First experiences porting a parallel application to a hybrid supercomputer with OpenMP 4.0 device constructs

Alistair Hart

(Cray UK Ltd.)

EPiGRAM

Safe Harbor Statement

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts. These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

Contents of the talk

- A brief introduction to accelerator directives
- Porting a simple example
- Porting the NekBone code

Directive based programming

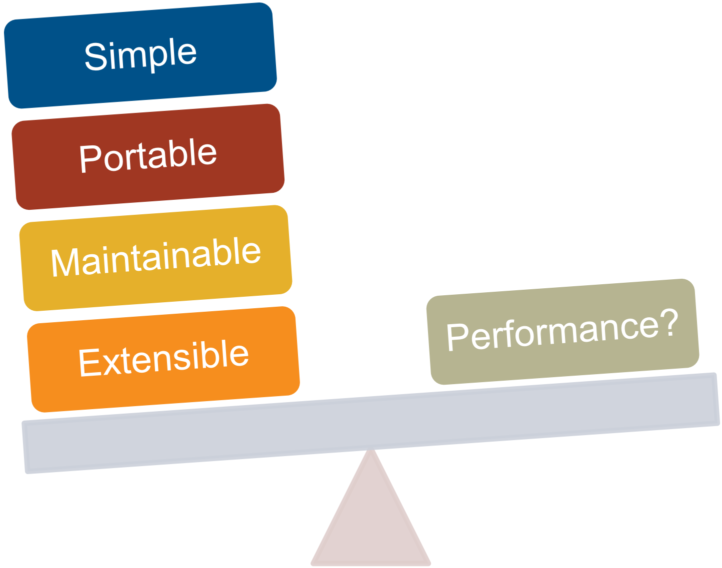
Add directives to code

Compile for CPU as usual

Or compile for accelerator

Why use accelerator directive models?

Positives | **Trade-offs**



Accelerator directives are not new

2007
HMPP
CUDA

2008
OpenCL

2009
PGI Accelerator

2012
OpenACC
Intel LEO

2013
OpenMP 4.0

2014
XcalableACC

2015 (?)
OpenMP 4.1
OpenACC 2.5

A simple example

Memory management

Data movement

```
#pragma omp target teams \
            distribute
for (j=0; j<N; j++)
 for (i=1; i<N-1; i++)
  c[j][i]=a[j][i]+b[j][i];
```

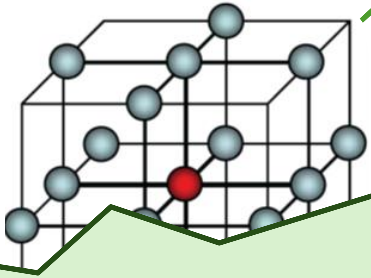
Loop scheduling

Synchronisation

target teams distribute

- **target creates an offload kernel**
- **teams creates a "league" of "threadteams"**
  - Compiler chooses number of teams and threads per team
  - Can over-ride this with optional clauses
- **distribute partitions loop iterations over threads**
  - Multiple loops can be partitioned (unlike host OpenMP)
- **CCE v8.4 targets Nvidia GPUs**
  - distribute fully partitions iterations over all levels of parallelism
  - "distribute parallel simd" not needed
    - distribute simd can be used for tuning

The Himeno code



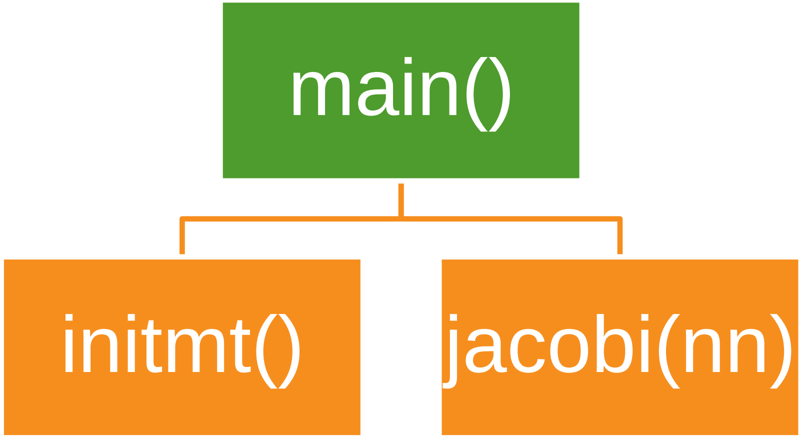
- Poisson solver
  - 3d
  - iterative loop
- Memory intensive
  - 19pt stencil
  - b/w bound
- Small
  - 250 lines
  - Fortran, C

1thread: 3.3 Gflops
10 threads: 11.1 Gflops

Acceleration planning

- **Understand call tree**
- **Inspection or profiling**
  - e.g. CrayPAT

```
Table 1:  Calltree View with Callsite Line Numbers

  Time% |     Time | Calls |Calltree

 100.0% | 4.432070 |    -- |Total
|-------------------------------------------------------------
|  91.9% | 4.070905 |    -- |main_:himeno_F_v0.F90:line.171
|  91.9% | 4.070905 |   1.0 | jacobi_:himeno_F_v0.F90:line.241
|   4.9% | 0.215086 |    -- |main_:himeno_F_v0.F90:line.118
|   4.9% | 0.215086 |   1.0 | initmt_:himeno_F_v0.F90:line.189
|   3.1% | 0.136999 |    -- |main_:himeno_F_v0.F90:line.150
|   3.1% | 0.136999 |   1.0 | jacobi_:himeno_F_v0.F90:line.241
|=============================================================
```



Understanding the loops

- Loopnest dominated?
- Iteration counts
- Granularity

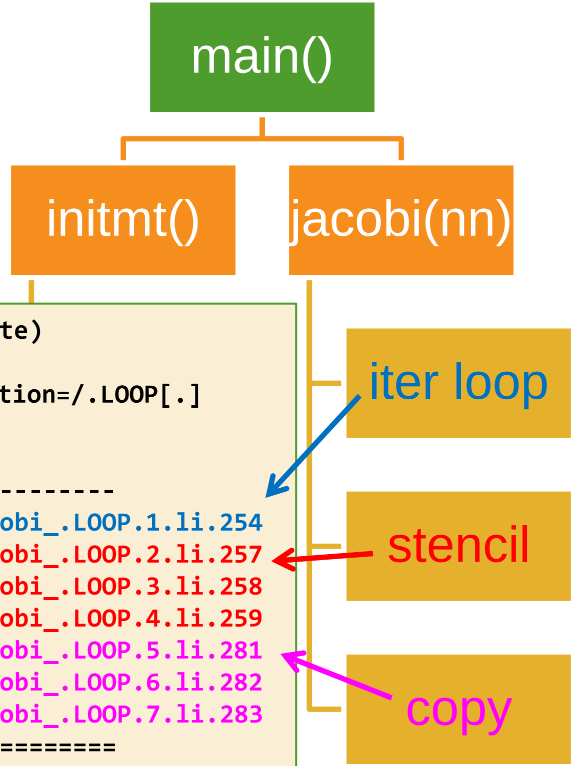


Table 2: Inclusive and Exclusive Time in Loops (from -hprofile_generate)

Loop Incl Time%	Loop Incl Time	Time (Loop Adj.)	Loop Hit	Loop Trips Avg	Loop Trips Min	Loop Trips Max	Function=/.LOOP[.]
95.3%	3.957659	0.000022	2	51.5	3	100	jacobi_.LOOP.1.li.254
81.8%	3.397517	0.000653	103	126.0	126	126	jacobi_.LOOP.2.li.257
81.8%	3.396864	0.031151	12,978	126.0	126	126	jacobi_.LOOP.3.li.258
81.1%	3.365713	3.365713	1,635,228	254.0	254	254	jacobi_.LOOP.4.li.259
13.5%	0.560120	0.000175	103	126.0	126	126	jacobi_.LOOP.5.li.281
13.5%	0.559945	0.080416	12,978	126.0	126	126	jacobi_.LOOP.6.li.282
11.5%	0.479529	0.479529	1,635,228	254.0	254	254	jacobi_.LOOP.7.li.283

Accelerating the stencil loopnest

```
DO k = 2,kmax-1
 DO j = 2,jmax-1
  DO i = 2,imax-1
   s0 = a(i,j,k,1) *   p(i+1,j  ,k  ) &
      + a(i,j,k,2) *   p(i  ,j+1,k  ) &
      + a(i,j,k,3) *   p(i  ,  j,k+1) &
      + b(i,j,k,1) * ( p(i+1,j+1,k  ) - p(i+1,j-1,k  ) &
                     - p(i-1,j+1,k  ) + p(i-1,j-1,k  ) ) &
      + b(i,j,k,2) * ( p(i  ,j+1,k+1) - p(i  ,j-1,k+1) &
                     - p(i  ,j+1,k-1) + p(i  ,j-1,k-1)) &
      + b(i,j,k,3) * ( p(i+1,j  ,k+1) - p(i-1,j  ,k+1) &
                     - p(i+1,j  ,k-1) + p(i-1,j  ,k-1)) &
      + c(i,j,k,1) *   p(i-1,j  ,k  ) &
      + c(i,j,k,2) *   p(i  ,j-1,k  ) &
      + c(i,j,k,3) *   p(i  ,j  ,k-1) &
      + wrk1(i,j,k)

   ss = ( s0 * a(i,j,k,4) - p(i,j,k) ) * bnd(i,j,k)
   gosa = gosa + ss*ss
   wrk2(i,j,k) = p(i,j,k) + omega * ss
  ENDDO
 ENDDO
ENDDO
```

Accelerating the stencil loopnest

```
!$omp target teams distribute reduction(+:gosa) private(s0,ss)
DO k = 2,kmax-1
 DO j = 2,jmax-1
  DO i = 2,imax-1
   S0 = a(i,j,k,1) *   p(i+1,j  ,k  ) &
      + a(i,j,k,2) *   p(i  ,j+1,k  ) &
      + a(i,j,k,3) *   p(i  ,  j,k+1) &
      + b(i,j,k,1) * ( p(i+1,j+1,k  ) - p(i+1,j-1,k  ) &
                     - p(i-1,j+1,k  ) + p(i-1,j-1,k  ) ) &
      + b(i,j,k,2) * ( p(i  ,j+1,k+1) - p(i  ,j-1,k+1) &
                     - p(i  ,j+1,k-1) + p(i  ,j-1,k-1)) &
      + b(i,j,k,3) * ( p(i+1,j  ,k+1) - p(i-1,j  ,k+1) &
                     - p(i+1,j  ,k-1) + p(i-1,j  ,k-1)) &
      + c(i,j,k,1) *   p(i-1,j  ,k  ) &
      + c(i,j,k,2) *   p(i  ,j-1,k  ) &
      + c(i,j,k,3) *   p(i  ,j  ,k-1) &
      + wrk1(i,j,k)

   ss = ( s0 * a(i,j,k,4) - p(i,j,k) ) * bnd(i,j,k)
   gosa = gosa + ss*ss
   wrk2(i,j,k) = p(i,j,k) + omega * ss
  ENDDO
 ENDDO
ENDDO
!$omp end target teams distribute
```

Accelerating the stencil loopnest

```
+ 1 G----------< !$omp target teams distribute reduction(+:gosa) private(s0,ss)
  1 G g--------< DO k = 2,kmax-1
+ 1 G g b------<  DO j = 2,jmax-1
  1 G g b gb---<   DO i = 2,imax-1
  1 G g b gb        S0 = a(i,j,k,1) *   p(i+1,j  ,k  ) &
  1 G g b gb           + a(i,j,k,2) *   p(i  ,j+1,k  ) &
  1 G g b gb           + a(i,j,k,3) *   p(i  ,  j,k+1) &
  1 G g b gb           + b(i,j,k,1) * ( p(i+1,j+1,k  ) - p(i+1,j-1,k  ) &
  1 G g b gb                          - p(i-1,j+1,k  ) + p(i-1,j-1,k  ) ) &
  1 G g b gb           + b(i,j,k,2) * ( p(i  ,j+1,k+1) - p(i  ,j-1,k+1) &
  1 G g b gb                          - p(i  ,j+1,k-1) + p(i  ,j-1,k-1)) &
  1 G g b gb           + b(i,j,k,3) * ( p(i+1,j  ,k+1) - p(i-1,j  ,k+1) &
  1 G g b gb                          - p(i+1,j  ,k-1) + p(i-1,j  ,k-1)) &
  1 G g b gb           + c(i,j,k,1) *   p(i-1,j  ,k  ) &
  1 G g b gb           + c(i,j,k,2) *   p(i  ,j-1,k  ) &
  1 G g b gb           + c(i,j,k,3) *   p(i  ,j  ,k-1) &
  1 G g b gb           + wrk1(i,j,k)
  1 G g b gb
  1 G g b gb        ss = ( s0 * a(i,j,k,4) - p(i,j,k) ) * bnd(i,j,k)
  1 G g b gb        gosa = gosa + ss*ss
  1 G g b gb        wrk2(i,j,k) = p(i,j,k) + omega * ss
  1 G g b gb--->   ENDDO
  1 G g b------>  ENDDO
  1 G g--------> ENDDO
  1 G----------> !$omp end target teams distribute
```

Accelerating the stencil loopnest

```
272.  + 1 G----------< !$omp target teams distribute reduction(+:gosa) private(s0,ss)  map(from:wrk2)
273.    1 G g--------< DO k = 2,kmax-1
274.  + 1 G g b------<  DO j = 2,jmax-1
275.    1 G g b gb---<   DO i = 2,imax-1
276.    1 G g b gb         S0 = a(i,j,k,1) *   p(i+
277.    1 G g b gb            + a(i,j,k,2) *   p(i
278.    1 G g b gb            + a(i,j,k,3) *   p(i
279.    1 G g b gb            + b(i,j,k,1) * ( p(i+
280.    1 G g b gb                           - p(i-
281.    1 G g b gb            + b(i,j,k,2) * ( p(i
282.    1 G g b gb                           - p(i
283.    1 G g b gb            + b(i,j,k,3) * ( p(i+
284.    1 G g b gb                           - p(i+
285.    1 G g b gb            + c(i,j,k,1) *   p(i-
286.    1 G g b gb            + c(i,j,k,2) *   p(i
287.    1 G g b gb            + c(i,j,k,3) *   p(i
288.    1 G g b gb            + wrk1(i,j,k)
289.    1 G g b gb
290.    1 G g b gb         ss = ( s0 * a(i,j,k,4) -
291.    1 G g b gb         gosa = gosa + ss*ss
292.    1 G g b gb         wrk2(i,j,k) = p(i,j,k) +
293.    1 G g b gb--->   ENDDO
294.    1 G g b------>  ENDDO
295.    1 G g-------->  ENDDO
296.    1 G----------> !$omp end target teams dist
```

```
ftn-6405 ftn: ACCEL File = himeno_F_v1.F90, Line = 272
  A region starting at line 272 and ending at line 296 was placed on
the accelerator.

ftn-6418 ftn: ACCEL File = himeno_F_v1.F90, Line = 272
  If not already present: allocate memory and copy whole array "p"
to accelerator, free at line 296 (acc_copyin).

Similar messages for a,b,c,bnd,wrk1.

ftn-6420 ftn: ACCEL File = himeno_F_v1.F90, Line = 272
  If not already present: allocate memory and copy wh    array
"wrk2" to accelerator, copy back a   line 29

ftn-6430 ftn: ACCEL File = him
  A loop starting at line 27                           ead
blocks.

ftn-6049 ftn: SCALAR File = himeno_  v1.F90, Line = 274
  A loop starting at line 274 was blocked with block size 8.

ftn-6430 ftn: ACCEL File = himeno_F_v1.F90, Line = 275
  A loop starting at line 275 was partitioned across the 128 threads
within a threadblock.
```

~~3.3 Gflops~~

1.5 Gflops

Runtime feedback: CRAY_ACC_DEBUG

```
ACC: Start transfer 14 items from himeno_F_v1.F90:158
ACC:       allocate, copy to acc 'p' (34213896 bytes)
ACC:       allocate 'wrk2' (34213896 bytes)
ACC:       allocate, copy to acc 'imax' (4 bytes)

ACC:       allocate reusable <internal> (8 bytes)
ACC:       allocate reusable, copy to acc <internal> (4 bytes)
ACC:       allocate reusable <internal> (1008 bytes)
ACC: End transfer (to acc 444780672 bytes, to host 0 bytes)

ACC: Execute kernel main_$ck_L158_1 blocks:126 threads:128
                  async(auto) from himeno_F_v1.F90:158
ACC: Wait async(auto) from himeno_F_v1.F90:158

ACC: Start transfer 14 items from himeno_F_v1.F90:158
ACC:       free 'p' (34213896 bytes)

ACC:       copy to host, free 'wrk2' (34213896 bytes)

ACC:       copy to host, done reusable <internal> (8 bytes)
ACC:       done reusable <internal> (4 bytes)
ACC:       done reusable <internal> (0 bytes)
ACC: End transfer (to acc 0 bytes, to host 34213904 bytes)
```

Similar messages for a,b,c,bnd,wrk1

Similar messages for jmax,kmax,omega

Similar messages for a,b,c,bnd,wrk1, imax,jmax,kmax,omega

Runtime feedback: CRAY_ACC_DEBUG

```
ACC: Start transfer 14 items from himeno_F_v1.F90:158
ACC:       allocate, copy to acc 'p' (34213896 bytes)
ACC:       allocate
ACC:       allocate,

ACC:       allocate 
ACC:       allocate 
ACC:       allocate 
ACC: End transfer (t

ACC: Execute kernel 

ACC: Wait async(auto

ACC: Start transfer 
ACC:       free 'p' 

ACC:       copy to h

ACC:       copy to h
ACC:       done reus
ACC:       done reus
ACC: End transfer (t
```

Similar messages for

Bytes transfrd	variable	dirn	no. xfrs	mean size
14096125152	a	acc	103	136855584
10572093864	c	acc	103	102641688
10572093864	b	acc	103	102641688
3524031288	wrk2	host	103	34213896
3524031288	wrk1	acc	103	34213896
3524031288	p	acc	103	34213896
3524031288	bnd	acc	103	34213896
824	omega	acc	103	8
412	kmax	acc	103	4
412	jmax	acc	103	4
412	imax	acc	103	4

Total bytes transferred: 49336440092

A first data region

```
!$omp target data map(tofrom:p) &
!$omo             map(to:a,b,c,wrk1,bnd) &
!$omp             map(alloc:wrk2)
   iter_loop: DO loop = 1,nn

    gosa = 0d0

!$omp target teams distribute &
!$omp   reduction(+:gosa) private(s0,ss) map(from:wrk2)
<stencil: p(:,:,:) -> wrk2(:,:,:)>

!$omp target teams distribute
<copy: wrk2(:,:,:) -> p(:,:,:)>

   ENDDO iter_loop

!$omp end target data
```

~~3.3 Gflops~~

~~1.5 Gflops~~

27.0 Gflops

COMPUTE | STORE | ANALYZE

An outer data region

```
!$omp target data map(alloc:p,a,b,c,bnd,wrk1,wrk2)

CALL initmt ! initialisation
<two target regions>

CALL jacobi(nn,gosa) ! calibration
<two target regions, one data region>

cpu0 = gettime()
CALL jacobi(nn,gosa) ! performance
cpu1 = gettime()

!$omp end target data
```

~~3.3 Gflops~~
~~1.5 Gflops~~
~~27.0 Gflops~~
32.5 Gflops
(3x CPU)

An outer data region: performance feedback

```
Table 3:  Time and Bytes Transferred for Accelerator Regions

   Host | Host |  Acc | Acc Copy | Acc Copy | Events |Calltree
  Time% | Time | Time |       In |      Out |        | Thread=HIDE
        |      |      | (MBytes) | (MBytes) |        |

 100.0% | 0.01 | 0.01 |     0.00 |     0.00 |     34 |Total
|------------------------------------------------------------------------------
| 100.0% | 0.01 | 0.01 |     0.00 |     0.00 |     34 |main_
|        |      |      |          |          |        | main_.ACC_DATA_REGION@li.163
3  99.9% | 0.01 | 0.01 |     0.00 |     0.00 |     30 |  main_.ACC_REGION@li.163
||||---------
4|||  94.4% |
4|||   4.3% |
|============
```

```
Bytes transfrd              variable dirn   no. xfrs        mean size

           828                  kmax  acc        207               4
           828                  jmax  acc        207               4
           828                  imax  acc        207               4
           824                 omega  acc        103               8

Total bytes transferred:  3308
```

NekBone

EPiGRAM Project

3 years to Oct.16

www.epigram-project.eu

EPiGRAM

6 partners

€3M

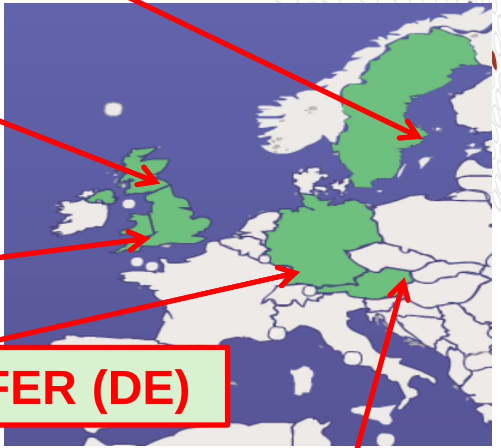
KTH (SE)

EPCC (UK)

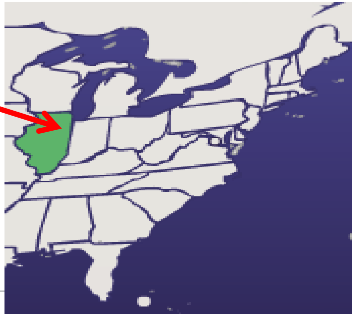
CRAY (UK)

FRAUNHOFER (DE)

TUW (AT)



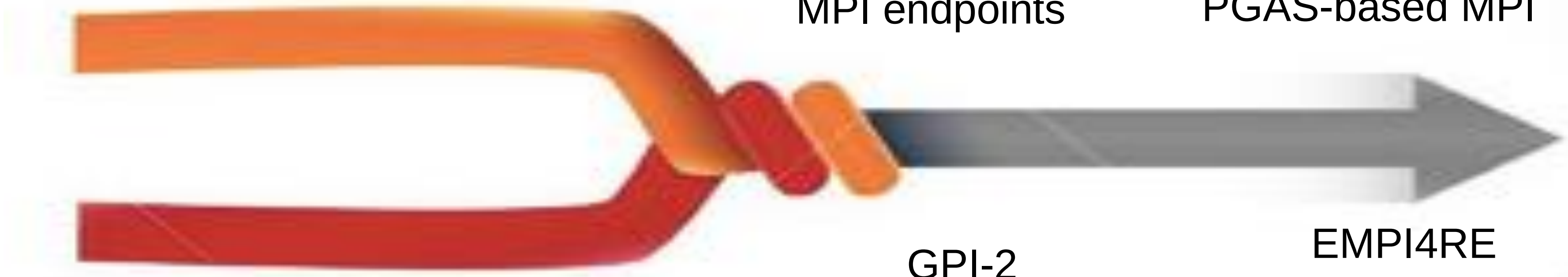
UIUC (USA)



CREST

EPiGRAM Integrated Vision

MPI:
- Persistent Collectives
- Neighborhood collectives



MPI endpoints

PGAS-based MPI

GPI-2:
- Fast RDMA; Fault-tolerance

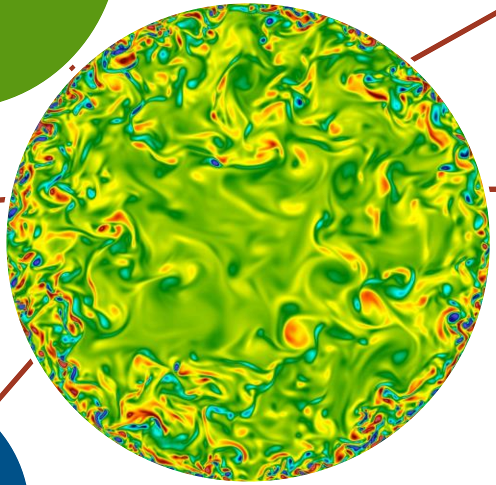
Diverse memory spaces:
- OpenACC, OpenMP, …

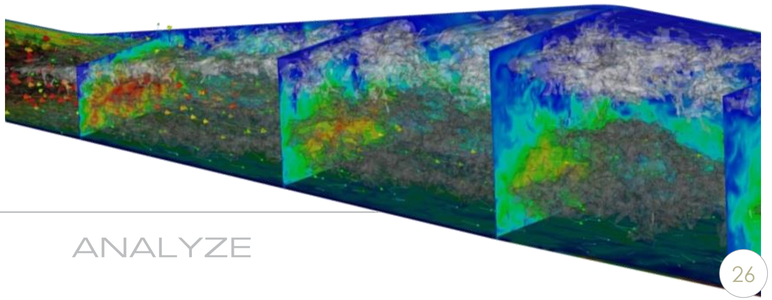
GPI-2
Isolation of library

EMPI4RE
EPiGRAM MPI for
RESEARCH

Nek5000

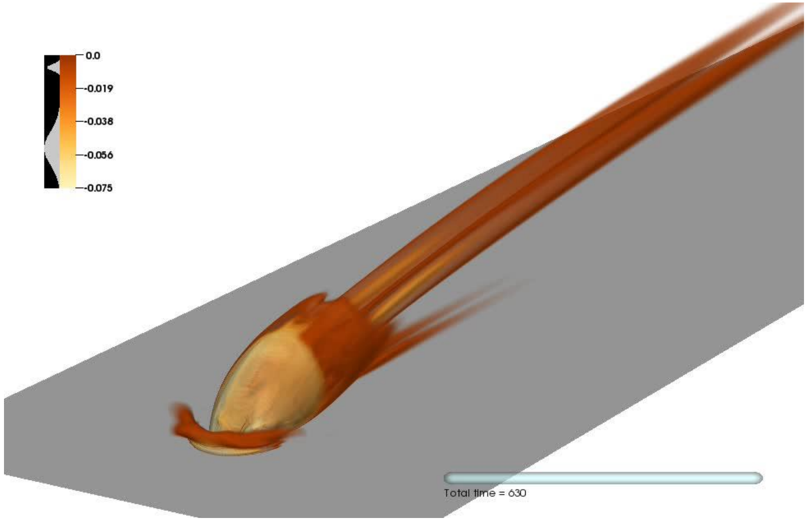
- astrophysics
- combustion
- reactor core
- oceanography
- vascular flow





Nek5000 and NekBone

- **Nek5000: CFD code**
  - simulates incompressible fluids.
  - solves Navier-Stokes equations
  - semi-spectral element method.
- **~70k lines of code:**
  - 90% in Fortran 77
  - 10% in C (comms).
  - Parallelised with MPI
- **NekBone mini-app**
  - captures this in 11k lines
  - EPiGRAM simplified comms



Nek5000 simulation of a cross-flow on a flat panel

NekBone

- **Small testcase, for functionality/quick development**
  - Elements per PE: Nelt=32
  - Spectral order: N=10
- **Computational load:**
  - Multiple (Nelt), small (NxN) matrix-matrix multiplications
- **Configuration:**
  - 8 nodes, one MPI rank per node (no OpenMP threading)
- **Code reports performance (and correctness)**
  - Intel IVB CPU: 40.2 GFlops

Nekbone porting

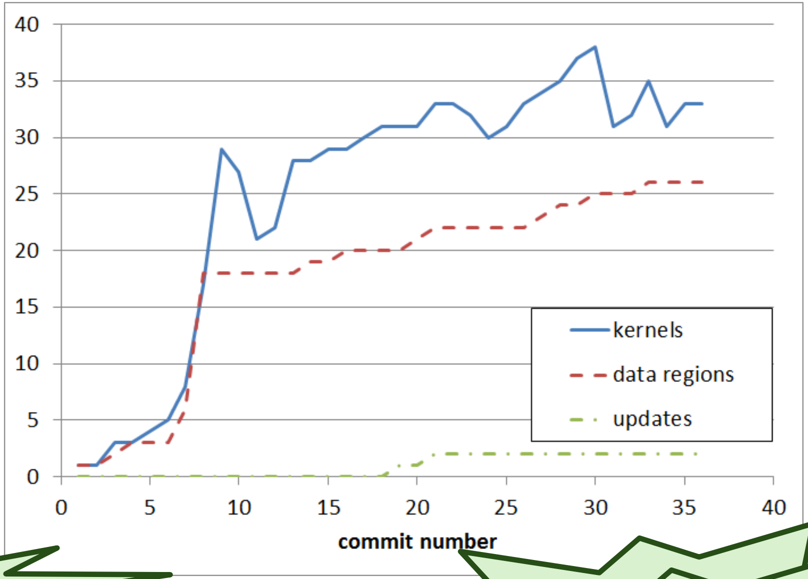
- **Proceeds in same way as for Himeno**
  - Code already has performance counters and correctness checks
  - Profile to understand the hotspots, calltree and loopnests
  - Begin with target regions on leaf nodes
  - Expand data regions up the calltree

- **Parallel ports: OpenMP device constructs; OpenACC**
  - Maintained audit trail in git log:
    - Correctness, directive count, performance, rough profile

How many directives?

OpenMP

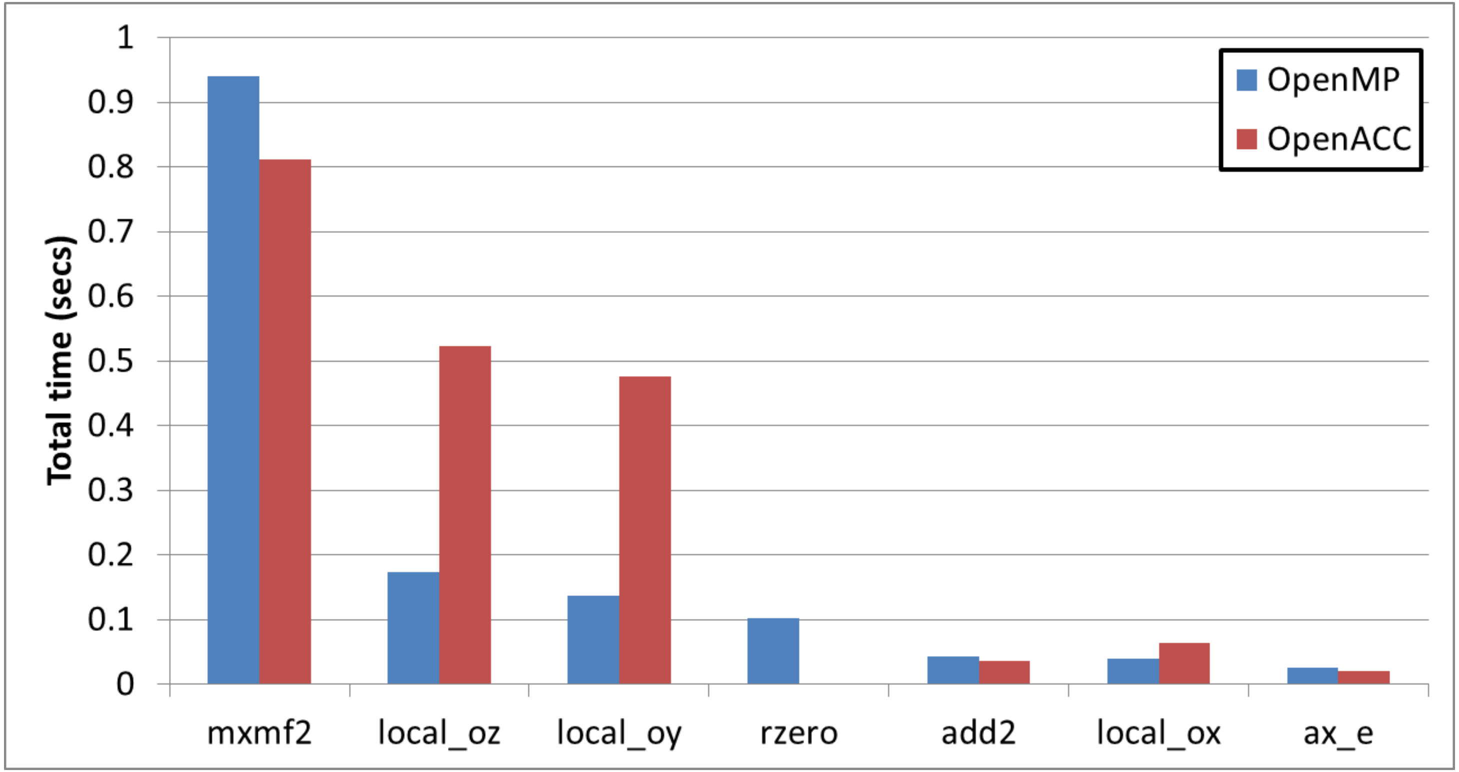
OpenACC



420 Mflops

G2G MPI

Performance of NekBone kernels



mxmf2

```
G---------< !$omp target teams distribute
G g-------< DO j = 1,n3
G g g-----<  DO i = 1,n1
G g g         c(i,j) = 0
G g g r4--<   DO k = 1,n2
G g g r4       c(i,j) = c(i,j) + a(i,k)*b(k,j)
G g g r4-->   ENDDO
G g g----->  ENDDO
G g-------> ENDDO
G---------> !$omp end target teams d
```

```
G---------< !$acc parallel loop
G g-------< DO j = 1,n3
G g g-----<  DO i = 1,n1
G g g         c(i,j) = 0
G g g r4--<   DO k = 1,n2
G g g r4       c(i,j) = c(i,j) + a(i,k)*b(k,j)
G g g r4-->   ENDDO
G g g----->  ENDDO
G g-------> ENDDO
G---------> !$acc end parallel loop
```

local_oz

```
G-----------< !$omp target teams distribute
G             !$omp&  private(e_back,e_front)
G g---------< do e_z =1,mz-1
G g C-------<  do e_y = 1,my
G g C C-----<   do e_x = 1,mx
```

```
G-----------< !$omp target teams distribute collapse(3)
G             !$omp&  private(e_back,e_front)
G C---------< do e_z =1,mz-1
G C C-------<  do e_y = 1,my
G C C g-----<   do e_x = 1,mx
G C C g          e_back = e_x + mx*(e_y-1)+mx*my*(e_z-1)
G C C g          e_front = e_back+mx*my
G C C g r2--<    do i=1,n_shared
G C C g r2        w(l_face_z_back(i),e_back)= &
G C C g r2           w(l_face_z_back(i),e_back)+ &
G C C g r2           w(l_face_z_front(i),e_front)
G C C g r2        w(l_face_z_front(i),e_front) = &
G C C g r2           w(l_face_z_back(i),e_back)
G C C g r2-->    enddo
G C C g----->   enddo
G C C------->  enddo
G C---------> enddo
G----------->
```

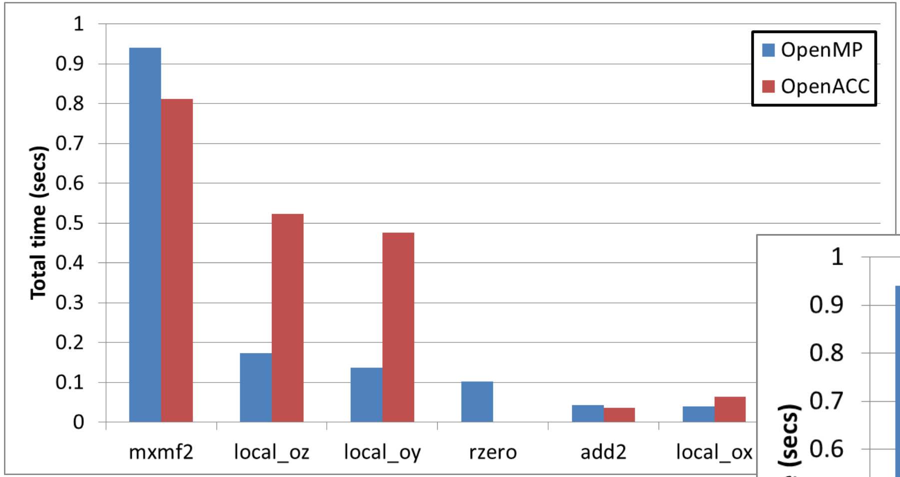
863µs→147µs

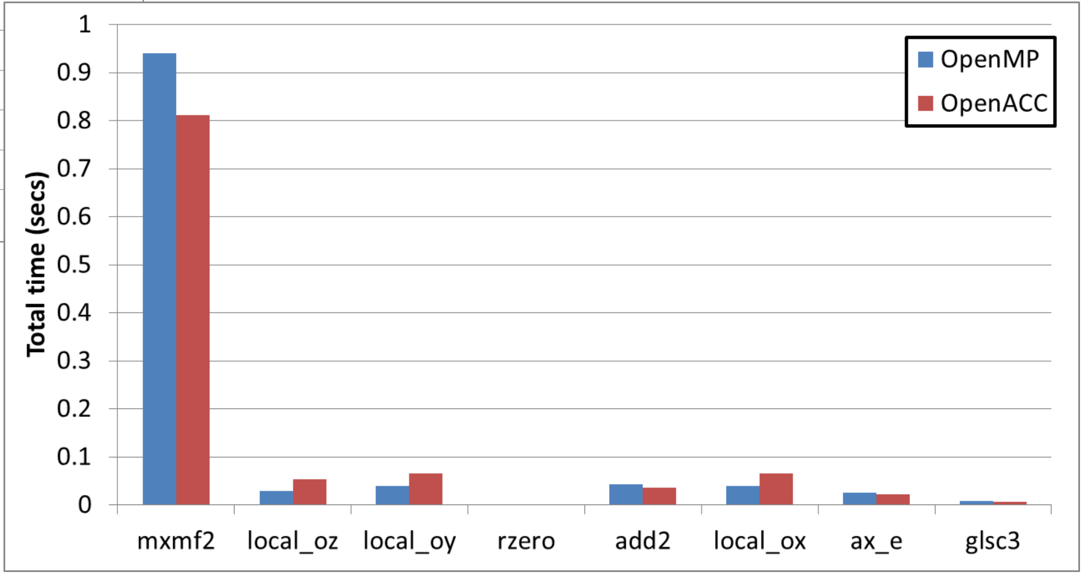
```
G-----------< !$acc parallel loop

G g---------< do e_z =1,mz-1
G g 3-------<  do e_y = 1,my
G g 3 4-----<   do e_x = 1,mx
```

```
G-----------< !$acc parallel loop collapse(3)

G C---------< do e_z =1,mz-1
G C C-------<  do e_y = 1,my
G C C g-----<   do e_x = 1,mx
G C C g          e_back = e_x + mx*(e_y-1)+mx*my*(e_z-1)
G C C g          e_front = e_back+mx*my
G C C g 5--<     do i=1,n_shared
G C C g 5         w(l_face_z_back(i),e_back)= &
G C C g 5            w(l_face_z_back(i),e_back)+ &
G C C g 5            w(l_face_z_front(i),e_front)
G C C g 5         w(l_face_z_front(i),e_front) = &
G C C g 5            w(l_face_z_back(i),e_back)
G C C g 5-->     enddo
G C C g----->   enddo
G C C------->  enddo
G C---------> enddo
G----------->
```

2592µs→261µs

Performance after tuning





Conclusions

- **Directives offer productive performance-portability**
  - NekBone: 1 directive per 160 lines of code
- **OpenMP device constructs are mature prog. model**
  - v4.0 offers rich feature set; more coming v4.1
- **OpenMP device constructs can (should) be performant**
  - CCE: default comparable and often better than OpenACC
  - Fewer tuning clauses, but does not appear to be a problem
- **Straightforward migration path: OpenACC ⇔ OpenMP**

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, and URIKA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.