Performance Studies of a Co-Array Fortran Applications Versus MPI

Mike Ashworth
Associate Director
Computational Science & Engineering Department
STFC Daresbury Laboratory

mike.ashworth@stfc.ac.uk
Outline

- The SBLI Scalable DNS Application
- The HECToR Cray XE6
- The Co-Array Implementation
- Performance Using Co-Arrays
- Conclusions
Direct Numerical Simulation (DNS) of near-wall turbulent flow

ND Sandham, G Coleman et al, University of Southampton, UK
Partitioned 3D grid with halo exchange

**calculation cost:**
scales as $n^3$

**communication cost:**
scales as $n^2$

**strong scaling:**
increasing $P$
decreasing $n$
comms will dominate
SBLI on Jaguar & JaguarPF

Turbulent channel flow benchmark

Larger problems scale better
Cray XE6

30 cabinets
2816 compute nodes
90,112 cores

Each node has two 16-core AMD 2.3GHz Interlagos processors 32 GB memory
Cray Gemini routers

cce 7.4.4 –hcaf –O3
Migrating A Scientific Application from MPI to Coarrays

John Ashby and John Reid
HPCx Consortium
Rutherford Appleton Laboratory
STFC
UK
Co-Arrays themselves cannot be allocatable as the address needs to be the same on all images

In order to maintain dynamic allocations Ashby & Reid use allocatable components:

```fortran
type co_double_4
  double precision, allocatable, dimension(:, :, :, :):: array
end type co_double_4

type(co_double_4), save, dimension[*] :: q

q%array(i,j,k,n) = ...
```
Ashby & Reid (2008)
Performance on Cray X1

![Graph showing speedup relative to one processor against number of processors. The graph compares different communication methods: linear, MPI, and Co-Array.]
Using the Ashby & Reid code out of the box did not immediately yield the same performance as MPI on the XE6
Co-array options which were tested

**Original:** halo transfer using co-array assignment

**Packed:** halo data packed into contiguous array, transfer using co-array assignment, unpacked

**F90:** co-array assignment using F90 array syntax

**F77:** co-array assignment using F77 DO loops

**Push:** co-array assignment pushes data to remote processor (put); co-array on LHS

**Pull:** co-array assignment pulls data from remote processor (get); co-array on RHS
call sync_all()

\[a[\text{dimxm}(1),\text{dimxm}(2),\text{dimxm}(3)]\%\text{array}(\text{nxp}[\text{procxm}]+1:]
& \quad \text{nxp}[\text{procxm}]+\text{xhalo},::) =
& \quad a\%\text{array}(1:\text{xhalo},::)
\]

call sync_all()

\[PUSH \uparrow\]

\[PULL \downarrow\]

Repeat six times for north, south, east, west, up, down

call sync_all()

\[a\%\text{array}(1-\text{xhalo}:0,::) =
& \quad a[\text{dimxm}(1),\text{dimxm}(2),\text{dimxm}(3)]
& \quad \%\text{array}(\text{nxp}[\text{procxm}]-\text{xhalo}+1:\text{nxp}[\text{procxm}],::)
\]

call sync_all()
Code with packing for a single halo transfer

\[
x_{\text{hsize}} = x_{\text{halo}}(nyd+2*y_{\text{halo}})(nzd+2*z_{\text{halo}})
\]

allocate (ahixm%array(x_{\text{hsize}}),stat=ialloc)
allocate (ahoxm%array(x_{\text{hsize}}),stat=ialloc)
l=0

do k=1-z_{\text{halo}},nzp+z_{\text{halo}}
do j=1-y_{\text{halo}},nyp+y_{\text{halo}}
do i=1,x_{\text{halo}}
 l=l+1
    a_{\text{hoxm}}%array(l) = a%array(i,j,k)
enddo
endo
dendo
call sync_all()

ahixm[procx\text{m}]%array(1:x_{\text{hsize}}) = a_{\text{hoxm}}%array(1:x_{\text{hsize}})
call sync_all()
l=0

do k=1-z_{\text{halo}},nzp+z_{\text{halo}}
do j=1-y_{\text{halo}},nyp+y_{\text{halo}}
do i=1,x_{\text{halo}}
 l=l+1
    a%array(i-x_{\text{halo}},j,k) = ahixm%array(l)
enddo
endo
dendo
deallocate(a_{\text{hoxm}}%array,ahixm%array,ahoxp%array,ahixp%array)
Notes on sync_all

In halo exchange sync_all() could be replaced with sync_images(team), where team is the set of nearest neighbours

However sync_images is only quicker than sync_all on large core counts - see Henty (2012) at this conference

Sync_all is quicker than MPI_Barrier, but **IT NEEDS TO BE**

Sync_all is needed frequently, before and after every halo put/get

A good MPI code doesn’t need MPI_Barrier, it self-synchronises through MPI_Send/MPI_Recv
Performance of Co-Arrays on the Cray XE6 – ON NODE

![Graph showing performance of Co-Arrays on Cray XE6](image)

- MPI
- CAF push f90
- CAF pull f90
- CAF push f77
- CAF pull f77
- CAF packed push
- CAF packed pull

Performance (arbitrary) vs. Number of Cores
Performance of Co-Arrays on the Cray XE6 – OFF NODE

- Get/pull is better than put/push (cf. Henty (2012))
- F90 syntax doesn’t hurt
- Packing helps – a little

Graph showing performance of different methods (MPI, CAF push f90, CAF pull f90, CAF push f77, CAF pull f77, CAF packed push, CAF packed pull) with number of cores on the x-axis and performance on the y-axis.
Does CrayPAT help?

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PE=HIDE</td>
</tr>
<tr>
<td>100.0%</td>
<td>2345</td>
<td>--</td>
<td>--</td>
<td>Total</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80.9%</td>
<td>1897</td>
<td>--</td>
<td>--</td>
<td>USER</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18.6%</td>
<td>436.594</td>
<td>280.41</td>
<td>39.3%</td>
<td>dns3d_</td>
</tr>
<tr>
<td>15.9%</td>
<td>372.070</td>
<td>138.93</td>
<td>27.3%</td>
<td>swap_x_4_</td>
</tr>
<tr>
<td>13.1%</td>
<td>306.207</td>
<td>212.79</td>
<td>41.2%</td>
<td>period_xmb_</td>
</tr>
<tr>
<td>8.3%</td>
<td>195.078</td>
<td>296.92</td>
<td>60.6%</td>
<td>period_z_4_</td>
</tr>
<tr>
<td>7.6%</td>
<td>177.176</td>
<td>24.82</td>
<td>12.3%</td>
<td>rhs_</td>
</tr>
<tr>
<td>3.9%</td>
<td>91.125</td>
<td>94.88</td>
<td>51.2%</td>
<td>bparam_</td>
</tr>
<tr>
<td>2.5%</td>
<td>58.531</td>
<td>11.47</td>
<td>16.4%</td>
<td>pow</td>
</tr>
<tr>
<td>2.1%</td>
<td>49.410</td>
<td>16.59</td>
<td>25.2%</td>
<td>ent_euler_</td>
</tr>
<tr>
<td>1.5%</td>
<td>35.062</td>
<td>14.94</td>
<td>30.0%</td>
<td>dlxi_2$deriv_</td>
</tr>
<tr>
<td>1.4%</td>
<td>33.066</td>
<td>17.93</td>
<td>35.3%</td>
<td>dleta_2$deriv_</td>
</tr>
<tr>
<td>1.2%</td>
<td>27.891</td>
<td>136.11</td>
<td>83.3%</td>
<td>period_x_</td>
</tr>
<tr>
<td>1.2%</td>
<td>27.277</td>
<td>21.72</td>
<td>44.5%</td>
<td>dlz_2$deriv_</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19.1%</td>
<td>447.383</td>
<td>--</td>
<td>--</td>
<td>ETC</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16.0%</td>
<td>374.820</td>
<td>109.18</td>
<td>22.6%</td>
<td>syscall</td>
</tr>
<tr>
<td>2.9%</td>
<td>68.617</td>
<td>31.38</td>
<td>31.5%</td>
<td>__strcmp</td>
</tr>
</tbody>
</table>

CAF push f90
256 cores
Ashby & Reid (2008)
Conclusions on the Cray X1 & X1E

+ Simple assignment statements replace MPI calls
+ No need to pack and unpack data (scope for programming errors)
+ Simpler, shorter, more maintainable code
+ Performance comparable with MPI
- Added indirection through allocatable components
Conclusions on the Cray XE6

- Simple assignment statements replace MPI calls
- No need to pack and unpack data (scope for programming errors)
- Simpler, shorter, more maintainable code
  - Performance not comparable with MPI
  - Added indirection through allocatable components

depends on whether we need packing
Thank you for your attention

Mike Ashworth
mike.ashworth@stfc.ac.uk