Fortran coarray library for 3D cellular automata microstructure simulation

Anton Shterenlikht

Mech Eng Dept, University of Bristol, UK
mexas@bris.ac.uk
http://eis.bris.ac.uk/∼mexas/cgpack

PGAS 2013 Conference, 3-4-OCT-2013, EPCC, Edinburgh, UK

Outline

3D cellular automata models of materials

Fortran coarrays

Modelling results

Synchronisation, profiling and scaling

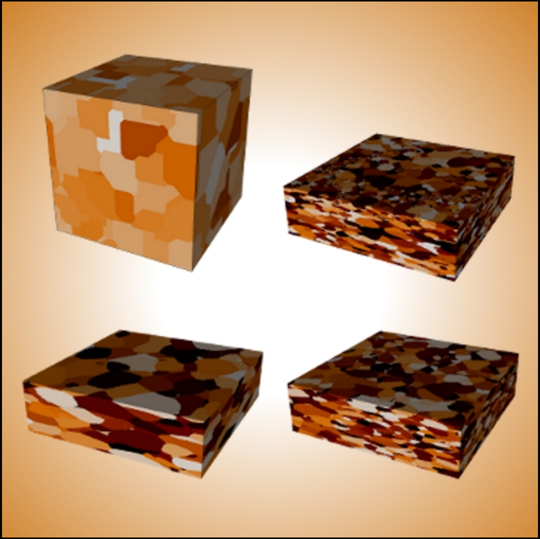
Unresolved: coarray I/O

Multiple physics, size and time scales

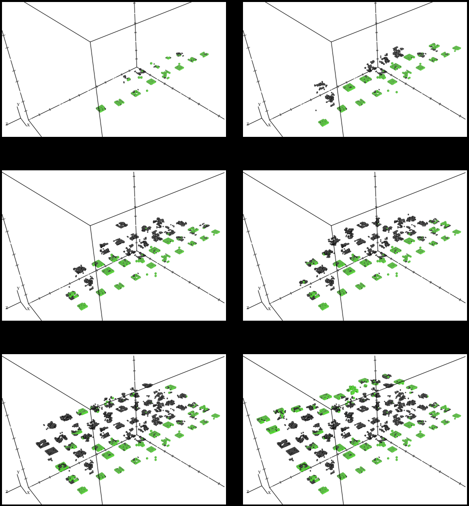
physics	size, m	time, s
recrystallisation	$10^{-9} \dots 10^{-2}$	$10^{-6} \dots 10^{5}$
cleavage	$10^{-6} \dots 10^{1}$	$10^{-6} \dots 10^{-3}$
ductile fracture	$10^{-8} \dots 10^{1}$	$10^{-6} \dots 10^{2}$
fatigue	$10^{-6} \dots 10^{-3}$	$10^{1} \dots \infty$
corrosion	?	?

- molecular dynamics
- discrete & continuous dislocation
- cellular automata, CA
- finite elements (boundary elements, meshless, etc.)

CA examples: recrystallisation

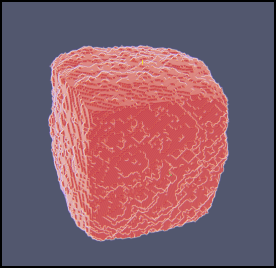


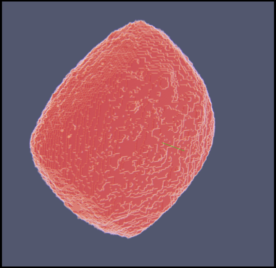
CA examples: ductile to brittle transitional fracture in steel – Charpy impact test [1]

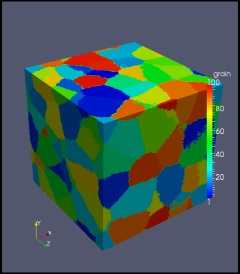


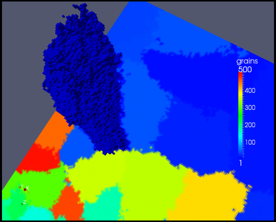
- Green - ductile failure
- Black - transgranular cleavage

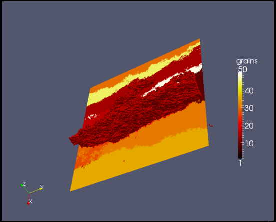
CA examples: single- and poly-crystals

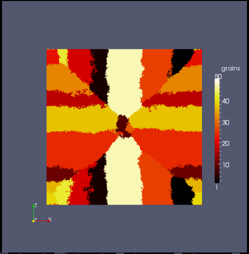




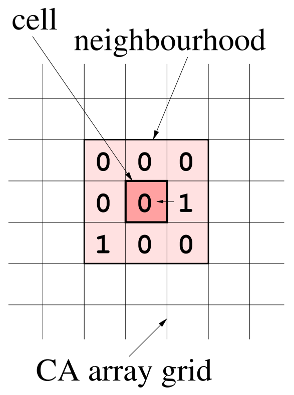








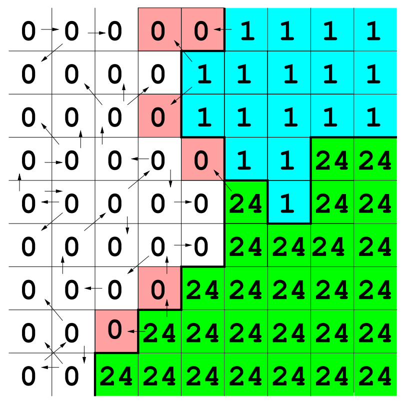
Basics of a 3D CA model



- 3D space, partitioned into identical cubic cells
- Self-similar or fixed boundaries
- Multiple, pre-defined cell states, e.g. liquid, grain, crack, etc.
- A grain is a collection of cells with the same state
- A neighbourhood of 26 nearest neighbours
- The state of each cell at $t+1$ is a function of the state of the neighbourhood at t

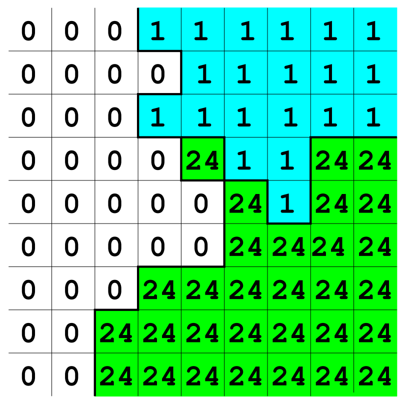
Basics of a 3D model: solidification

A liquid cell (state 0) acquires state of a randomly chosen neighbour:



0	0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1	1
0	0	0	0	1	1	1	1	1
0	0	0	0	0	1	1	24	24
0	0	0	0	0	24	1	24	24
0	0	0	0	0	24	24	24	24
0	0	0	0	24	24	24	24	24
0	0	0	24	24	24	24	24	24
0	0	24	24	24	24	24	24	24

t



0	0	0	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1
0	0	0	1	1	1	1	1	1
0	0	0	0	24	1	1	24	24
0	0	0	0	0	24	1	24	24
0	0	0	0	0	24	24	24	24
0	0	0	24	24	24	24	24	24
0	0	24	24	24	24	24	24	24
0	0	24	24	24	24	24	24	24

$t+1$

CA computational challenge

- High spatial resolution is required, i.e. 10^5 cells per grain [2]
- Typical engineering steel has a grain size about $10^1 \ldots 10^2$ μm
- 1 cm^3 of such material might have $10^6 \ldots 10^9$ grains
- The model will need $10^{11} \ldots 10^{14}$ cells
- A complete solution might require $10^3 \ldots 10^5$ iterations

Conclusion: need lots of memory and fast processing!

Fortran coarray basics

- Simple new syntax for SIMD problems
- Feature of Fortran 2008 standard [3]
- Fortran coarrays, **NOT** Coarray fortran or CAF
- Standard Fortran is important for linking with other Fortran codes
- Coarray collectives and teams by about 2015 [4]
- Cray provide collectives as an extension to the standard
- Cray and Intel coarray support is most comprehensive [5]
- Most thoroughly explained in "MFE" – Metcalf, Reid, Cohen (2011) Modern Fortran Explained [6]

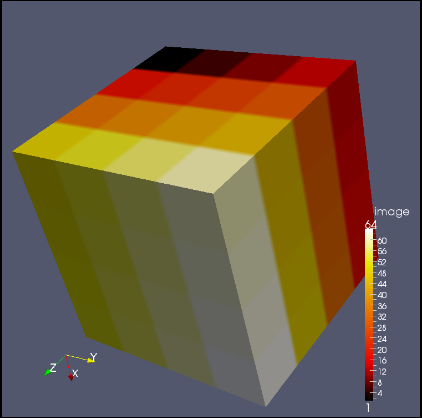
Fortran coarray example

```
integer :: i[*]                        ! scalar coarray
real,allocatable :: r(:)[:]            ! array coarray
integer :: img, num
img = this_image()                     ! new f2008 intrinsic
num = num_images()                     ! new f2008 intrinsic
i = img
r = 1.0
allocate (r(100)[*])                   ! allocating array coarray
if (img .eq. 1   ) &
 i = i + i[num]                        ! remote read
if (img .eq. num) &
 r(5)[1] = sum(r)                      ! remote write
end
```

CA data structures

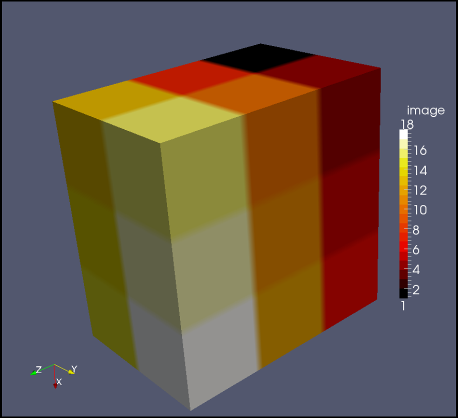
- Model *assembly*, not *partition*: `real :: z(1000)[*]` run on 64 images effectively gives a real array of 64,000
- Coarrays waste memory, use sparingly
- The model itself:
  `integer,allocatable :: coarray(:,:,:,:)[:,:,:]`
  note 3D image grid, to minimise boundary area, i.e. minimise halo exchange data transfer
- Various other arrays, depending on the need, e.g. the grain size:
  `integer,allocatable :: gs(:)[:,:,:]`
  note the wasted memory

CA model coarray



```
allocate( &
coarray(10,10,10)[4,4,*])
```

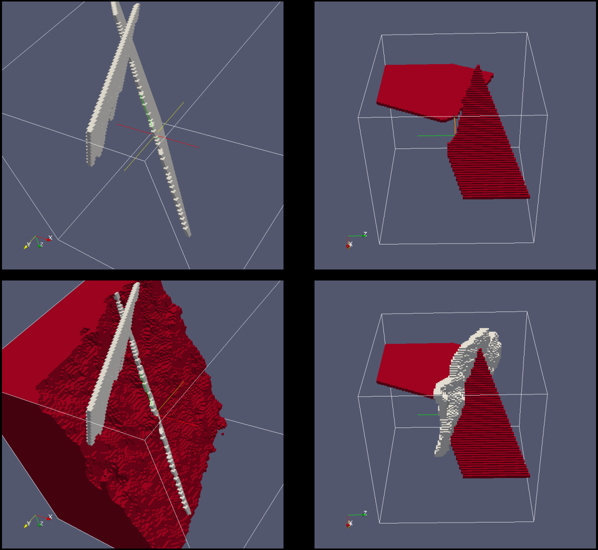
on 64 images



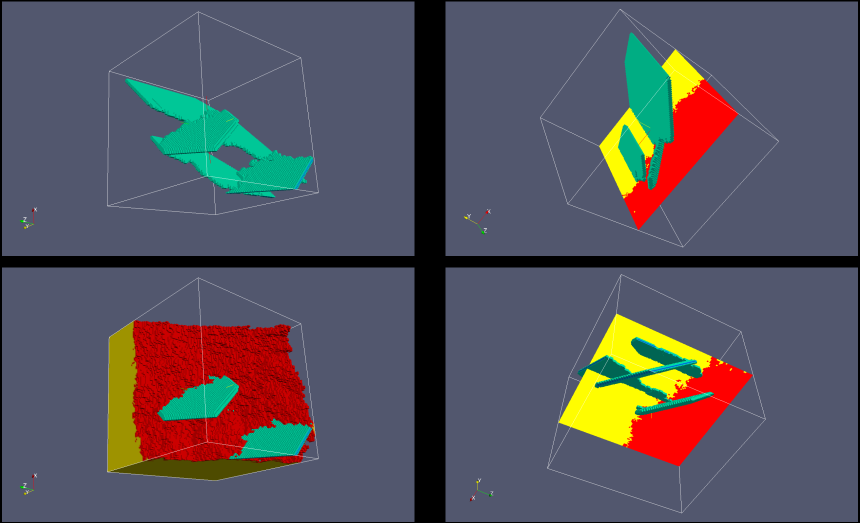
```
allocate( &
coarray(10,10,10)[3,2,*])
```

on 18 images

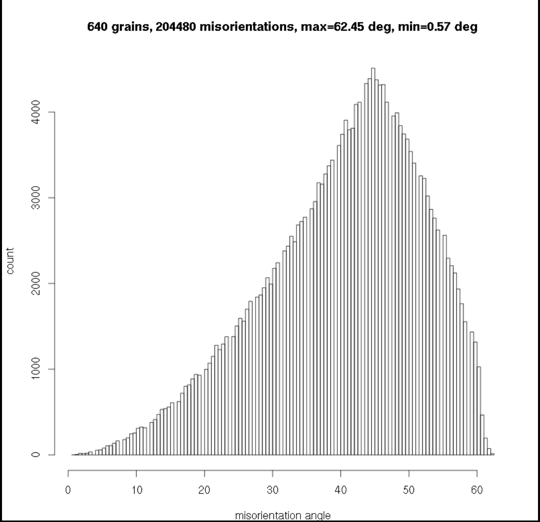
Modelling results: cleavage 1



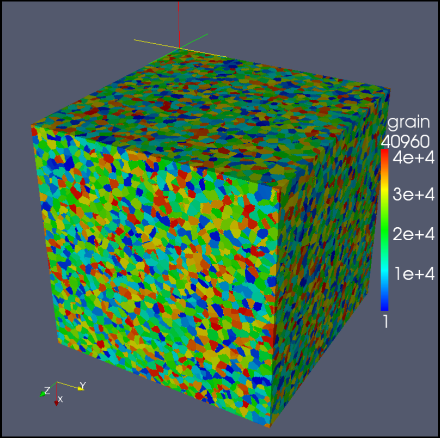
Modelling results: cleavage 2

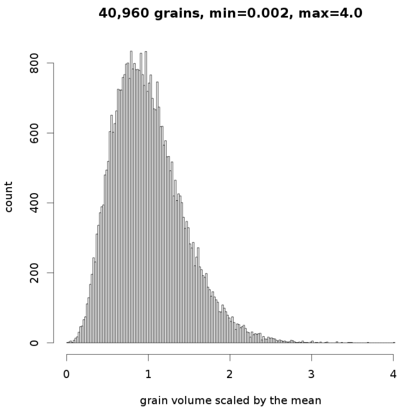


Modelling results: grain mis-orientation distribution



Modelling results: micro-structure, 10^9 cells





Solidification kernel

```
integer :: fin, x1, x2, x3, z(3), lbr(3), ubr(3)
logical :: finished
real :: candidate(3)
main: do
  array = space( :, :, :, type_grain )
  do x3 = lbr(3),ubr(3)
  do x2 = lbr(2),ubr(2)
  do x1 = lbr(1),ubr(1)
    if ( space( x1, x2, x3, type_grain ) .eq. liquid ) then
      call random_number( candidate )     ! 0 .le. candidate .lt. 1
      z = nint( candidate*2 - 1 )         ! step = [-1 0 1]
      array( x1,x2,x3 ) = space( x1+z(1), x2+z(2), x3+z(3), type_grain )
    end if
  end do
  end do
  end do
  space( :, :, :, type_grain ) = array
  finished = all( space( lbr(1):ubr(1), lbr(2):ubr(2), lbr(3):ubr(3), &
                   type_grain ) .ne. liquid )
  fin = 1                   ! not solid yet
  if (finished) fin = 0     ! solid
!!!  exit if (fin .eq. 0) on *all* images
end do main
```

Synchronising solidification

```
integer :: fin, x1, x2, x3, z(3), lbr(3), ubr(3)
logical :: finished
real :: candidate(3)
main: do

call hxi(coarray) ! halo exchange - remote read only - no sync required!

! kernel

  finished = all( space( lbr(1):ubr(1), lbr(2):ubr(2), lbr(3):ubr(3), &
                   type_grain ) .ne. liquid )
  fin = 1                   ! not solid yet
  if (finished) fin = 0     ! solid

! some sync required
! global reduction
! some sync required

!!!  exit if (fin .eq. 0) on *all* images

end do main
```

Global reduction

1. Image 1 does all reduction work, other images wait
2. All images do reduction work, one at a time
3. Divide & conquer
4. Cray collectives

In the following:

```
  img = this_image()
nimgs = num_images()
```

Global reduction: serial algorithms

image 1 only

```
if ( img .eq. 1 ) then
  do i = 2,nimgs
    fin = fin + fin[i]
  end do
end if

sync all
fin = fin[1]
```

all images in turn

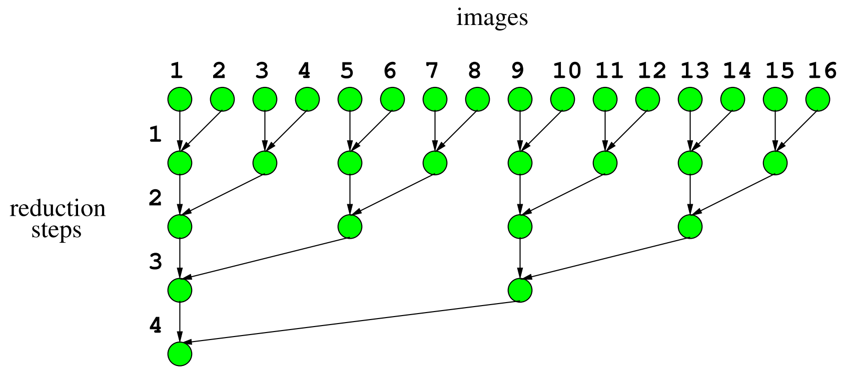
```
if ( img .ne. 1 ) then
  sync images (img - 1)
  fin[1] = fin[1] + fin
end if

if ( img .lt. nimgs )&
  sync images (img + 1)

sync all
fin = fin[1]
```

On `sync images`

- `sync images` are *not* tagged!
- `sync images` must match!
  From the standard [3]:

  > *Executions of SYNC IMAGES statements on images M and T correspond if the number of times image M has executed a SYNC IMAGES statement with T in its image set is the same as the number of times image T has executed a SYNC IMAGES statement with M in its image set.*

Divide & conquer or a binary tree



2^p images need p reduction steps.
Generally n images need $\log n$ steps.

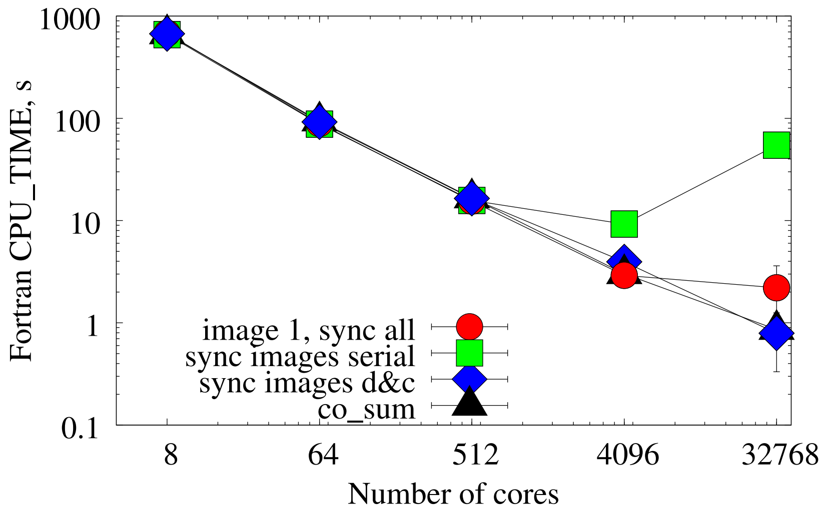
Divide & conquer synchronisation

```
step    = 2
stepold = 1
redu: do i = 1,p
  if ( mod(img,step) - 1 .eq. 0 ) then
    sync images ( img + stepold )
    fin = fin + fin[ img + stepold ]
  else if ( mod(img+stepold,step) - 1 .eq. 0 ) then
    sync images ( img - stepold )
  end if
  stepold = step
  step = step * 2
end do redu

sync all
fin = fin[1]
```

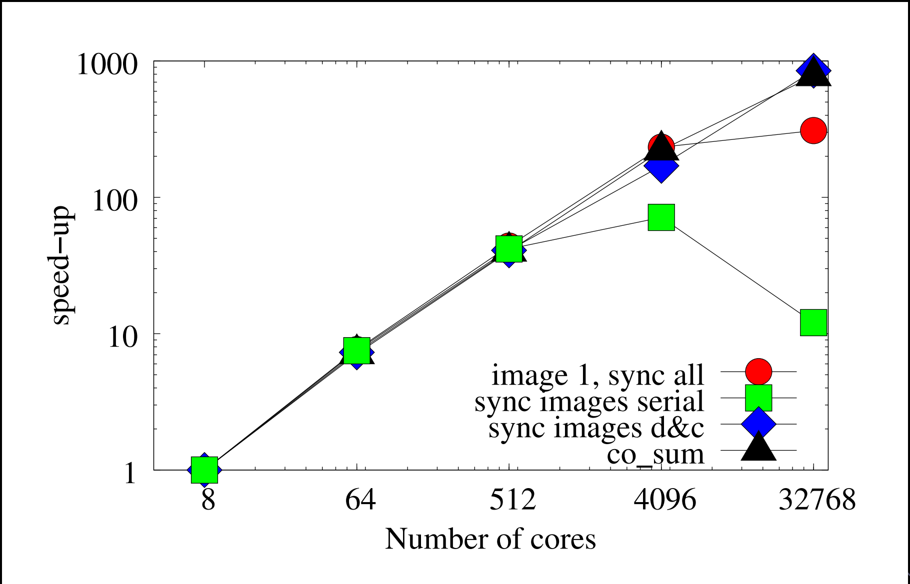
Cray `co_sum` reduction

- Cray extension to the standard [3]
- Quite similar to what is proposed in TS 18508 [4]
- Must be called by all images
- Can be used 'only in a context that allows an image control statement' [4]
- Hence synchronisation might or might not be required, depending on the algorithm
- No synchronisation is required in the solidification algorithm!
- `call co_sum(fin)`

Performance : HECToR XE6 : time



Performance : HECToR XE6 : speedup



Performance : HECToR XE6 : conclusion

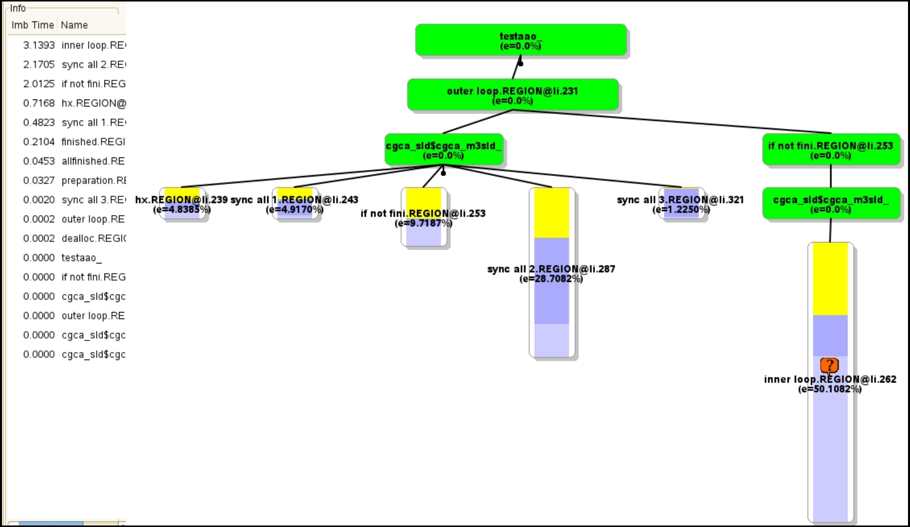
- A model with 2^{30} cells
- From $2^3 = 8$ cores, as: `coarray(512,512,512)[2,2,2]`
- to $2^{15} = 32768$ cores as: `coarray(32,32,32)[32,32,32]`
- The times calculated with `cpu_time` intrinsic
- The speed-up is nearly 10^3 for the core count raising by a factor of $2^{12} = 4096$!

Profiling: CrayPAT [7] on HECToR XE6

- API calls

-
  ```
  call pat_region_begin(num, "name", pat_status)
  ...
  call pat_region_end  (num,         pat_status)
  ```
- Profiling runs were done on 4096 processors

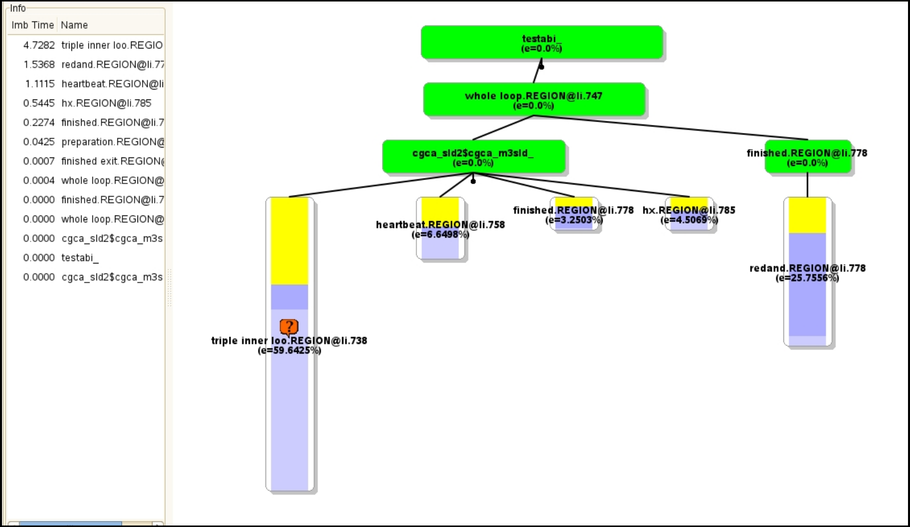
Profiling: serial reduction



Info

Imb Time	Name
3.1393	inner loop.RE
2.1705	sync all 2.RE
2.0125	if not fini.REG
0.7168	hx.REGION@
0.4823	sync all 1.RE
0.2104	finished.REGI
0.0453	allfinished.RE
0.0327	preparation.RE
0.0020	sync all 3.RE
0.0002	outer loop.RE
0.0002	dealloc.REGI
0.0000	testaao_
0.0000	if not fini.REG
0.0000	cgca_sld$cgc
0.0000	outer loop.RE
0.0000	cgca_sld$cgc
0.0000	cgca_sld$cgc

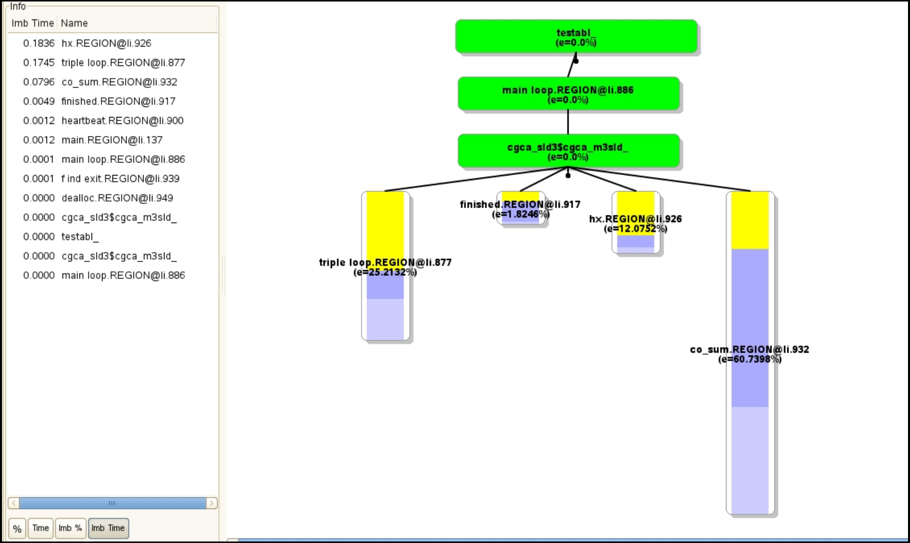
Profiling: divide & conquer reduction



Info

Imb Time	Name
4.7282	triple inner loo.REGIO
1.5368	redand.REGION@li.77
1.1115	heartbeat.REGION@li
0.5445	hx.REGION@li.785
0.2274	finished.REGION@li.7
0.0425	preparation.REGION@
0.0007	finished exit.REGION(
0.0004	whole loop.REGION@
0.0000	finished.REGION@li.7
0.0000	whole loop.REGION@
0.0000	cgca_sld2$cgca_m3s
0.0000	testabi_
0.0000	cgca_sld2$cgca_m3s

Profiling: Cray collective `co_sum` reduction



Info

Imb Time	Name
0.1836	hx.REGION@li.926
0.1745	triple loop.REGION@li.877
0.0796	co_sum.REGION@li.932
0.0049	finished.REGION@li.917
0.0012	heartbeat.REGION@li.900
0.0012	main.REGION@li.137
0.0001	main loop.REGION@li.886
0.0001	f ind exit.REGION@li.939
0.0000	dealloc.REGION@li.949
0.0000	cgca_sld3$cgca_m3sld_
0.0000	testabl_
0.0000	cgca_sld3$cgca_m3sld_
0.0000	main loop.REGION@li.886

Profiling: conclusion

Relative times, %, spent in different parts of the solidification routines and the total run time in seconds.

	`sync all`	d&c	`co_sum`
triple loop	50	60	25
global reduction	35	25	61
serial reduction + I/O	10	10	2
halo exchange	5	5	12
Time, s	35	47	20

- `co_sum` spends twice as long doing the global reduction as the triple loop computation
- With image 1 + `sync all` and with divide & conquer the triple loop takes roughly twice as long as the global reduction.
- `co_sum` approach is twice as fast overall

Coarray I/O

- From [6]:

> *Although a file is not permitted to be connected to more than one image in Fortran 2008, it is expected that a forthcoming Technical Report will define such a facility.*

- The first part is true
- The second part is not so true anymore [4]!
- Multiple writers, multiple files – tried, bad
- Single writer, single file - tried, more on this later
- Multiple writers, single file - not supported
- Few writers, few files - not tried, possibly using MPI-IO

Coarray I/O: single writer, single file

```
if (this_image() .eq. 1) then
    open(unit=iounit, file=fname, access="stream", &
         form = "unformatted", status = "replace")
do coi3 = lcob(3), ucob(3)                     !  Nested loops
  do i3 = lb(3), ub(3)                         !  for writing
    do coi2 = lcob(2), ucob(2)                 !  in correct order
      do i2 = lb(2), ub(2)                     !  from all images.
        do coi1 = lcob(1), ucob(1)             !
          write( unit = iounit )    &          !  Write one column
            coarray(lb(1):ub(1),i2,i3) &       !  at a time
            [coi1, coi2, coi3]                 !  Don't write halos!
          end do
        end do
      end do
    end do
  end do
end if
```

Simple, but extremely expensive!

Further development of the library

- Linking with a parallel FE library - possibly ParaFEM [8] - a CAFE model
- Optimising I/O
- Exploring OpenMP + coarrays for speeding up nested loops
- Looking for good PhD candidates in: metallurgy, physics, numerical methods, HPC, visualisation, etc.
- Freely available under BSD license from http://eis.bris.ac.uk/∼mexas/cgpack

Conclusions

- Fortran coarrays - simple, but flexible and powerful!
- Fortran coarray data structures are recommended for CA modelling.
- Speed-up of 10^3 from 8 to 30k cores was achieved.
- Optimising synchronisation and minimising single image computation are the key to good performance.

Acknowledgments

- This work made use of the facilities of HECToR, the UK's national high-performance computing service, which is provided by UoE HPCx Ltd at the University of Edinburgh, Cray Inc and NAG Ltd, and funded by the Office of Science and Technology through EPSRC's High End Computing Programme.
- This work also used the computational facilities of the Advanced Computing Research Centre, University of Bristol - http://www.bris.ac.uk/acrc/.
- David Henty (EPCC) for the Fortran coarray course
- Reinhold Bader (LRZ) for the Advanced Fortran course
- Reviewers for helpful suggestions!

References

[1] A. Shterenlikht and I. C. Howard. The CAFE model of fracture – application to a TMCR steel. *Fatigue and Fracture of Engineering Materials and Structures*, 29:770–787, 2006.

[2] J. Phillips, A. Shterenlikht, and M. J. Pavier. Cellular automata modelling of nano-crystalline instability. In *Proceedings of the 20th UK Conference of the Association for Computational Mechanics in Engineering, 27-28 March 2012, The University of Manchester, UK*, 2012.

[3] ISO/IEC 1539-1:2010. *Fortran – Part 1: Base language, International Standard*. 2010. http://j3-fortran.org/doc/standing/links/007.pdf.

[4] ISO/IEC JTC1/SC22/WG5 N1983. *Additional Parallel Features in Fortran*. JUL-2013. ftp://ftp.nag.co.uk/sc22wg5/N1951-N2000/N1983.pdf.

[5] I. D. Chivers and J. Sleightholme. Compiler support for the fortran 2003 and 2008 standards, revision 12. *ACM Fortran Forum*, 32(1):8–19, 2013. www.fortranplus.co.uk/resources/fortran_2003_2008_compiler_support.pdf.

[6] M. Metcalf, J. Reid, and M. Cohen. *Modern Fortran explained*. Oxford University Press, 7 edition, 2011.

[7] Cray Inc. *Using Cray Performance Measurement and Analysis Tools*. MAR-2013. http://docs.cray.com/books/S-2376-610/.

[8] ParaFEM - A General Parallel Finite Element Message Passing Library, Research Computing Services, The University of Manchester, UK, 2009. http://www.parafem.org.uk/ (HTML).