An Accelerated, Distributed Hydro Code with MPI and OpenACC

Andy Herdman & Wayne Gaudin

Andy.Herdman@awe.co.uk
www.awe.co.uk

Background

- Changing HPC landscape, future uncertain
  - Multi core: slower clock, but more of them
  - Many core: GPUs, MIC, Fusion
  - Massive scalability: Sequoia – 1.6 million cores
- Issues for current code base
  - Programming model? MPI, OpenMP, OpenCL, CUDA, Clik, etc, etc
  - Re-write not an option!
    - Decades of manpower (c.f. MPP – but we got 3D from this!)
    - Hardware temporary, software permanent
  - Effort vs. gains

How to Investigate Options?

- Current code base
  - Classified
  - Big! (~0.5M Lines of Code (LoC))
  - Complex – multi physics, utilities / libraries
  - Mostly Fortran
  - Flat MPI

How to Investigate Options?

- Benchmarks
  - Benchmarks of current algorithms
  - Big(ish) (~90k LoC – comms package 46k)
  - Complex
  - Flat MPI
  - Inefficient tool to evaluate software techniques
    - Turnaround taking too long (~18 months CUDA/OpenCL)

How to Investigate Options?

- Lightweight, but representative application
  - Written with computer science in mind
  - Small(ish) (~4.5k LoC)
  - Amenable to range of programming methods and hardware
    - No “cut-offs”, etc
  - Hence CloverLeaf “mini-app”
  - Open source via Mantevo Project

CloverLeaf

2D Structured Hydrodynamic “mini-app”

- Explicit solution to the compressible Euler equations
- Finite volume predictor/corrector lagrangian step followed by an advective remap
- Single material
- Common base to all interested physics models (they all do hydro!)

- Simplest physics for computer science
- Already know hydro scales to 10k’s way parallel
- If methodology fails for hydro scheme, or is difficult to get performance; then other physics models are going to be more difficult

Written with computer science in mind

- Simple Fortran “kernels”
- Minimised loop logic (Reduced error checking. We know we’re running robust problems)
- Kernels are lowest level of compute – don’t call subroutines

- No derived types
- Minimal Pointers
- No Array Syntax
- ~4500 LoC
- Amenable to range of programming methods: MPI, OpenMP, OpenACC, CUDA, OpenCL, PGAS, etc.

Programming Models: OpenACC

- Directives provide high level approach
- Based on original source code (e.g. Fortran, C)
  - Easier to maintain/port/extend
  - Users with OpenMP experience find it a familiar programming model
  - Compiler handles repetitive boilerplate code (cudaMalloc, cudaMemcpy, etc.)
  - Compiler handles default scheduling: user can step in with clauses where needed

CloverLeaf: OpenACC

- Worked with Cray's Exascale Research Initiative in Europe since late 2010
  - Access to early HW (Puffin) and SW (proposed OpenMP extensions)
  - Direct Fortran interface
  - Easy of implementation
    - 2 months to write CloverLeaf from scratch and develop fully resident OpenACC version
  - Summary of OpenACC directives:
    - 14 unique kernels
    - 25 ACC DATA constructs
    - 121 ACC PARALLEL + LOOP regions
    - 4 REDUCTION LOOPS
    - 12 ASYNC
    - 4 UPDATE HOST
    - 4 UPDATE DEVICE

Test Problem Definition

- Asymmetric test problem.
- Regions of ideal gas at differing initial densities and energies cause shock wave to be generated
- Gives rise to shock front which penetrates low density region
- 0.25 million cells
  - relatively quick turnaround
  - Long enough to see compute as main work load

Chilean Pine, AWE's Cray XK6

40 Compute nodes, each:

- 1 x AMD 16-core Interlagos
  - 2.1 GHz
  - 32GB DDR3 1600 MHz
- 1 NVIDIA X2090
  - 1.16 GHz
  - 6GB GDDR5
  - PCIe 2.0

- PrgEnv-cray 4.0.36
  - CCE (8.0.2 to 8.1.0.157)
  - Cuda 4.0.17a
- Craype-hugepages2M
- Craype-accel-nvidia20

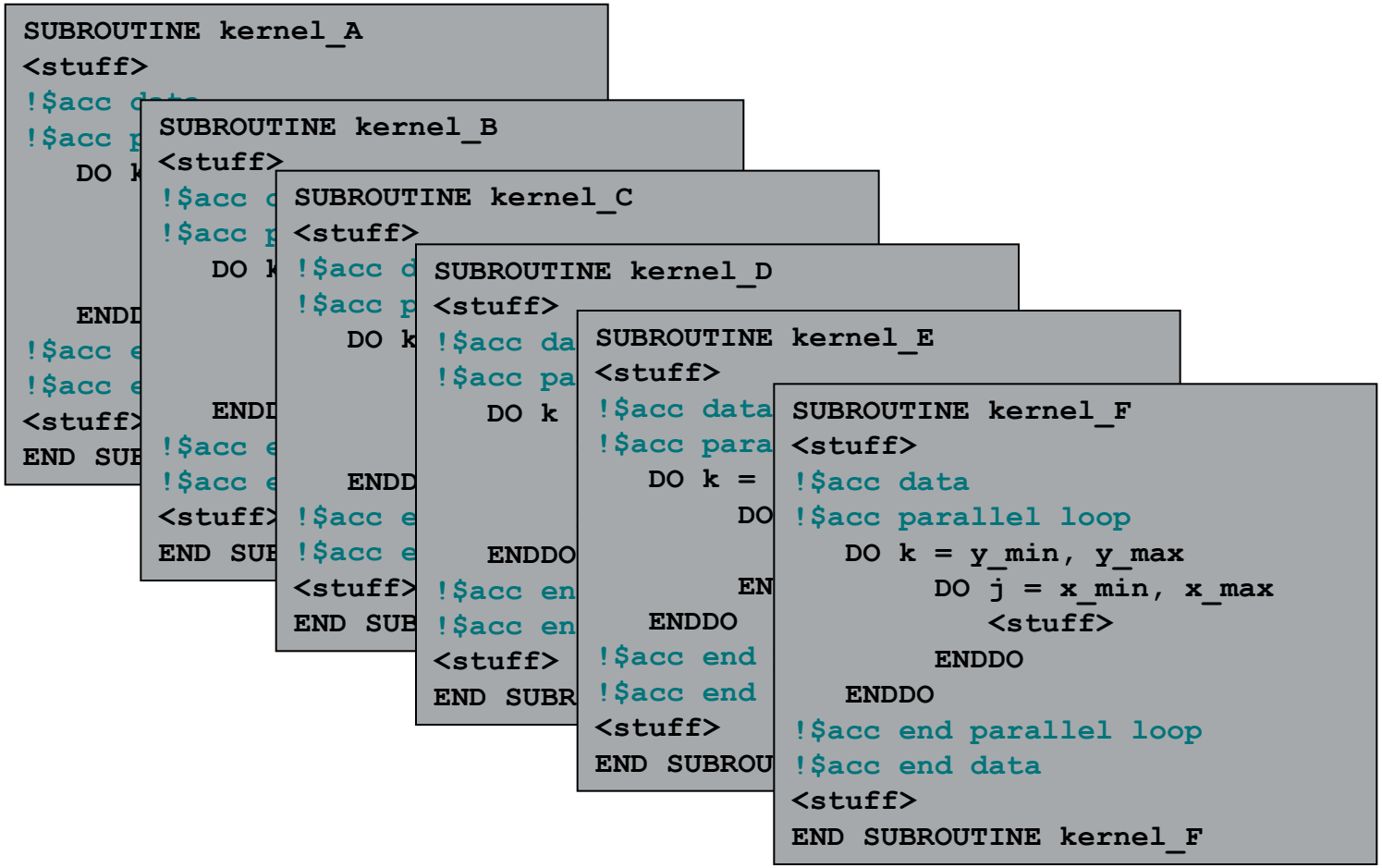
OpenACC: Steps

- Profile “hot spots”
- Accelerate on a kernel by kernel basis
- Accelerate all kernels
- Make entire code resident on device
- Effect of problem size
- Compare with MPI/OpenMP hybrid version
- Hybrid MPI/OpenACC implementation
- Optimisations

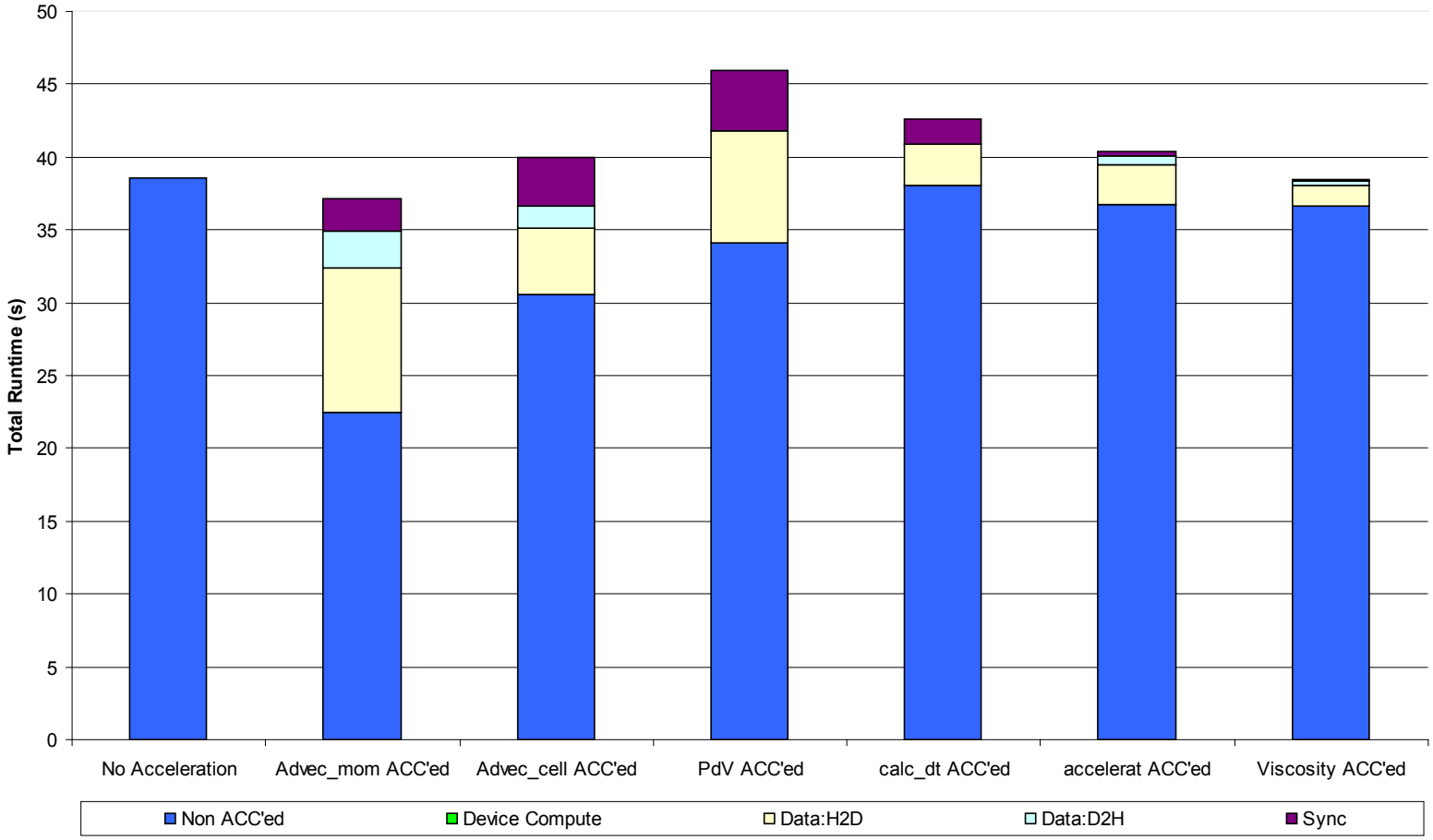
“Hot Spots”

% of Runtime	Routine
41.79	advec_mom
20.54	advec_cell
12.72	pdv
9.06	calc_dt
5.32	accelerate
5.24	viscosity

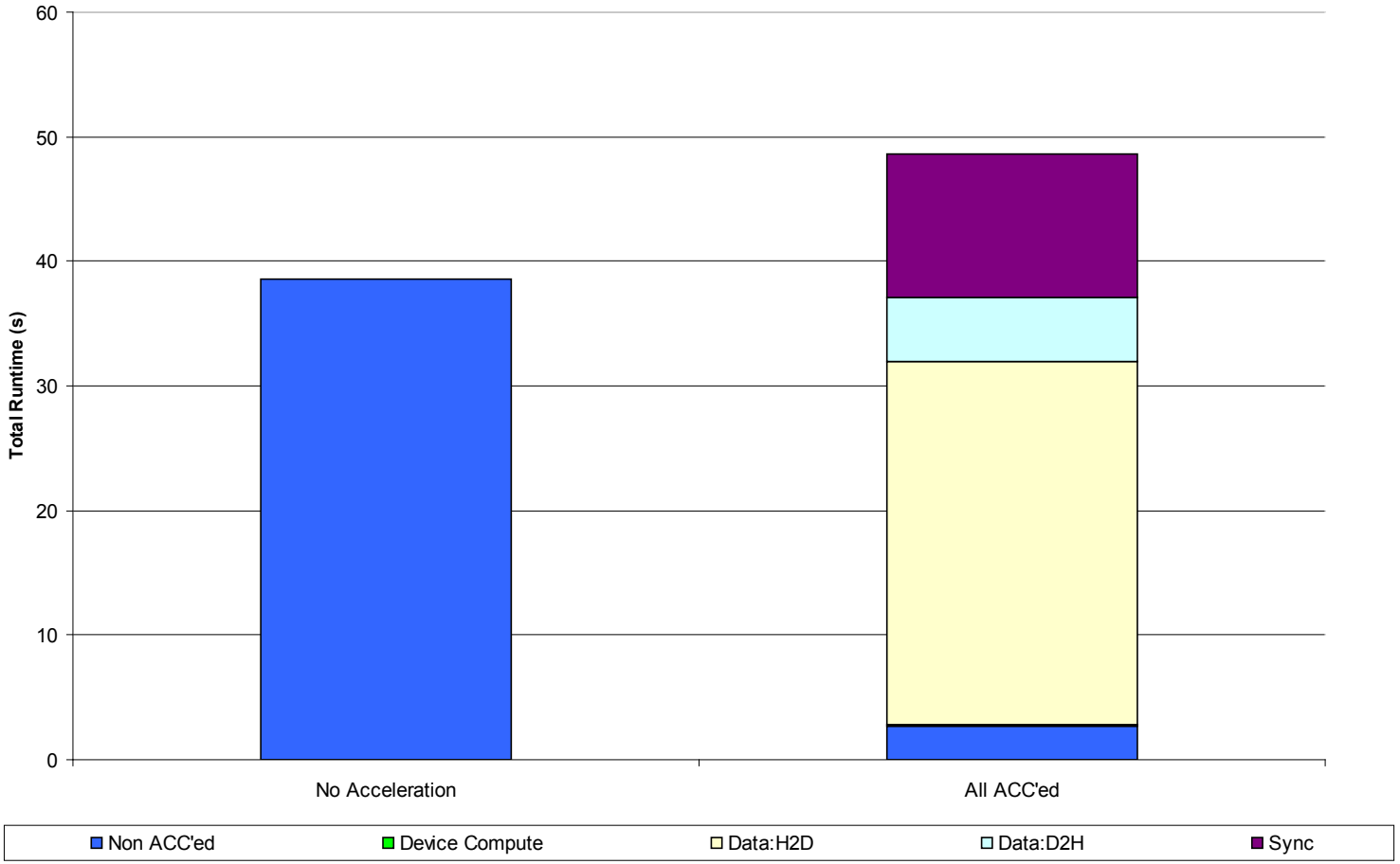
Accelerate Individual Kernels



Accelerate Individual Kernels



Accelerate Multiple Kernels



Fully Resident on Accelerator

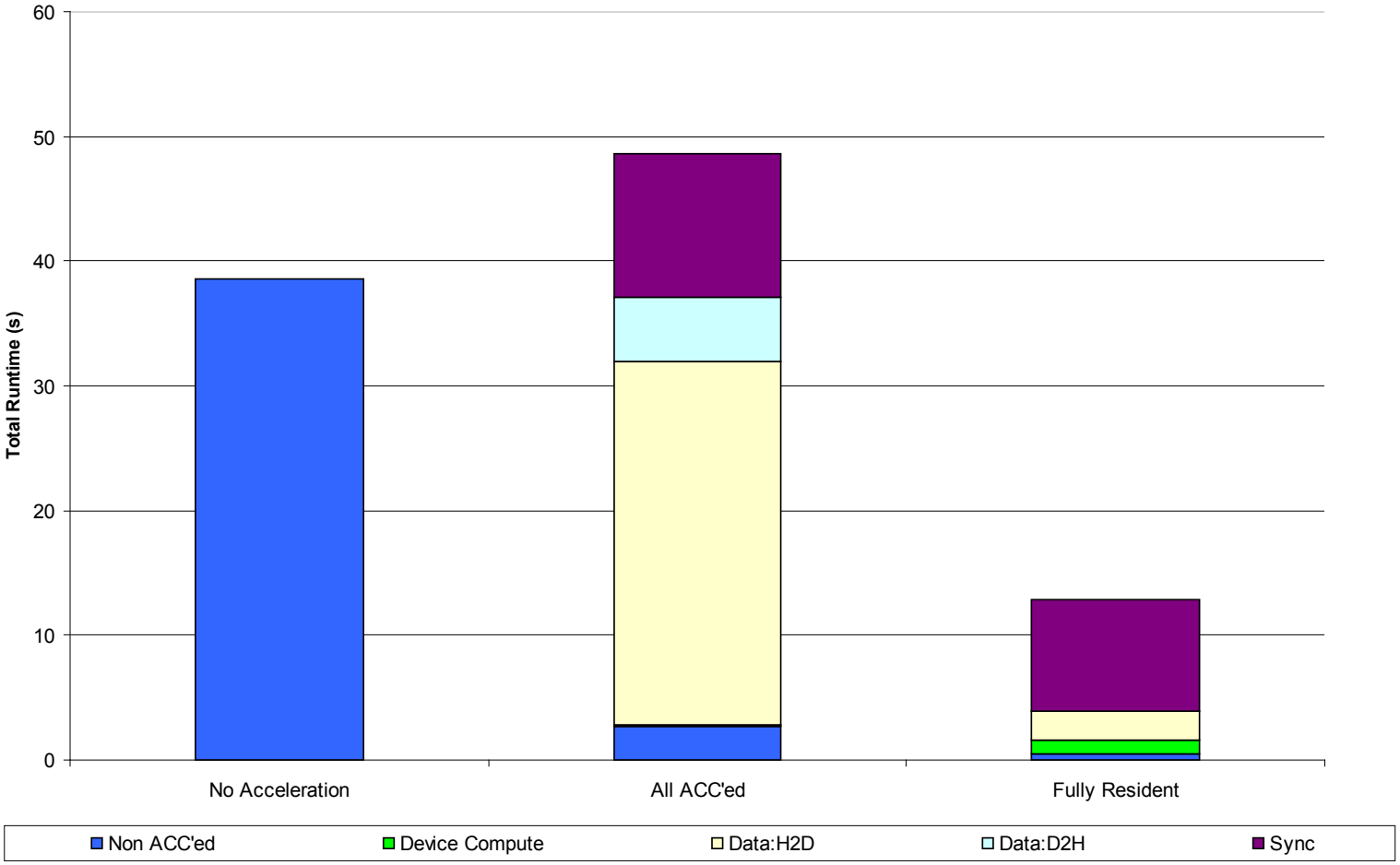
```
PROGRAM main
<stuff>
!$acc data &
!$acc copyin(r,s,t)&
!$acc copyin(u,v,w)&
!$acc copyin(x,y,z)&
!$acc copyout(a)
CALL CloverLeaf
!$acc end data
END PROGRAM main
```

```
SUBROUTINE kernel_A
<stuff>
!$acc data &
!$acc present(r,s,t)
!$acc copyout(r)
!$acc parall
   DO k = y_
        DO j
           <
        ENDD
   ENDDO
!$acc end pa
!$acc end da
<stuff>
END SUBROUTI
```

```
SUBROUTINE kernel_B
<stuff>
!$acc data &
!$acc present(u,v,w)
!$acc copyout(u)
!$acc parall
   DO k = y_
        DO j
           <
        ENDD
   ENDDO
!$acc end pa
!$acc end da
<stuff>
END SUBROUTI
```

```
SUBROUTINE kernel_C
   <stuff>
!$acc data &
!$acc present(x,y,z)
!$acc copyout(x)
!$acc parallel loop
   DO k = y_min, y_max
        DO j = x_min, x_max
           <stuff>
        ENDDO
   ENDDO
!$acc end parallel loop
!$acc end data
<stuff>
END SUBROUTINE kernel_C
```

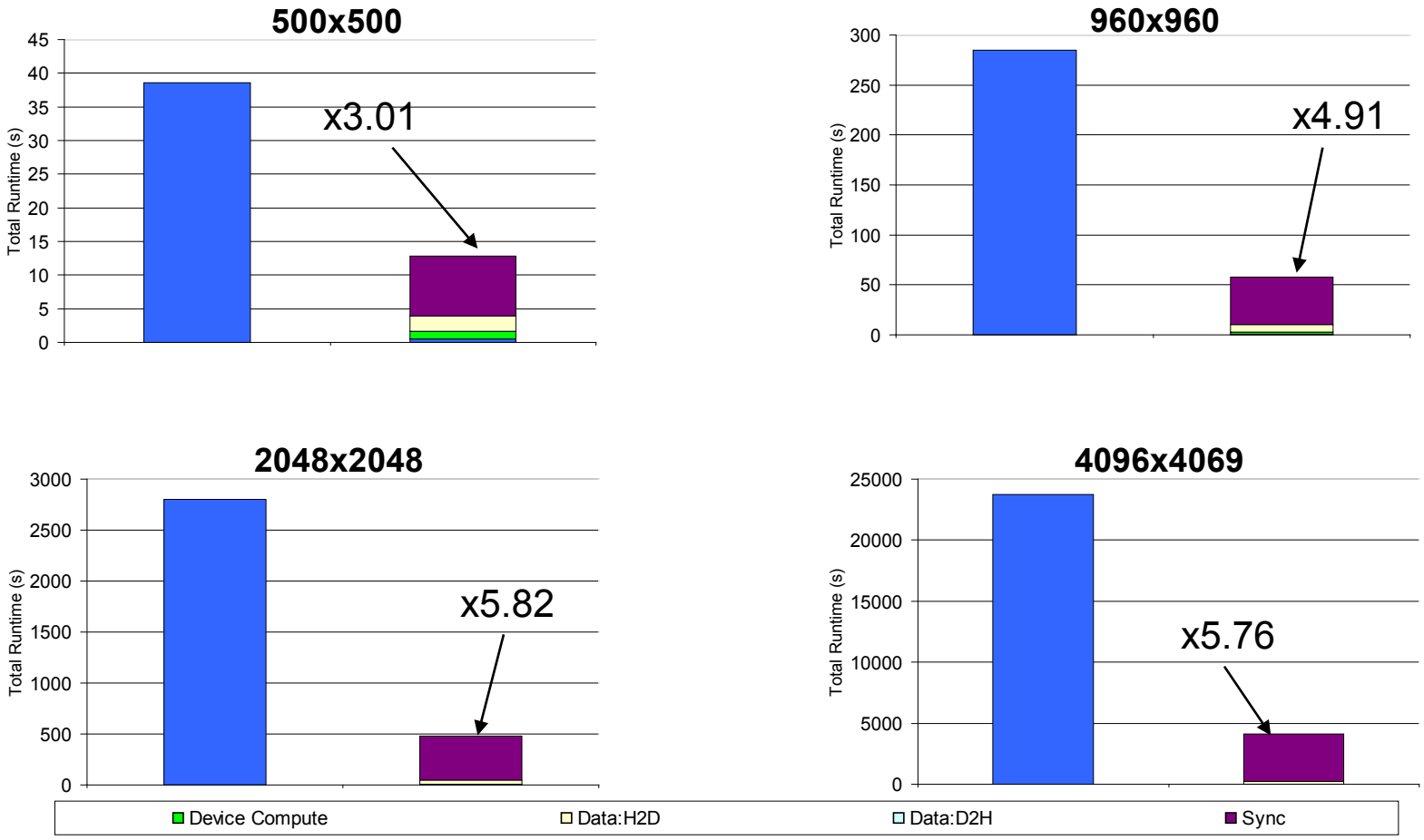
Fully Resident on Accelerator



Increasing Problem Size

- 500 x 500 modest problem size
- Typical problem sizes in range 300k to 8M cells
- Science demanding even greater
- Same performance as this is increased?

Increasing Problem Size



Hybrid MPI/OpenMP Comparison

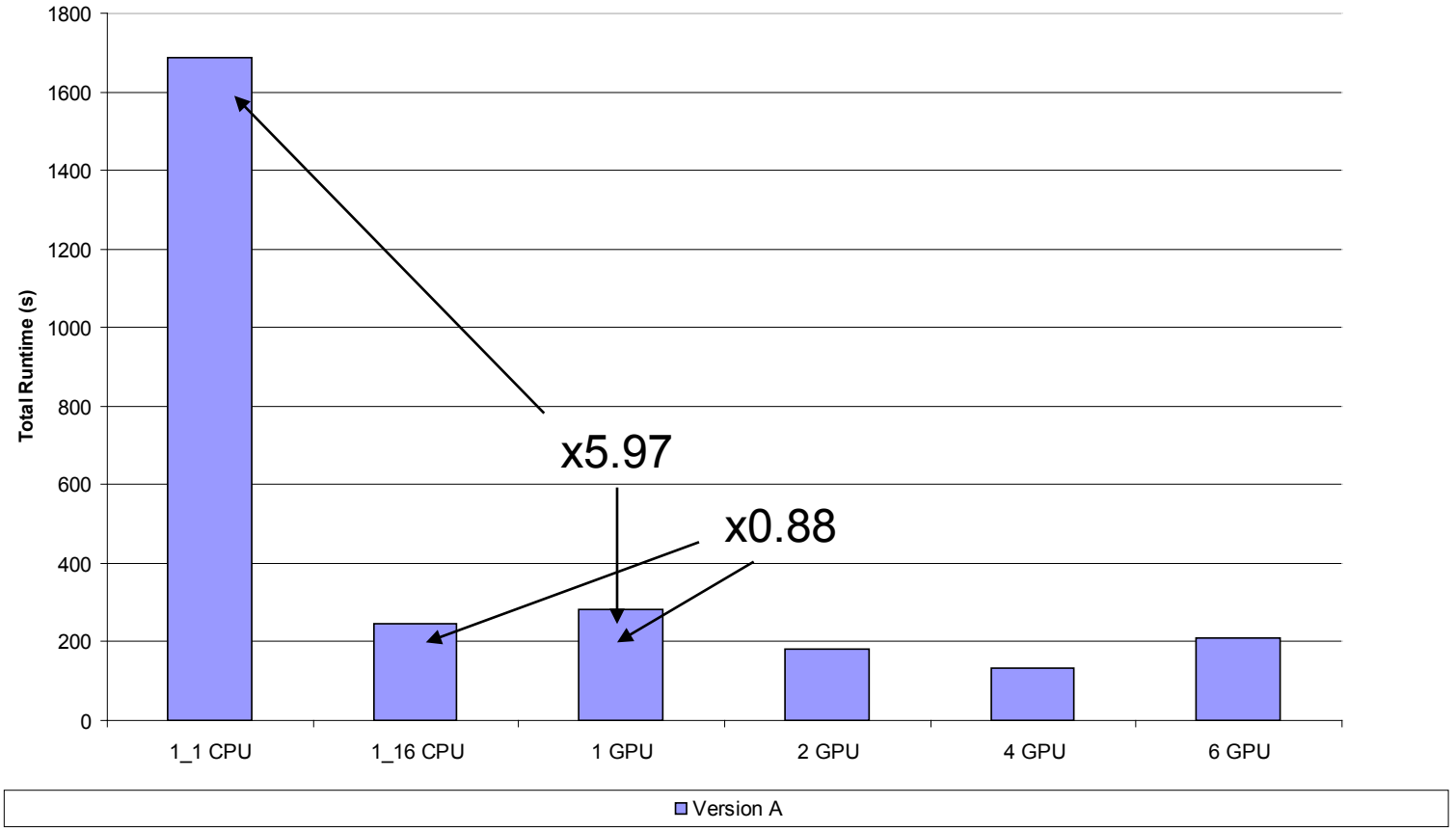
Architecture	Optimal Utilisation	Turnaround Time for 960^2
2.1 GHz Interlagos	8 MPI 1 OpenMP (1 core / “Bulldozer”)	43.52
1.16 GHz nVidia X2090	OpenACC	58.03

Hybrid MPI/OpenACC

- Same test case
- Use the 960x960 mesh
- Up simulation time from 0.5 μs to 15.5 μs
  - Need to run a little longer now going distributed
- Halo exchange data now needs to be updated on host and device
- Explicitly packing our own buffers
- This is “Version A”

```
SUBROUTINE exchange
!$acc data &
!$acc present(snd_buffer)
!$acc parallel loop
   DO k = y_min_dpth, y_max_dpth
        DO j = 1, dpth
           <pack snd buffer>
        ENDDO
   ENDDO
!$acc end parallel loop
!$acc update host(snd_buffer)
!$acc end data
CALL MPI_IRECV(rcv_buffer)
CALL MPI_ISEND(snd_buffer)
CALL MPI_WAITALL
!$acc data &
!$acc present(rcv_buffer)
!$acc update device(rcv_buffer)
!$acc parallel loop
   DO k = y_min_dpth, y_max_dpth
        DO j = 1, dpth
           <unpack rcv buffer>
        ENDDO
   ENDDO
!$acc end parallel loop
!$acc end data
END SUBROUTINE exchange
```

“Version A” Performance



What's it Actually Doing?

- Use CrayPat to re-profile for the GPU
- Tells us advec and cell momentum routines still dominate as they did for CPU profile
- Intuitively this is what you'd expect
- But what's it actually doing?
- Is it "good" or "bad"?

```
Time% |     Time |   Calls  | Function

 100.0% | 83.010479 | 415631.0 |Total
|-----------------------------------------------------------------------------------------------------
| 100.0% | 83.010473 | 415629.0 |USER
||----------------------------------------------------------------------------------------------------
||  33.6% | 27.873962 |   2000.0 | advec_cell_kernelACC_SYNC_WAIT@li.238
||  24.1% | 19.985739 |   4000.0 | advec_mom_kernel_.ACC_SYNC_WAIT@li.216
||  15.5% | 12.873780 |   1000.0 | timestep.ACC_SYNC_WAIT@li.51
||   5.2% |  4.329525 |   4000.0 | advec_mom_kernel_.ACC_COPY@li.216
||   2.9% |  2.443764 |   1000.0 | accelerate_kernel_.ACC_SYNC_WAIT@li.101
||   2.6% |  2.166314 |   2000.0 | advec_cell_kernel_.ACC_COPY@li.238
||   2.4% |  1.970592 |   4000.0 | advec_mom_kernel_.ACC_COPY@li.68
||   1.5% |  1.278792 |   1000.0 | pdv_kernel_.ACC_SYNC_WAIT@li.133
||   1.2% |  1.034328 |   2000.0 | pdv_kernel_.ACC_SYNC_WAIT@li.137
||   1.2% |  0.987142 |   2000.0 | advec_cell_kernel_.ACC_COPY@li.58
||   1.2% |  0.976646 |   4000.0 | timestep_.ACC_COPY@li.51
```

Use the “listing file”!

- Use the “-ra” CCE compiler option
- Creates listing files (*.lst)
- Shows that kernel isn’t threaded in advec_cell.f90
- work is done sequentially
  - j is split among the threads, then all threads are doing the same j at the same time
- Why? - loops are calculating values and then using these updated values i.e. “dependencies”
- Can remove them? - Yes!
- Break loop in two
- move updates (pre / post mass, energy, volume) into separate update loop

```
G - Accelerated  g - partitioned
```

```
G-----< !$acc parallel loop
G g---<  DO k = y_min, y_max
G g 3-<    DO j = x_min, x_max
G g 3-<        <stuff>
G g 3-<    ENDDO
G g---<  ENDDO
G-----< !$acc end parallel loop
```

Ftn-6405 ftn: ACCEL File=advec_cell.f90 , Line=93

A region starting at line 93 and ending at line 99 was placed on the accelerator

Ftn-6430 ftn:ACCEL File=advec_cell.f90, Line=94

A loop starting at line 94 was partitioned across the threadblocks and the 128 threads within a threadblock

Ftn-6411 ftn: ACCEL File advec_cell.f90, Line=95

A loop starting at line 95 will be serially executed

Check the “listing file”

- Check the “*.lst” file
- Inner loop is now partitioned over the threadblocks

```
G - Accelerated  g - partitioned
```

```
G-----< !$acc parallel loop
G g---<  DO k = y_min, y_max
G g g-<    DO j = x_min, x_max
G g g-<        <stuff>
G g g-<    ENDDO
G g---<  ENDDO
G-----< !$acc end parallel loop
```

Ftn-6405 ftn: ACCEL File=advec_cell.f90 , Line=93

A region starting at line 93 and ending at line 99 was placed on the accelerator

Ftn-6430 ftn:ACCEL File=advec_cell.f90, Line=94

A loop starting at line 94 was partitioned across the threadblocks

Ftn-6430 ftn:ACCEL File=advec_cell.f90, Line=95
A loop starting at line 95 was partitioned across the 128 threads within a threadblock

Re-Profile

- Re-profile with CrayPat
- advec_cell: 28s to 12s
- Still correct answer?
- Yes!
- This is “Version B”

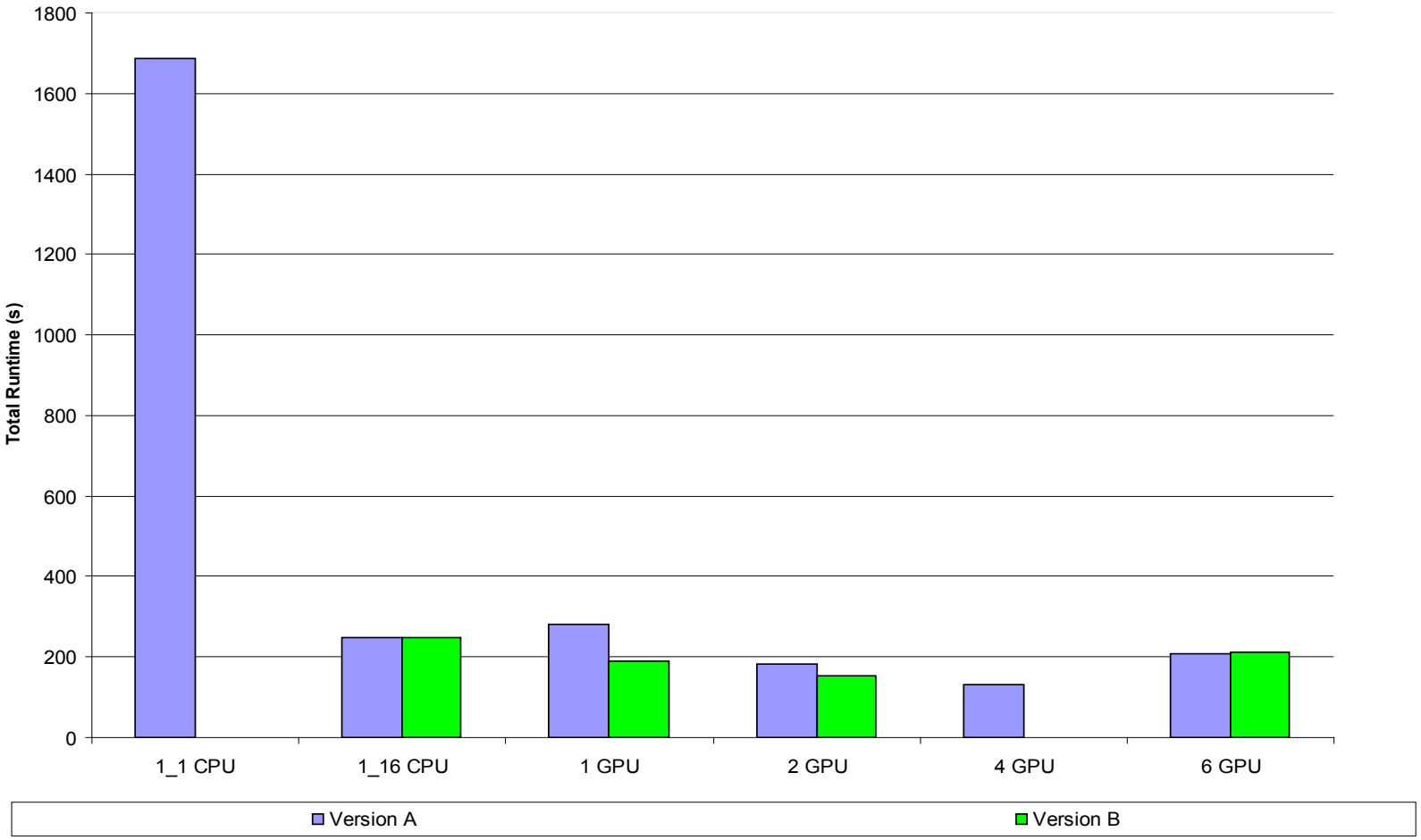
```
Time% |     Time |   Calls  | Function

 100.0% | 83.010479 | 415631.0 |Total
|-----------------------------------------------------------------------------------------------------------
| 100.0% | 83.010473 | 415629.0 |USER
||----------------------------------------------------------------------------------------------------------
||  33.6% | 27.873962 |   2000.0 | advec_cell_kernelACC_SYNC_WAIT@li.238
||  24.1% | 19.985739 |   4000.0 | advec_mom_kernel_.ACC_SYNC_WAIT@li.216
||  15.5% | 12.873780 |   1000.0 | timestep.ACC_SYNC_WAIT@li.51
||   5.2% |  4.329525 |   4000.0 | advec_mom_kernel_.ACC_COPY@li.216
||   2.9% |  2.443764 |   1000.0 | accelerate_kernel_.ACC_SYNC_WAIT@li.101
||   2.6% |  2.166314 |   2000.0 | advec_cell_kernel_.ACC_COPY@li.238
||   2.4% |  1.970592 |   4000.0 | advec_mom_kernel_.ACC_COPY@li.68
||   1.5% |  1.278792 |   1000.0 | pdv_kernel_.ACC_SYNC_WAIT@li.133
||   1.2% |  1.034328 |   2000.0 | pdv_kernel_.ACC_SYNC_WAIT@li.137
||   1.2% |  0.987142 |   2000.0 | advec_cell_kernel_.ACC_COPY@li.58
||   1.2% |  0.976646 |   4000.0 | timestep_.ACC_COPY@li.51
```

```
Time% |     Time |   Calls  | Function

100.0% | 66.375613 | 415631.0 |Total
|-----------------------------------------------------------------------------------------------------------
| 100.0% | 66.375607 | 415629.0 |USER
||----------------------------------------------------------------------------------------------------------
||  29.2% | 19.370430 |   4000.0 |advec_mom_kernel.ACC_SYNC_WAIT@li.216
||  19.3% | 12.785822 |   1000.0 |timestep.ACC_SYNC_WAIT@li.51
||  17.9% | 11.913056 |   2000.0 |advec_cell_kernel_.ACC_SYNC_WAIT@li.240
||   6.5% |  4.327830 |   4000.0 |advec_mom_kernel_.ACC_COPY@li.216
||   3.7% |  2.444010 |   1000.0 |accelerate_kernel.ACC_SYNC_WAIT@li.101
||   3.3% |  2.165092 |   2000.0 |advec_cell_kernel.ACC_COPY@li.240
||   3.0% |  1.970679 |   4000.0 |advec_mom_kernel_.ACC_COPY@li.68
||   1.9% |  1.278686 |   1000.0 |pdv_kernel_.ACC_SYNC_WAIT@li.133
||   1.6% |  1.033906 |   2000.0 |pdv_kernel_.ACC_SYNC_WAIT@li.137
||   1.5% |  1.019408 |   4000.0 |timestep_.ACC_COPY@li.51
||   1.5% |  0.986405 |   2000.0 |advec_cell_kernel_.ACC_COPY@li.58
```

“Version B” Performance



Follow Same Procedure

- Now advec_mom dominating profile

```
Time% |     Time |   Calls  | Function
100.0% | 66.375613 | 415631.0 |Total
|-----------------------------------------------------------------------------------------------------------
| 100.0% | 66.375607 | 415629.0 |USER
||----------------------------------------------------------------------------------------------------------
||  29.2% | 19.370430 |   4000.0 |advec_mom_kernel.ACC_SYNC_WAIT@li.216
||  19.3% | 12.785822 |   1000.0 |timestep.ACC_SYNC_WAIT@li.51
||  17.9% | 11.913056 |   2000.0 |advec_cell_kernel_.ACC_SYNC_WAIT@li.240
||   6.5% |  4.327830 |   4000.0 |advec_mom_kernel_.ACC_COPY@li.216
||   3.7% |  2.444010 |   1000.0 |accelerate_kernel.ACC_SYNC_WAIT@li.101
||   3.3% |  2.165092 |   2000.0 |advec_cell_kernel.ACC_COPY@li.240
||   3.0% |  1.970679 |   4000.0 |advec_mom_kernel_.ACC_COPY@li.68
||   1.9% |  1.278686 |   1000.0 |pdv_kernel_.ACC_SYNC_WAIT@li.133
||   1.6% |  1.033906 |   2000.0 |pdv_kernel_.ACC_SYNC_WAIT@li.137
||   1.5% |  1.019408 |   4000.0 |timestep_.ACC_COPY@li.51
||   1.5% |  0.986405 |   2000.0 |advec_cell_kernel_.ACC_COPY@li.58
```

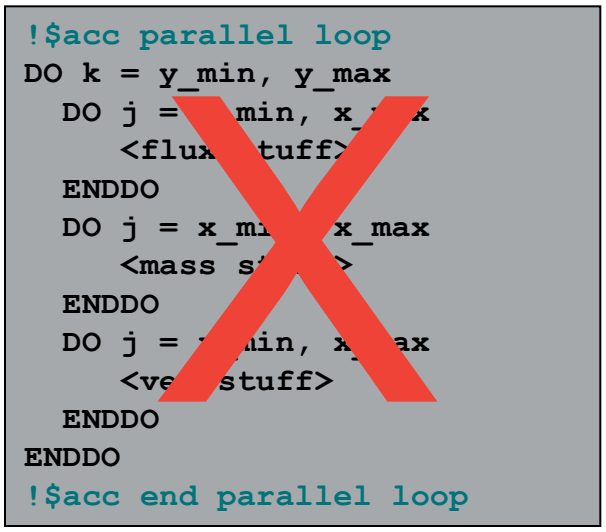
```
!$acc parallel loop
DO k = y_min, y_max
  DO j = x_min, x_max
     <flux stuff>
  ENDDO
  DO j = x_min, x_max
     <mass stuff>
  ENDDO
  DO j = x_min, x_max
     <vel stuff>
  ENDDO
ENDDO
!$acc end parallel loop
```

- "*.lst" file shows that loops with multiple levels of nesting are not being accelerated

Remove Nested Loops

- Multiple levels of loop nesting removed.

```
!$acc parallel loop
DO k = y_min, y_max
  DO j = x_min, x_max
     <flux stuff>
  ENDDO
  DO j = x_min, x_max
     <mass stuff>
  ENDDO
  DO j = x_min, x_max
     <vel stuff>
  ENDDO
ENDDO
!$acc end parallel loop
```



```
!$acc parallel loop
DO k = y_min, y_max
  DO j = x_min, x_max
     <flux stuff>
  ENDDO
ENDDO
!$acc end parallel loop
!$acc parallel loop
DO k = y_min, y_max
  DO j = x_min, x_max
     <mass stuff>
  ENDDO
ENDDO
!$acc end parallel loop
!$acc parallel loop
DO k = y_min, y_max
  DO j = x_min, x_max
     <vel stuff>
  ENDDO
ENDDO
!$acc end parallel loop
```

Not Quite Partitioned

- Multiple levels of loop nesting removed.
- Now all but one is partitioned across the threads.
- Know it's ok, so force it to be scheduled across all threads
  - !$ACC LOOP VECTOR

```
!$acc parallel loop
DO k = y_min, y_max
  DO j = x_min, x_max
     <flux stuff>
  ENDDO
ENDDO
!$acc end parallel loop
!$acc parallel loop
DO k = y_min, y_max
  DO j = x_min, x_max
      <mass stuff>
  ENDDO
ENDDO
!$acc end parallel loop
!$acc parallel loop
DO k = y_min, y_max
!$acc loop vector
  DO j = x_min, x_max
     <vel stuff>
  ENDDO
ENDDO
!$acc end parallel loop
```

Check “listing file” and Re-Profile

- “*.lst” file shows all loops now partitioned across the threads
- Re-profile with CrayPat
- advec_mom: 19s to 8s
- Still correct answer?
- Yes!

```
Time%  |     Time  |   Calls  | Function
100.0% | 66.375613 | 415631.0 |Total
|-----------------------------------------------------------------------------------------------------
| 100.0% | 66.375607 | 415629.0 |USER
||----------------------------------------------------------------------------------------------------
||  29.2% | 19.370430 |   4000.0 |advec_mom_kernel.ACC_SYNC_WAIT@li.216
||  19.3% | 12.785822 |   1000.0 |timestep.ACC_SYNC_WAIT@li.51
||  17.9% | 11.913056 |   2000.0 |advec_cell_kernel_.ACC_SYNC_WAIT@li.240
||   6.5% |  4.327830 |   4000.0 |advec_mom_kernel_.ACC_COPY@li.216
||   3.7% |  2.444010 |   1000.0 |accelerate_kernel.ACC_SYNC_WAIT@li.101
||   3.3% |  2.165092 |   2000.0 |advec_cell_kernel.ACC_COPY@li.240
||   3.0% |  1.970679 |   4000.0 |advec_mom_kernel_.ACC_COPY@li.68
||   1.9% |  1.278686 |   1000.0 |pdv_kernel_.ACC_SYNC_WAIT@li.133
||   1.6% |  1.033906 |   2000.0 |pdv_kernel_.ACC_SYNC_WAIT@li.137
||   1.5% |  1.019408 |   4000.0 |timestep_.ACC_COPY@li.51
||   1.5% |  0.986405 |   2000.0 |advec_cell_kernel_.ACC_COPY@li.58
```

```
Time%  |     Time  |   Calls  | Function
 100.0% | 48.804249 | 447631.0 |Total
|-----------------------------------------------------------------------------------------------------
| 100.0% | 48.804243 | 447629.0 |USER
||----------------------------------------------------------------------------------------------------
||  26.4% | 12.874175 |  1000.0 |timestep.ACC_SYNC_WAIT@li.51
||  16.6% |  8.094541 |  4000.0 |advec_mom_kernel.ACC_SYNC_WAIT@li.247
||   9.8% |  4.794964 |  2000.0 |advec_cell_kernel_.ACC_SYNC_WAIT@li.236
||   8.9% |  4.328033 |  4000.0 |advec_mom_kernel_.ACC_COPY@li.247
||   5.0% |  2.442117 |  1000.0 |accelerate_kernel.ACC_SYNC_WAIT@li.101
||   4.4% |  2.164464 |  2000.0 |advec_cell_kernel_.ACC_COPY@li.236
||   4.0% |  1.969379 |  4000.0 |advec_mom_kernel$_.ACC_COPY@li.68
||   2.6% |  1.278651 |  1000.0 |pdv_kernel$_.ACC_SYNC_WAIT@li.133
||   2.1% |  1.034398 |  2000.0 |pdv_kernel$_.ACC_SYNC_WAIT@li.137
||   2.0% |  0.985912 |  2000.0 |advec_cell_kernel$_.ACC_COPY@li.58
||   2.0% |  0.972801    4000.0 |timestep$_.ACC_COPY@li.51
```

Avoid Global Variable Access

- Now timestep routine is dominating
- Again the “*.lst” file shows timestep was just running on one thread
- Issue was global variables
- threads could potentially write to these, hence scheduled on one thread
- These are only used in summary print, so currently disabled
- If/once MINLOC supported then can get round this

```
Time% |     Time |   Calls  | Function
 100.0% | 48.804249 | 447631.0 |Total
|-----------------------------------------------------------------------------------------------------------
| 100.0% | 48.804243 | 447629.0 |USER
||----------------------------------------------------------------------------------------------------------
||  26.4% | 12.874175 |  1000.0 |timestep.ACC_SYNC_WAIT@li.51
||  16.6% |  8.094541 |  4000.0 |advec_mom_kernel.ACC_SYNC_WAIT@li.247
||   9.8% |  4.794964 |  2000.0 |advec_cell_kernel_.ACC_SYNC_WAIT@li.236
||   8.9% |  4.328033 |  4000.0 |advec_mom_kernel_.ACC_COPY@li.247
||   5.0% |  2.442117 |  1000.0 |accelerate_kernel.ACC_SYNC_WAIT@li.101
||   4.4% |  2.164464 |  2000.0 |advec_cell_kernel_.ACC_COPY@li.236
||   4.0% |  1.969379 |  4000.0 |advec_mom_kernel$_.ACC_COPY@li.68
||   2.6% |  1.278651 |  1000.0 |pdv_kernel$_.ACC_SYNC_WAIT@li.133
||   2.1% |  1.034398 |  2000.0 |pdv_kernel$_.ACC_SYNC_WAIT@li.137
||   2.0% |  0.985912 |  2000.0 |advec_cell_kernel$_.ACC_COPY@li.58
||   2.0% |  0.972801    4000.0 |timestep$_.ACC_COPY@li.51
```

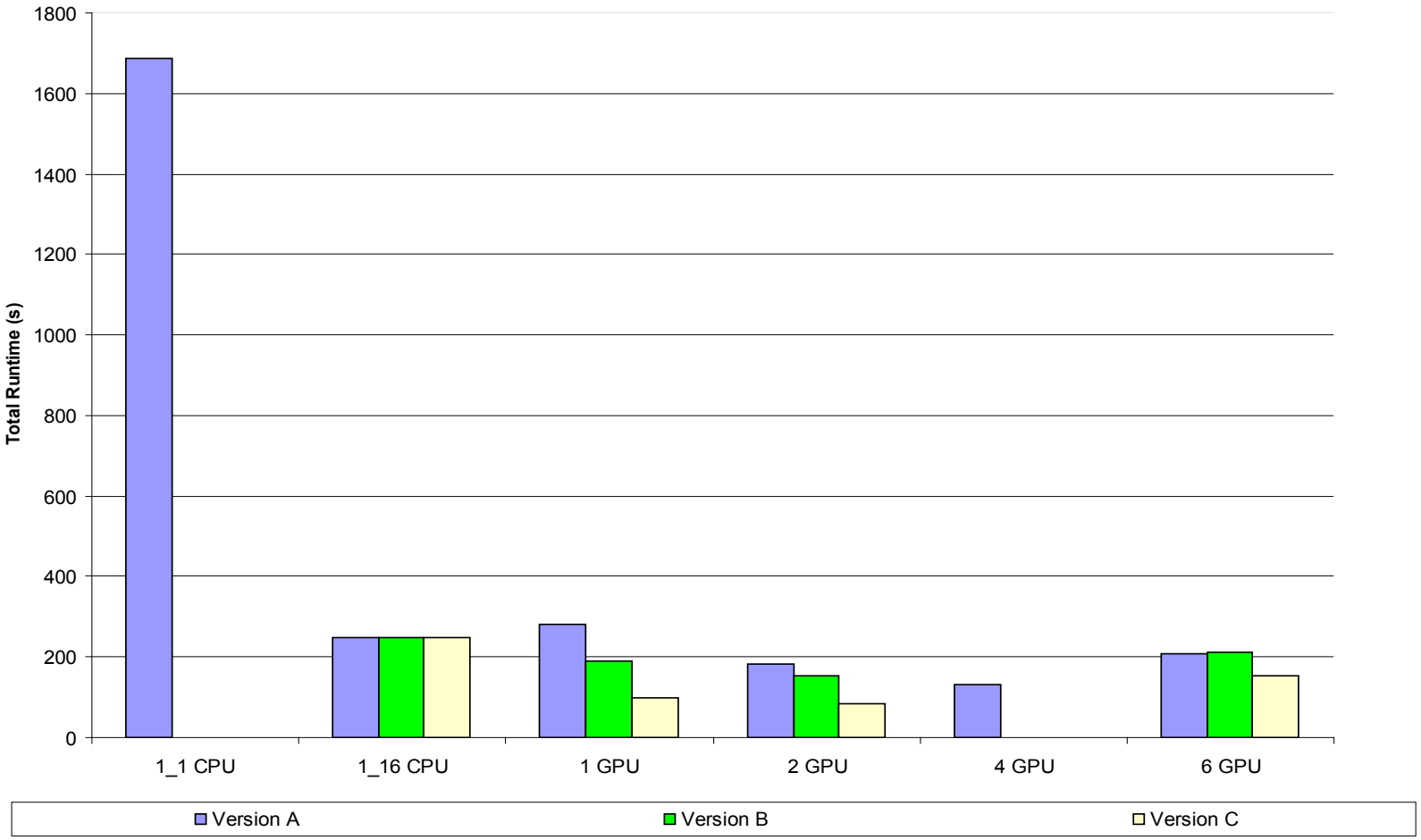
Check “listing file” and Re-Profile - Again

- “*.lst” file shows all loops now partitioned across the threads
- Re-profile with CrayPat
- timestep: 13s to < 1s
- Still correct answer?
- Yes!
- This is “Version C”

```
Time% |     Time |   Calls  | Function
 100.0% | 48.804249 | 447631.0 |Total
|-----------------------------------------------------------------------------------------------------------
| 100.0% | 48.804243 | 447629.0 |USER
||----------------------------------------------------------------------------------------------------------
||  26.4% | 12.874175 |  1000.0 |timestep.ACC_SYNC_WAIT@li.51
||  16.6% |  8.094541 |  4000.0 |advec_mom_kernel.ACC_SYNC_WAIT@li.247
||   9.8% |  4.794964 |  2000.0 |advec_cell_kernel_.ACC_SYNC_WAIT@li.236
||   8.9% |  4.328033 |  4000.0 |advec_mom_kernel_.ACC_COPY@li.247
||   5.0% |  2.442117 |  1000.0 |accelerate_kernel.ACC_SYNC_WAIT@li.101
||   4.4% |  2.164464 |  2000.0 |advec_cell_kernel_.ACC_COPY@li.236
||   4.0% |  1.969379 |  4000.0 |advec_mom_kernel$_.ACC_COPY@li.68
||   2.6% |  1.278651 |  1000.0 |pdv_kernel$_.ACC_SYNC_WAIT@li.133
||   2.1% |  1.034398 |  2000.0 |pdv_kernel$_.ACC_SYNC_WAIT@li.137
||   2.0% |  0.985912 |  2000.0 |advec_cell_kernel$_.ACC_COPY@li.58
||   2.0% |  0.972801    4000.0 |timestep$_.ACC_COPY@li.51
```

```
 Time% |     Time |  Calls | Function
 100.0% | 37.151994 | 447631.0 |Total
|-----------------------------------------------------------------------------------------------------------
| 100.0% | 37.151988 |  447629.0 |USER
||----------------------------------------------------------------------------------------------------------
||  21.8% |  8.103629 |   4000.0 |advec_mom_kernel_.ACC_SYNC_WAIT@li.247
||  12.9% |  4.797456 |   2000.0 |advec_cell_kernel_.ACC_SYNC_WAIT@li.236
||  11.6% |  4.322071 |   4000.0 |advec_mom_kernel_.ACC_COPY@li.247
||   6.6% |  2.447886 |   1000.0 |accelerate_kernel_.ACC_SYNC_WAIT@li.101
||   5.8% |  2.160784 |   2000.0 |advec_cell_kernel_.ACC_COPY@li.236
||   5.3% |  1.965759 |   4000.0 |advec_mom_kernel_.ACC_COPY@li.68
||   3.4% |  1.278799 |   1000.0 |pdv_kernel_.ACC_SYNC_WAIT@li.133
||   3.4% |  1.261867 |   1000.0 |timestep_.ACC_SYNC_WAIT@li.51
||   2.8% |  1.034047 |   2000.0 |pdv_kernel_.ACC_SYNC_WAIT@li.137
||   2.7% |  0.985090 |   2000.0 |advec_cell_kernel_.ACC_COPY@li.58
||   2.6% |  0.976743 |   4000.0 |timestep$timestep_module_.ACC_COPY@li.51
||   1.0% |  0.381213 |   1001.0 |update_halo_kernel_.ACC_KERNEL@li.334
```

“Version C” Performance



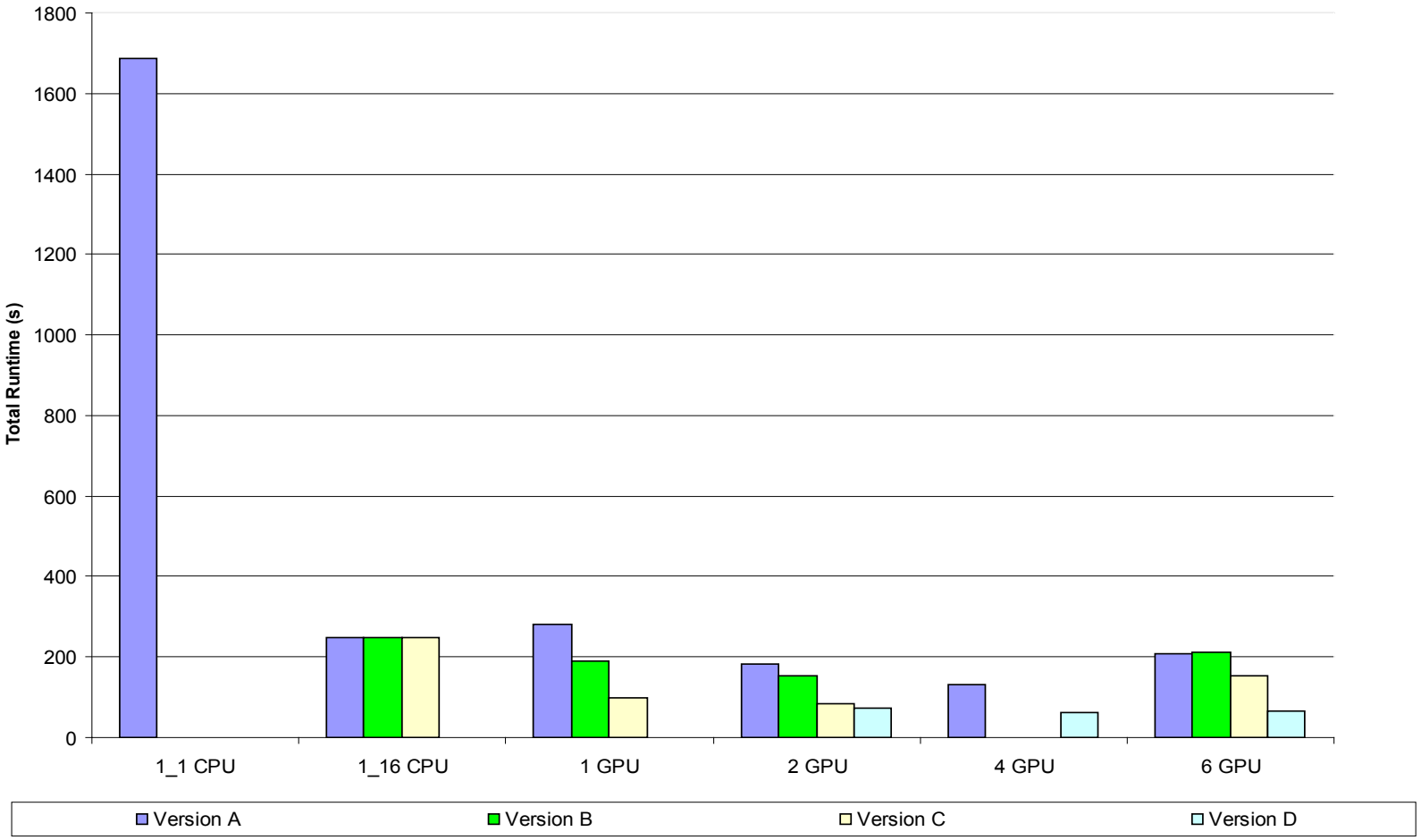
Going multi GPU

- Looked next at bottlenecks affecting multi GPU execution
- Limit halo depth copy appropriately
  - Previously copying max. data possible, rather than what is necessary: i.e. 1:size
- Used CRAY_ACC_DEBUG
  - Writes accelerator-related activity to stdout
  - Levels range 0 to 3 (Using 2)
  - Available on all versions of CCE
  - Only documented (“man crayftn”) on 8.1.0.165 or greater

```
ACC: Start transfer 1 items from accelerate.f90:26
ACC:   allocate, copy to acc 'chunks' (2376 bytes)
ACC: End transfer (to acc 2376 bytes, to host 0 bytes
```

- Showed copying of “chunk” derived type
- Why?
- Implicit copy of derived type happening for scalar components of that type
- Local scalars added and copied fields for “chunk” derived type to them.
  - Stopped the implicit copy of all of the chunk derived type being copied to the GPU.
- This is “Version D”

“Version D” Performance



What are all those “ACC_SYNC_WAITS”?

- “sync waits” still dominating profile
- These are only in kernels that allocate data on the device

```
SUBROUTINE advec_mom(a,b)
REAL ALLOCATABLE(:,:) :: node_flux

ALLOCATE(node_flux(xmin:ymax))

!$acc data present_or_create(node_flux)
!$acc parallel loop
DO k = y_min, y_max
  DO j = x_min, x_max
      <flux stuff>
  ENDDO
ENDDO
!$acc end parallel loop
!$acc end data
DEALLOCATE(node_flux)
END SUBROUTINE
```

```
 Time% |     Time  | Calls | Function
 100.0% | 37.151994 | 447631.0 |Total
|-----------------------------------------------------------------------------------------------------
| 100.0% | 37.151988 |  447629.0 |USER
||----------------------------------------------------------------------------------------------------
||  21.8% |  8.103629 |    4000.0 |advec_mom_kernel_.ACC_SYNC_WAIT@li.247
||  12.9% |  4.797456 |    2000.0 |advec_cell_kernel_.ACC_SYNC_WAIT@li.236
||  11.6% |  4.322071 |    4000.0 |advec_mom_kernel_.ACC_COPY@li.247
||   6.6% |  2.447886 |    1000.0 |accelerate_kernel_.ACC_SYNC_WAIT@li.101
||   5.8% |  2.160784 |    2000.0 |advec_cell_kernel_.ACC_COPY@li.236
||   5.3% |  1.965759 |    4000.0 |advec_mom_kernel_.ACC_COPY@li.68
||   3.4% |  1.278799 |    1000.0 |pdv_kernel_.ACC_SYNC_WAIT@li.133
||   3.4% |  1.261867 |    1000.0 |timestep_.ACC_SYNC_WAIT@li.51
||   2.8% |  1.034047 |    2000.0 |pdv_kernel_.ACC_SYNC_WAIT@li.137
||   2.7% |  0.985090 |    2000.0 |advec_cell_kernel_.ACC_COPY@li.58
||   2.6% |  0.976743 |    4000.0 |timestep_.ACC_COPY@li.51
```

Pre-Allocated Temp Array “Trick”

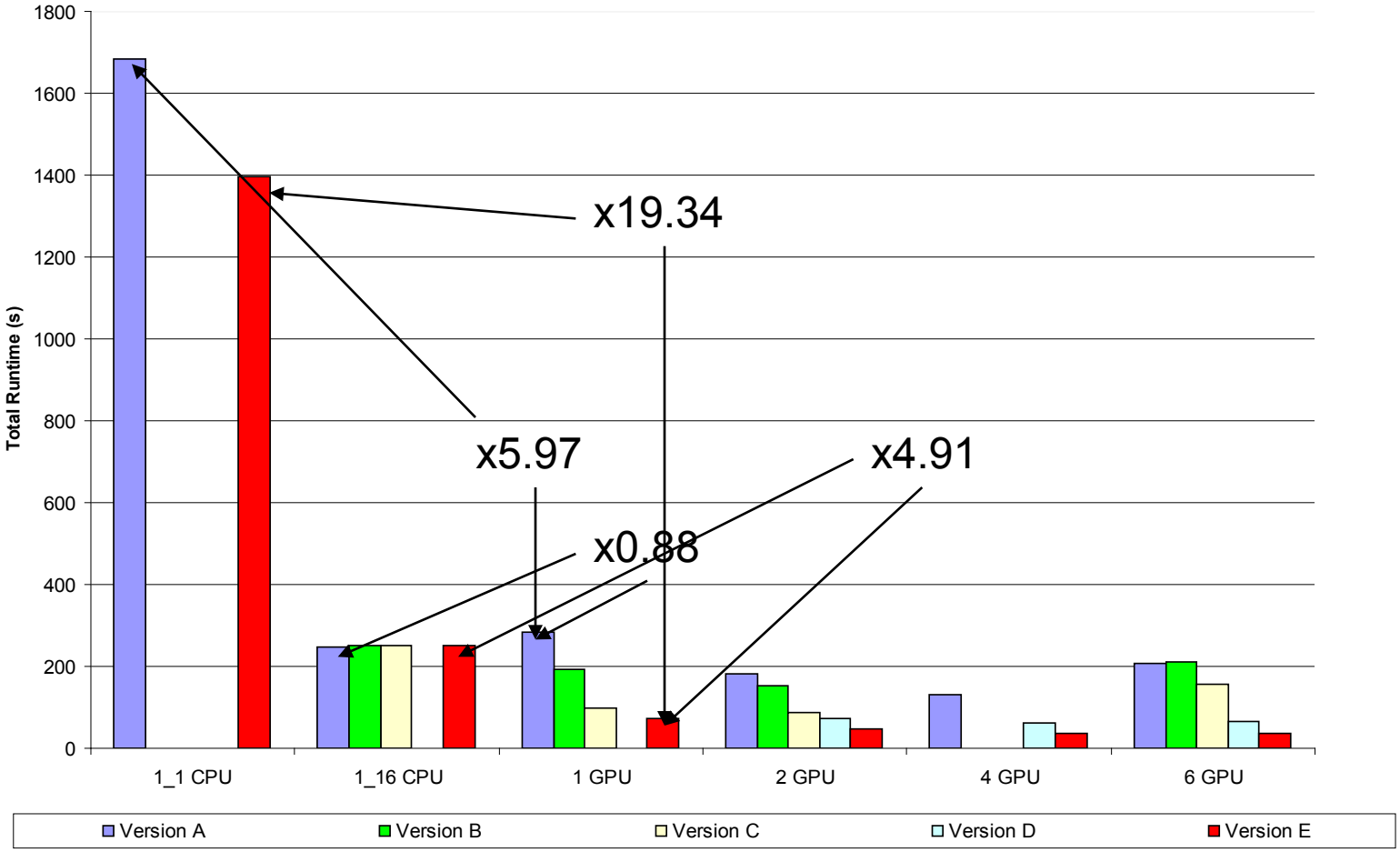
```
Time%  |     Time  |   Calls  | Function
 100.0% | 26.698295 |   437621.0 |Total
|-----------------------------------------------------------------------------------------------------------
| 100.0% | 26.698290 |   437619.0 |USER
||----------------------------------------------------------------------------------------------------------
||  17.8% |  4.749553 |    1000.0 |pdv_kernel_.ACC_SYNC_WAIT@li.132
||   8.6% |  2.297537 |    2001.0 |update_halo_kernel_.ACC_KERNEL@li.451
||   4.7% |  1.258818 |    1000.0 |timestep_.ACC_SYNC_WAIT@li.51
||   4.6% |  1.234257 |    2001.0 |update_halo_kernel_.ACC_KERNEL@li.442
||   3.6% |  0.970250 |    4000.0 |timestep_.ACC_COPY@li.51
```

```
SUBROUTINE advec_mom(a,node_flux,b)
REAL DIMENSION(xmin,ymax) :: node_flux

!$acc data present(node_flux)
!$acc parallel loop
DO k = y_min, y_max
  DO j = x_min, x_max
      <flux stuff>
  ENDDO
ENDDO
!$acc end parallel loop
!$acc end data
END SUBROUTINE
```

- Pre-allocate temporary arrays and initially copy these to device
- Re-use these multiple times by passing these through the subroutine
- Removes allocations from kernels
- “ACC_SYNC_WAIT”s gone
- No need for “present_or_create”, just “present”
- So “ACC_COPY”s gone
- This is “Version E”

“Version E” Performance



GPU Optimisations Improve the CPU Code

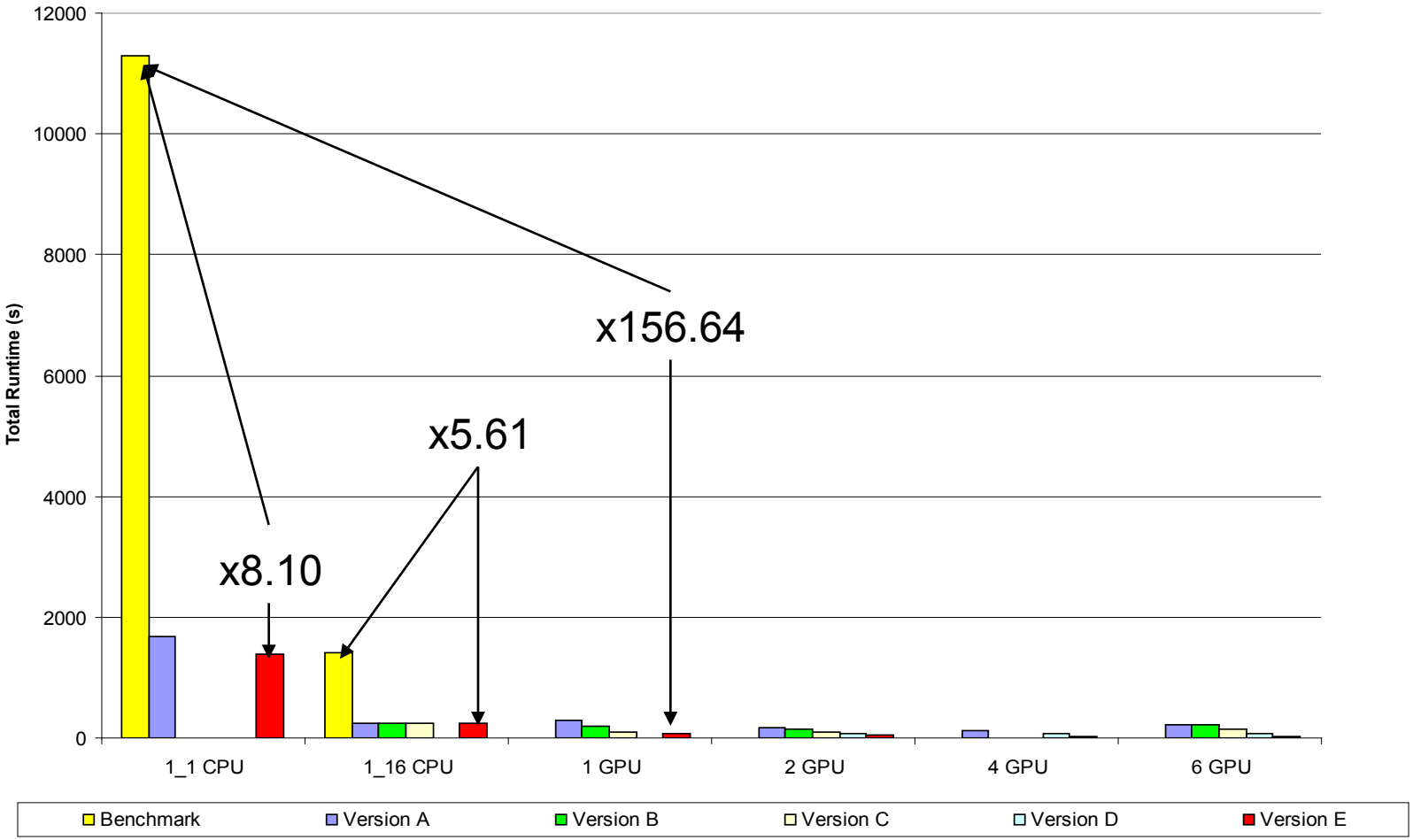
```
========================================================================
Total
------------------------------------------------------------------------
Time%                                          100.0%
Time                                        52.054641 secs
Imb. Time                                          -- secs
Imb. Time%                                         --
Calcs                          107.656 /sec         5604.0 calls
PAPI_L1_DCM                     72.452M/sec     3771487818 misses
PAPI_TLB_DM                     63.664 /sec           3314 misses
PAPI_L1_DCA                   1043.884M/sec    54339011642 refs
PAPI_FP_OPS                    652.655M/sec    33973718491 ops
Average Time per Call                           0.009289 secs
CrayPat Overhead : Time          0.0%
User time (approx)              52.055 secs    109315113219 cycles 100.0% Time
HW FP Ops / User time          652.655M/sec     33973718491 ops   3.9%peak(DP)
HW FP Ops / WCT                652.655M/sec
Computational intensity          0.31 ops/cycle       0.63 ops/ref
MFLOPS (aggregate)             652.65M/sec
TLB utilization           16396804.96 refs/miss      32025 avg uses
D1 cache hit,miss ratios         93.1% hits            6.9% misses
D1 cache utilization (misses)    14.41 refs/miss       1.801 avg hit
```

- Derived metrics “Version A”
  - PAT_RT_HWPC=1
- Looks ok
  - 652 MFLOPS

- Derived metrics “Version E”
  - L1 cache utilisation up factor 2.3
  - Average 4.164 uses per operand
  - Over 1 GFLOPS

```
========================================================================
Total
------------------------------------------------------------------------
Time%                                          100.0%
Time                                        31.776925 secs
Imb. Time                                          -- secs
Imb. Time%                                         --
Calcs                          220.725 /sec         7014.0 calls
PAPI_L1_DCM                     32.501M/sec     1032780862 misses
PAPI_TLB_DM                     73.386 /sec           2332 misses
PAPI_L1_DCA                   1082.638M/sec    34403038765 refs
PAPI_FP_OPS                   1032.789M/sec    32819004724 ops
Average Time per Call                           0.004530 secs
CrayPat Overhead : Time          0.1%
User time (approx)              31.777 secs     66732042821 cycles 100.0% Time
HW FP Ops / User time         1032.789M/sec     32819004724 ops   6.1%peak(DP)
HW FP Ops / WCT               1032.789M/sec
Computational intensity          0.49 ops/cycle       0.95 ops/ref
MFLOPS (aggregate)            1032.79M/sec
TLB utilization           14752589.52 refs/miss      28814 avg uses
D1 cache hit,miss ratios         97.0% hits            3.0% misses
D1 cache utilization (misses)    33.31 refs/miss       4.164 avg hits
```

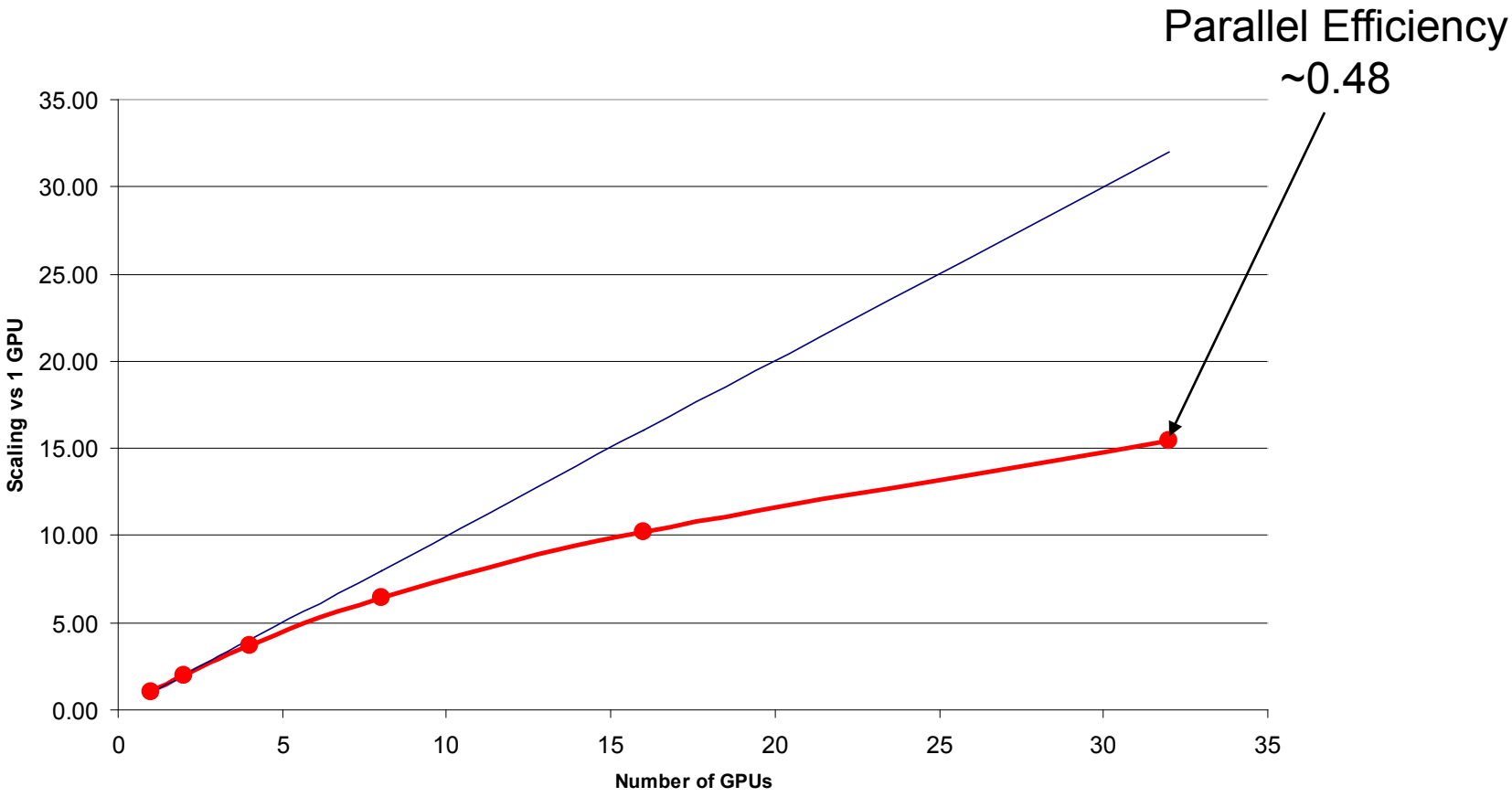
Performance Relative to Benchmark Code



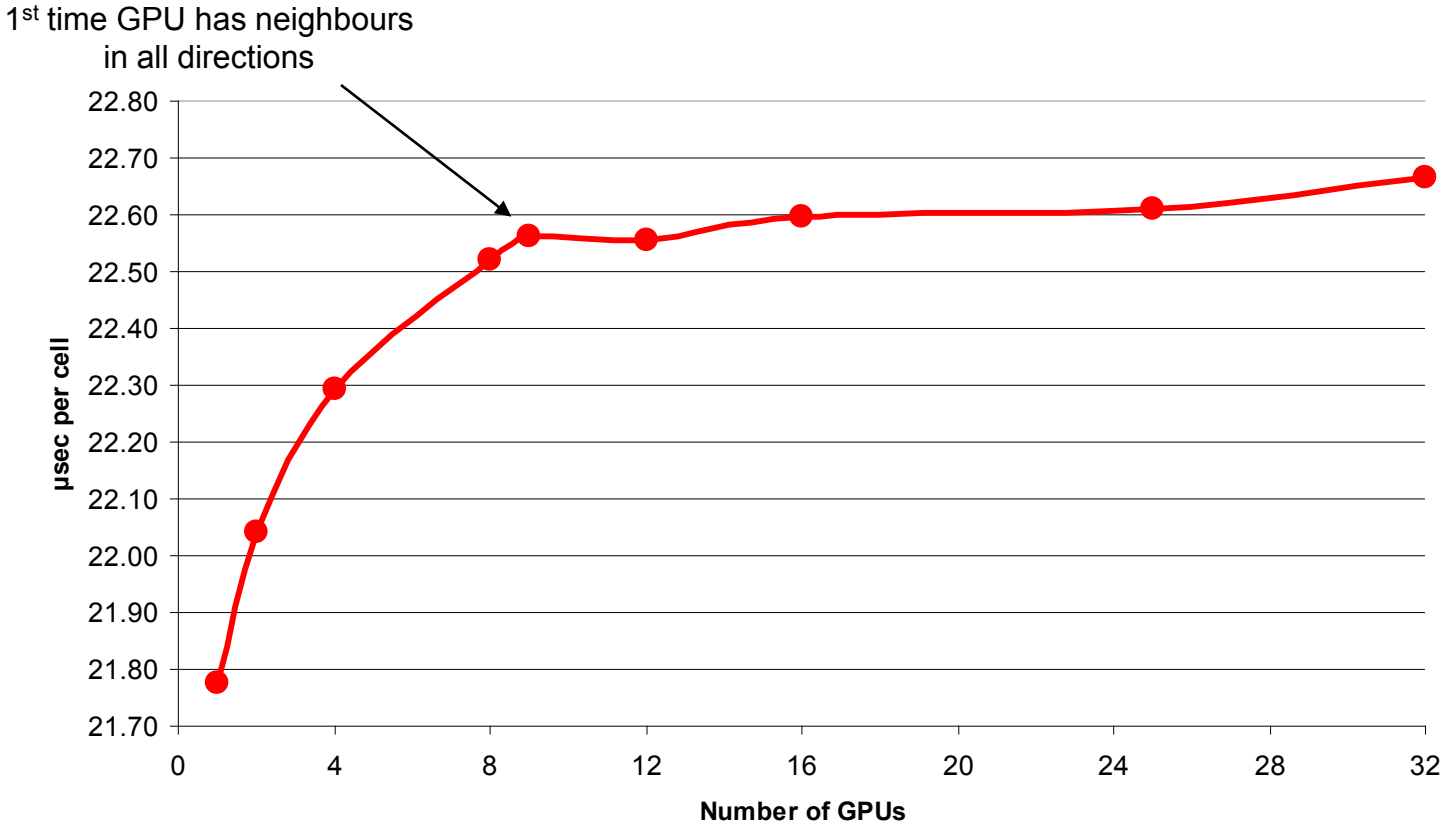
Multiple GPU Scalability

- Ramp up problem size again
- Fill the IL socket with the 960^2 => 3840^2
- Strong and Weak Scale 0.5μs problem

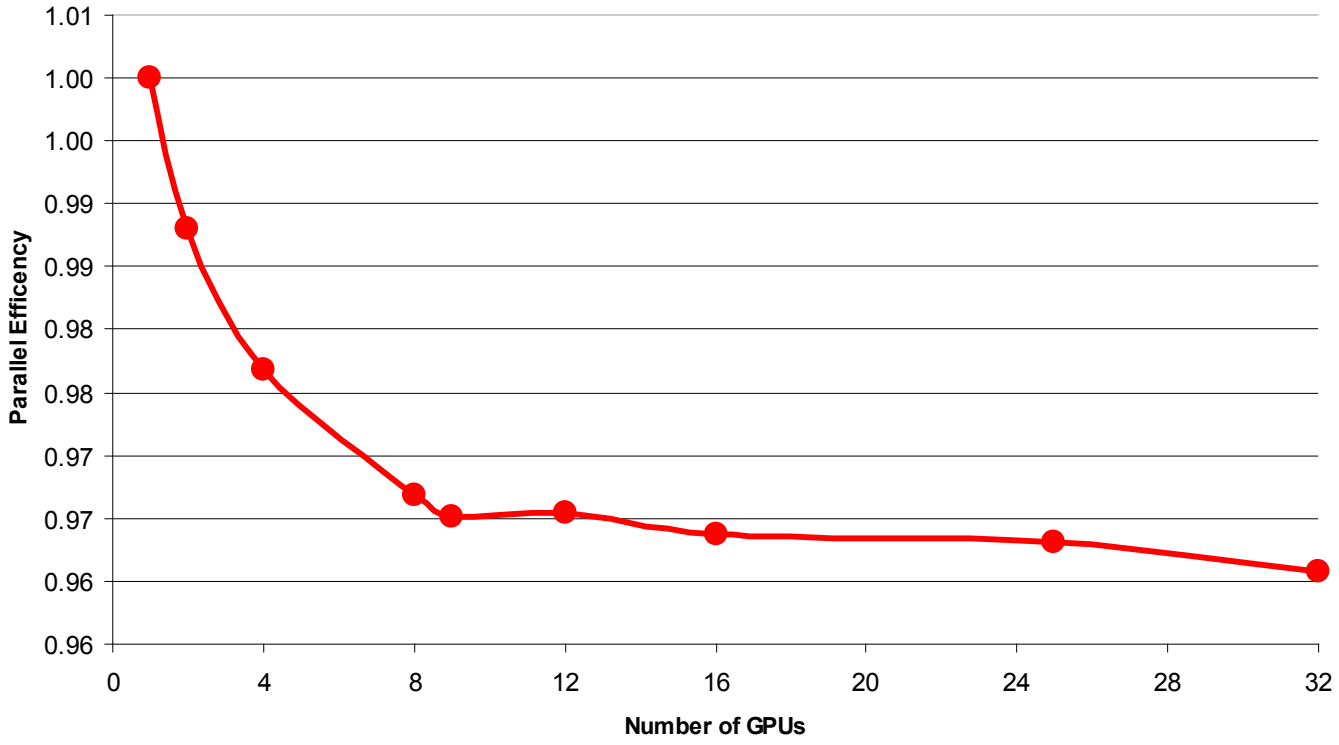
0.5μs, 3840^2 Mesh, Strong Scaled



0.5µs, 3840^2 Mesh, Weak Scaled



0.5μs, 3840^2 Mesh, Weak Scaled



Conclusions

- Quickly get accelerated with OpenACC
- Insights / optimisations that enable the compiler to generate partitioned threaded code for the accelerator via OpenACC, also enable compiler to improve generated code for CPU performance
- Compiler / Tool feedback is very good
  - Accelerator info in “*.lst” files invaluable
  - CRAY_ACC_DEBUG
  - CrayPat
- Some stuff missing
  - MINLOC
  - ATOMIC / CRITICAL operations
- Cray support has been 1st class

Summary

- Can develop OpenACC version of your code in a piecemeal manner
- Relative short timeframe have distributed accelerated code
- Explicit hydrodynamics is amenable to accelerated technology
- x19.34 over 1 Interlagos core
- x4.91 over 1 Interlagos socket
- Weak scaling shows relative constant cost per cell

What's Next?

- Feedback "mini-app" experience to production applications
- Ran on 32 GPUs (most we can) want more
- Mantevo
  - Project Leads: Richard Barrett & Mike Heroux
  - https://software.sandia.gov/mantevo
- OpenMP tasked based – MPI/OpenMP/OpenACC hybrid
- Other algorithms
  - Implicit solution to diffusion equation
  - Deterministic solution to transport equation
  - Monte Carlo solution of the transport equation

Acknowledgements

- Cray in general
  - Cray Partner Network
  - Compiler Team
  - Tools Team
- In particular Alistair Hart (Cray)