f90`. Both ParaFEM and CGPACK libraries were instrumented with TAU. Intel `-O2` optimisation was used. Profiling was done on 2 nodes with 32 images.

[8] http://parafem.org.uk

[9] https://sourceforge.net/projects/parafem

Figs. 12-14 show the profiling results for `xx14std`. The observations are consistent with the previous two examples. The load is well balanced across all images. However, `MPI_Win_unlock` dominates the run time and `MPI_Barrier` is in the second place. Only the program itself, `xx14`, the halo exchange routine, `cgca_hxi`, and the fracture propagation routine, `cgca_clvgp_nocosum`, exceed the threshold of 1% of the total time.

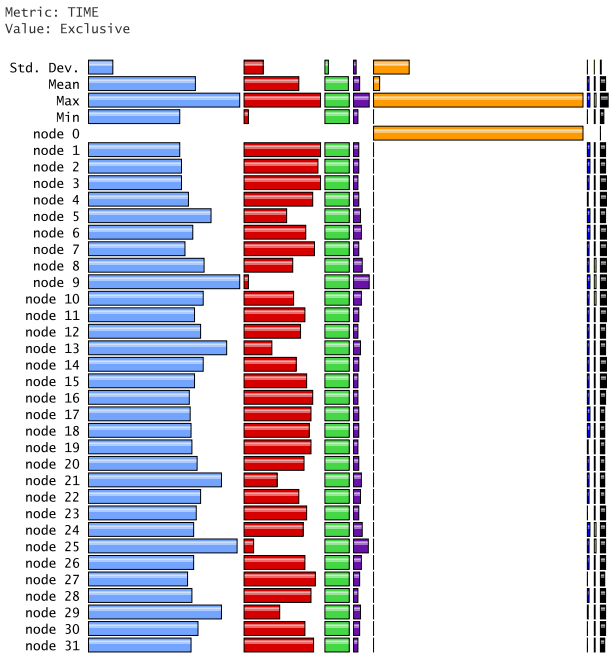

Figure 13: Profile of program `xx14std`, coarray+MPI, on 2 nodes with 32 images.

Note that the Intel coarray implementation includes task `caflaunch`, which is assigned a rank 0 (orange bar in Fig. 13). In TAU 2.25.1, when compiler instrumentation is used, as in the this work, `caflaunch` is instrumented too. This creates a problem that both the `caflaunch` thread and the coarray program thread write to the profiling file on node 0 (image 1). Because `caflaunch` persists until the program exits, this process overwrites all program data on node 0, so only `caflaunch` is seemingly present there, as seen in Fig.

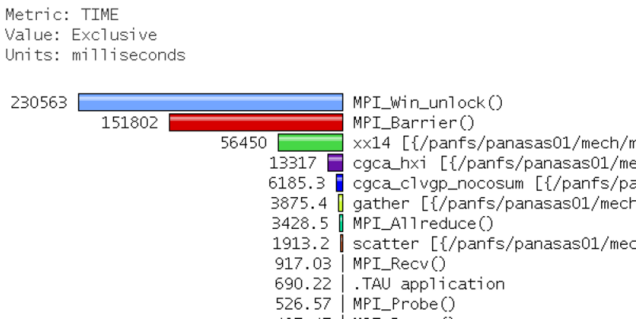

Figure 14: Profile of program `xx14std`, coarray+MPI, on a single image.

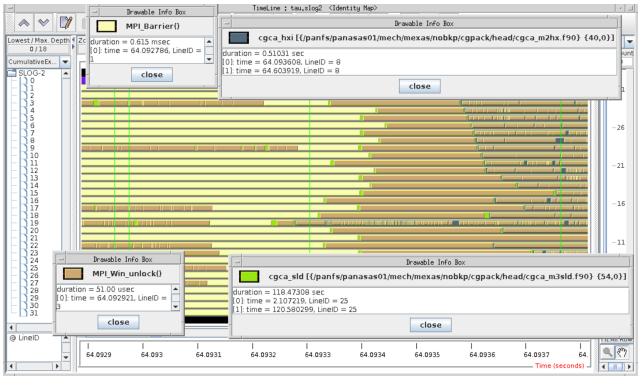

Figure 15: A fragment of the trace of coarray+MPI program `xx14std`, inside CGPACK routine `cgca_sld`.

13. The TAU team has proposed a fix for this behaviour, which is being tested at present.

`cgca_clvgp_nocosum` is supposed to be the most computationally expensive routine in program `xx14std`. It has a triple nested loop over the first 3 dimensions of `space` array coarray. Profiling of this program on Cray XC30 showed that indeed most of the time was spent in this routine [1]. As was mentioned in the introduction, Cray implementation of coarrays does not use MPI, so no direct comparison can be drawn with the present results.

Traces of `xx14std` show clear differences between those MPI calls which are included directly in the ParaFEM library, and the MPI calls into which the Intel compiler translated the CGPACK coarray remote operations.

Fig. 15 shows a typical fragment of the trace of `xx14std`, where CGPACK coarray routines are executed. No communication pattern or structure can be seen. `MPI_Win_unlock` and `MPI_Barrier` are executed from CGPACK halo exchange routine `cgca_hxi`, which is called from `cgca_sld`.

In contrast, Fig. 16 shows a very well structured MPI communication pattern translated directly from the MPI calls in the ParaFEM library.

3. CONCLUSIONS

TAU (Tuning and Analysis Utilities) toolkit is well suited for profiling and tracing analysis of pure coarray and mixed coarray+MPI programs, where coarray operations are ultimately mapped onto MPI calls. The Intel implementation of coarrays seems to rely on MPI-2 RMA, with `MPI_Win_unlock` heavily dominating run times in three out of four analysed programs. These results indicate a potential for optimisation of (MPI based) coarray implementations, such as Intel or OpenCoarrays.

4. ACKNOWLEDGMENTS

This work was carried out using the computational facilities of the Advanced Computing Research Centre, The University of Bristol, http://www.bris.ac.uk/acrc.

5. REFERENCES

[1] L. Cebamanos, A. Shterenlikht, D. Arregui, and L. Margetts. Scaling hybid coarray/MPI miniapps on Archer. In *Cray User Group meeting (CUG2016), London, 8-12-MAY-2016*, 2016.

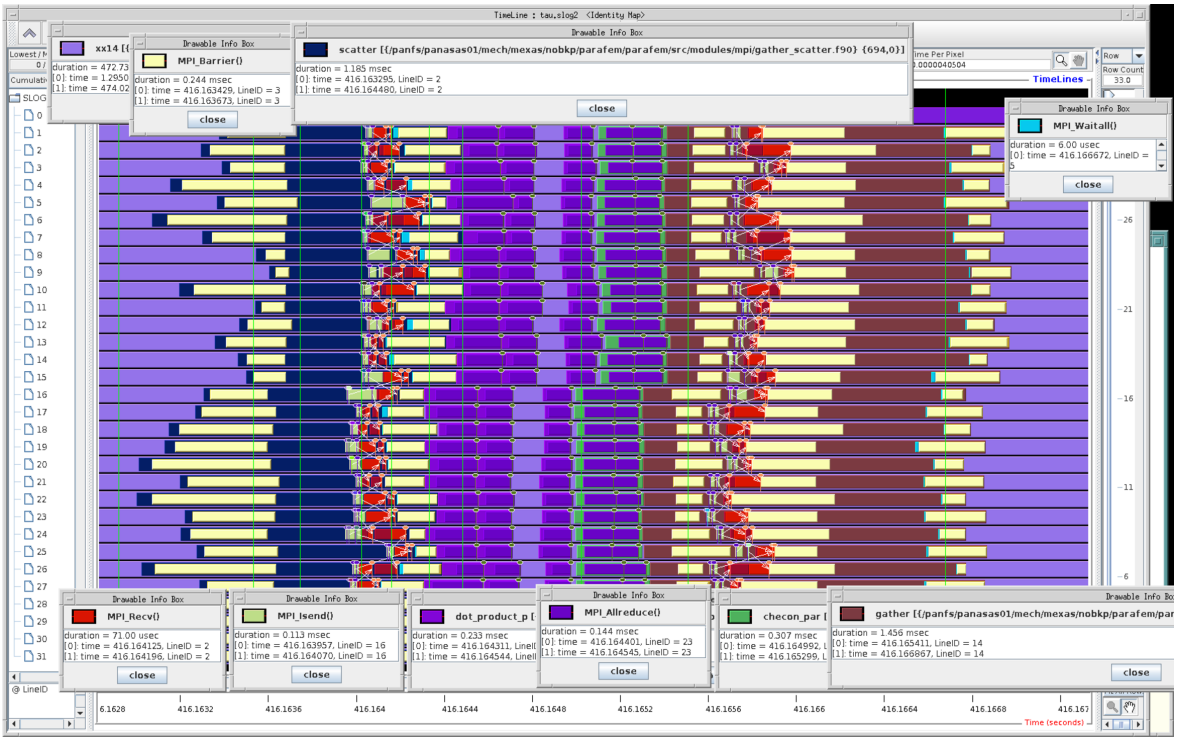

Figure 16: A fragment of the trace of program `xx14std`, showing MPI routines included directly in the ParaFEM library.

[2] 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*, 462:2085–2096, 2006.

[3] 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.*, 100:100–111, 2015.

[4] A. Fanfarillo. *Parallel Programming Techniques for heterogeneous exascale computing platforms*. PhD thesis, University of Rome Tor Vergata, Italy, 2016.

[5] M. Haveraaen, K. Morris, D. Rouson, H. Radhakrishnan, and C. Carson. High-performance design patterns for modern Fortran. *Sci. Prog.*, 2015:942059, 2015.

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

[7] ISO/IEC JTC1/SC22/WG5 N2048. *TS 18508 Additional Parallel Features in Fortran*. 2015.

[8] F. Levrero, 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.*, 61:384–396, 2016.

[9] 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.*, 29:261–273, 2015.

[10] R. W. Numrich, J. Reid, and K. Kim. Writing a multigrid solver using Co-Array Fortran. *Appl. Parallel. Comp.*, 1541:390–399, 1998.

[11] 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.*, 2015:904983, 2015.

[12] S. Shende and A. D. Malony. The TAU parallel performance system. *Int. J. High Perf. Comp. Appl.*, 20:287–331, 2006.

[13] A. Shterenlikht. Fortran coarray library for 3D cellular automata microstructure simulation. In *Proc. 7th PGAS Conf., Edinburgh, Scotland, UK*, 2013.

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

[15] A. Shterenlikht and L. Margetts. Three-dimensional cellular automata modelling of cleavage propagation across crystal boundaries in polycrystalline microstructures. *Proc. Roy. Soc. A*, 471:20150039, 2015.

[16] A. Shterenlikht, L. Margetts, L. Cebamanos, and D. Henty. Fortran 2008 coarrays. *ACM Fortran Forum*, 34:10–30, 2015.

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

[18] V. Tipparaju, W. Gropp, H. Ritzdorf, R. Thakur, and J. L. Träff. Investigating high performance RMA interfaces for the MPI-3 standard. In *Proc. 2009 Int. Conf. Parallel Processing*, 2009.