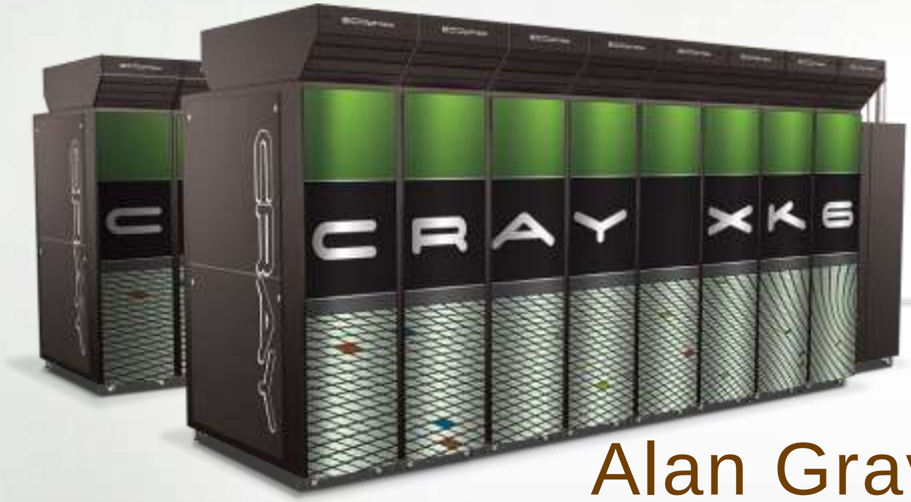
# Application acceleration and optimisation with directives on hybrid supercomputers



Alistair Hart[†],

Roberto Ansaloni,

Alan Gray, Kevin Stratford (EPCC)

([†]Cray Exascale Research Initiative Europe)

UKGPUCC3 – Goodenough College, London – Wed. 14.Dec.11

ahart@cray.com

## Contents

- The Now: the new Cray XK6
  - "Accelerating the Way to Better Science"
- The Future: Heterogeneous computing and the Exascale
- Accelerator directives
  - Why do we need them?
  - What do they look like?
    - OpenACC now, OpenMP in the future
  - How do we use them?
  - How do we port a full application?
- How do they perform?
  - Case studies in directive-based optimisation on GPU
  - performance ~~vs.~~ *and* productivity

## "Accelerating the Way to Better Science"

Cray XK6 supercomputer

- HPCwire readers: "Top 5 New Products or Technologies to Watch"

- Nvidia Fermi 2090 GPU
  - 20% better performance than 2070
  - compute: 448→512 cores; 1.15→1.30 GHz clock
  - memory: 6GB; 150→178GB/s bandwidth
  - Upgradable to Kepler in 2012
- AMD Series 6200 Interlagos CPU (16 cores)
- Cray Gemini interconnect
  - high bandwidth/low latency scalability
  - HPCwire editors: "Best HPC Interconnect Product or Technology"
- Fully integrated/optimised/supported
  - Hardware and full software stack stack (including libraries)
  - Also supports Cray Cluster Compatibility Mode for ISV applications
- Fully blendable with Cray XE6 product line
  - HPCwire readers: "Best HPC Server Product or Technology"
- Fully upgradeable from Cray XT/XE systems

HPC wire Readers' Choice Awards 2011

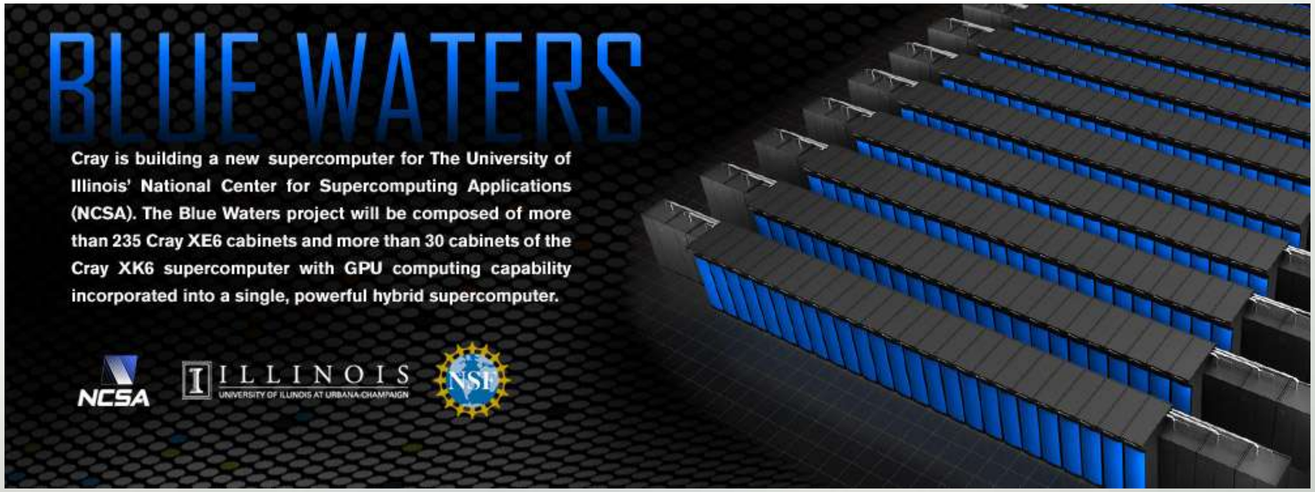
## Cray hybrids in future Top500

TITAN

Cray is upgrading the "Jaguar" XT5 supercomputer located at the Department of Energy's (DOE) Oak Ridge National Laboratory (ORNL) to a new Cray XK6 supercomputer, nicknamed "Titan". When completed, the Titan system will have a peak performance between 10 and 20 petaflops of high performance computing (HPC) power!

CRAY XK6
MORE >>
OAK RIDGE National Laboratory

ORNL Titan: 200 cabinets of Cray XK6

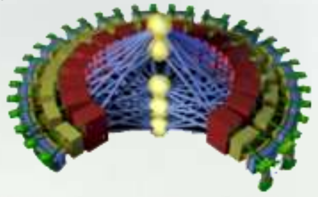


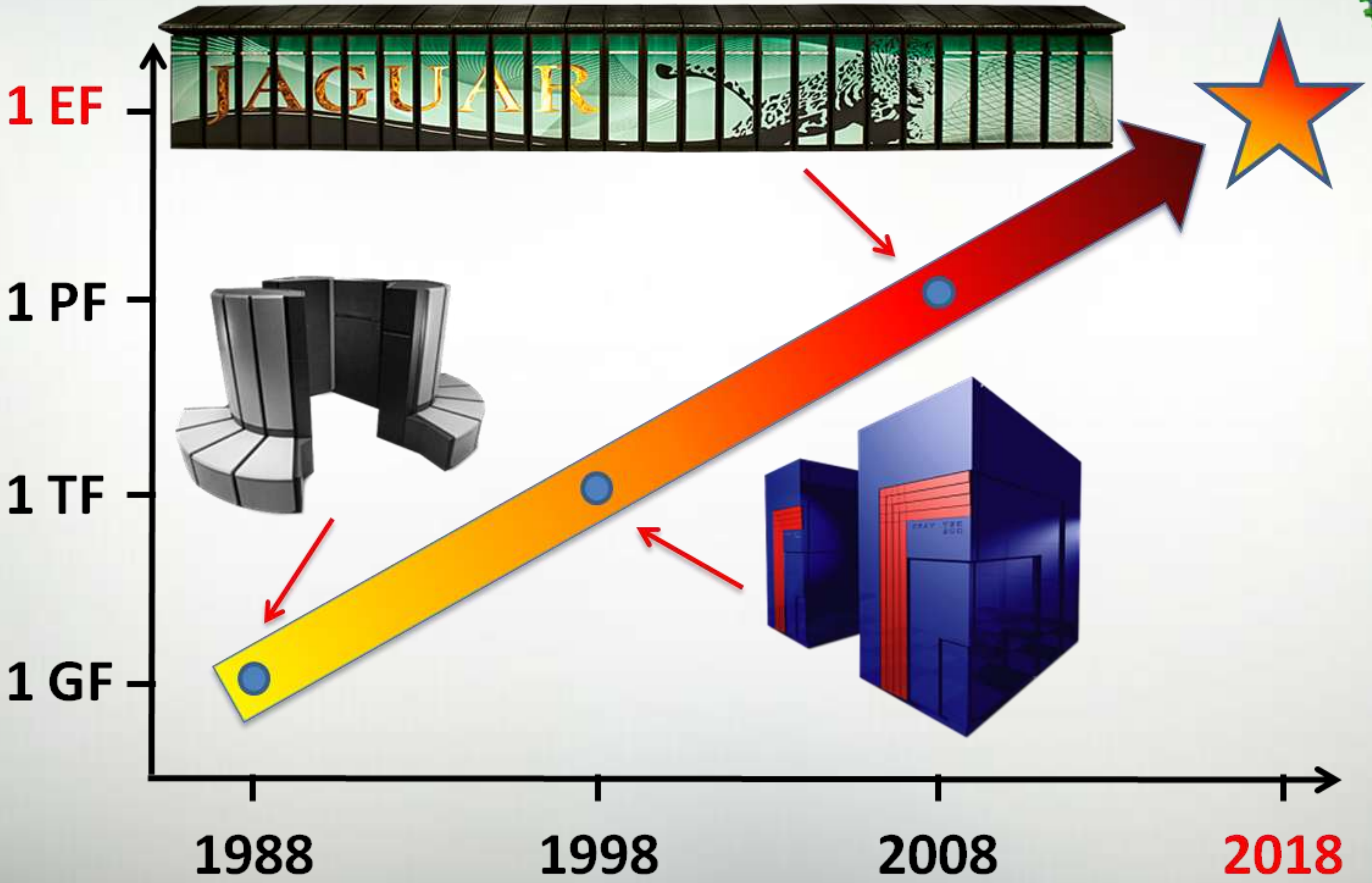
Cray is building a new supercomputer for The University of Illinois' National Center for Supercomputing Applications (NCSA). The Blue Waters project will be composed of more than 235 Cray XE6 cabinets and more than 30 cabinets of the Cray XK6 supercomputer with GPU computing capability incorporated into a single, powerful hybrid supercomputer.

NCSA Blue Waters: 235 cabinets of Cray XE6 + 30 cabinets of Cray XK6

## The Exascale is coming...

- Sustained performance milestones every 10 years...
  - 1000x the performance with 100x the PEs





(and they're all Crays)

## Exascale, not exawatts

- Power is a big consideration in an exascale architecture
  - Jaguar  (ORNL) draws 6MW to deliver 1PF
  - The US DoE demands 1EF from only 20MW (and $200M)
- A hybrid system is one way to reach this, e.g.
  - $10^5$ nodes (up from $10^4$ for Jaguar)
  - $10^4$ FPUs/node (up from 10 for Jaguar)
    - some full-featured cores for serial work
    - a lot more cutdown cores for parallel work
  - Instruction level parallelism will be needed
    - continues the SIMD trend SSE → AVX → ...
- This looks a lot like the current GPU accelerator model
  - manycore architecture, split into SIMT threadblocks
  - Complicated memory space/hierarchy (internal and PCIe)

- EU FP7 Network: “Exascale computing, software and simulation"
- Consortium has
  - Leading European HPC centres
    - EPCC, HLRS, CSC, PDC
  - Hardware partner
    - Cray
  - Tools providers
    - TUD (Vampir), Allinea (DDT)
  - Codesign application owners, specialists
    - ABO, JYU, UCL, ECMWF, ECP, DLR

CRESTA
Sverige
Sweden
Suomi
Finland
Gulf of Bothnia
Norway
Bergen
Oslo
Stockholm
Tallinn
Eesti
Estonia
Baltic Sea
Göteborg
Latvija
Latvia
North Sea
Glasgow
Edinburgh
United Kingdom
Belfast
Danmark
Denmark
Malmö
Lietuva
Lithuania
Gdansk
Vilnius
Ireland
Hamburg
Szczecin
Birmingham
London
Nederland
Berlin
Bydgoszcz
Polska
Poland
Hannover
Dresden
België
Belgique
Deutschland
Germany
Česká Rep
Czech Rep
Kraków
Celtic Sea
Ostrava
Slovensko
Slovakia
Freiburg im Breisgau
München
Wien
Österreich
Austria
France
Zürich
Vaduz
Zagreb
Magyarország
Hungary
Bay of Biscay
Lyon
Milano
Torino
Genova
Bologna
Hrvatska
Croatia
România
Romania
Beograd
Србија
Serbia
Marseille
Toulouse
Italia
Italy
Sarajevo
București
България
Vigo

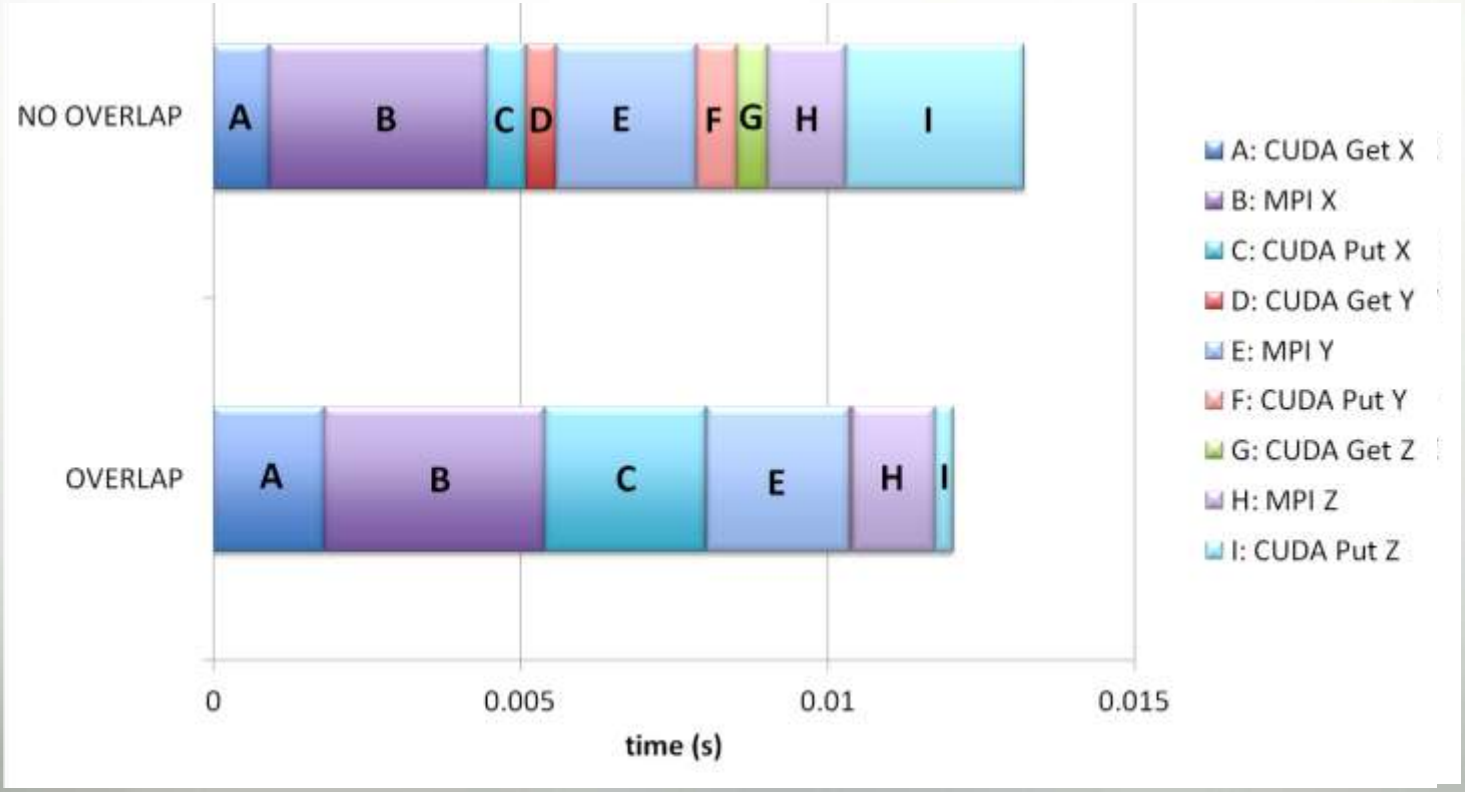
- CRESTA and its two partner projects are the first Exascale development projects funded by Europe
  - Run from Oct. 2011-Sept. 2014

## Accelerator programming

- Why do we need a new GPU programming model?
- Aren't there enough ways already?
  - CUDA (incl. PGI CUDA Fortran)
  - OpenCL
  - Stream
  - hiCUDA ...
- All are quite low-level and closely coupled to the GPU
  - User needs to rewrite kernels in specialist language:
    - Hard to write and debug
    - Hard to optimise for specific GPU
    - Hard to port to new accelerator
  - Multiple versions of kernels in codebase
    - Hard to add new functionality

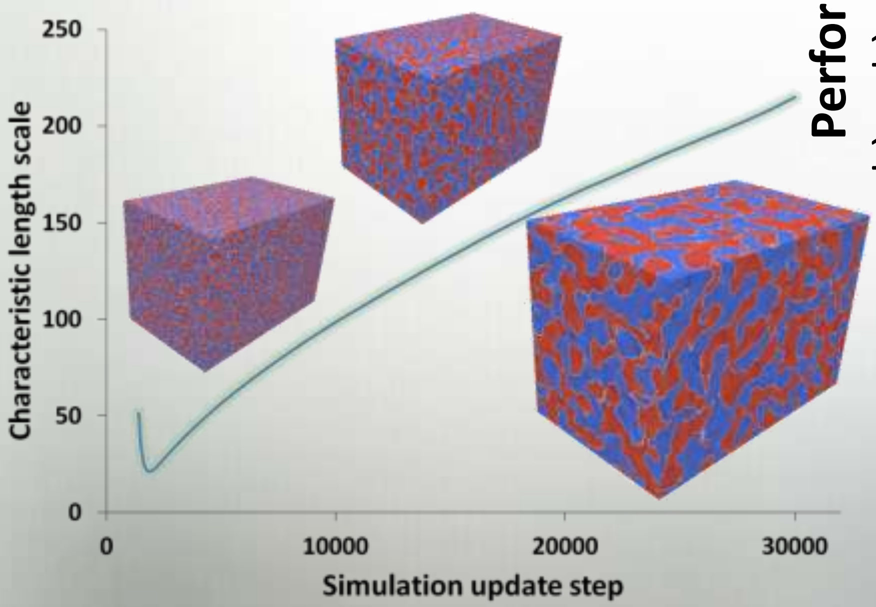
# CUDA on hybrid supercomputers

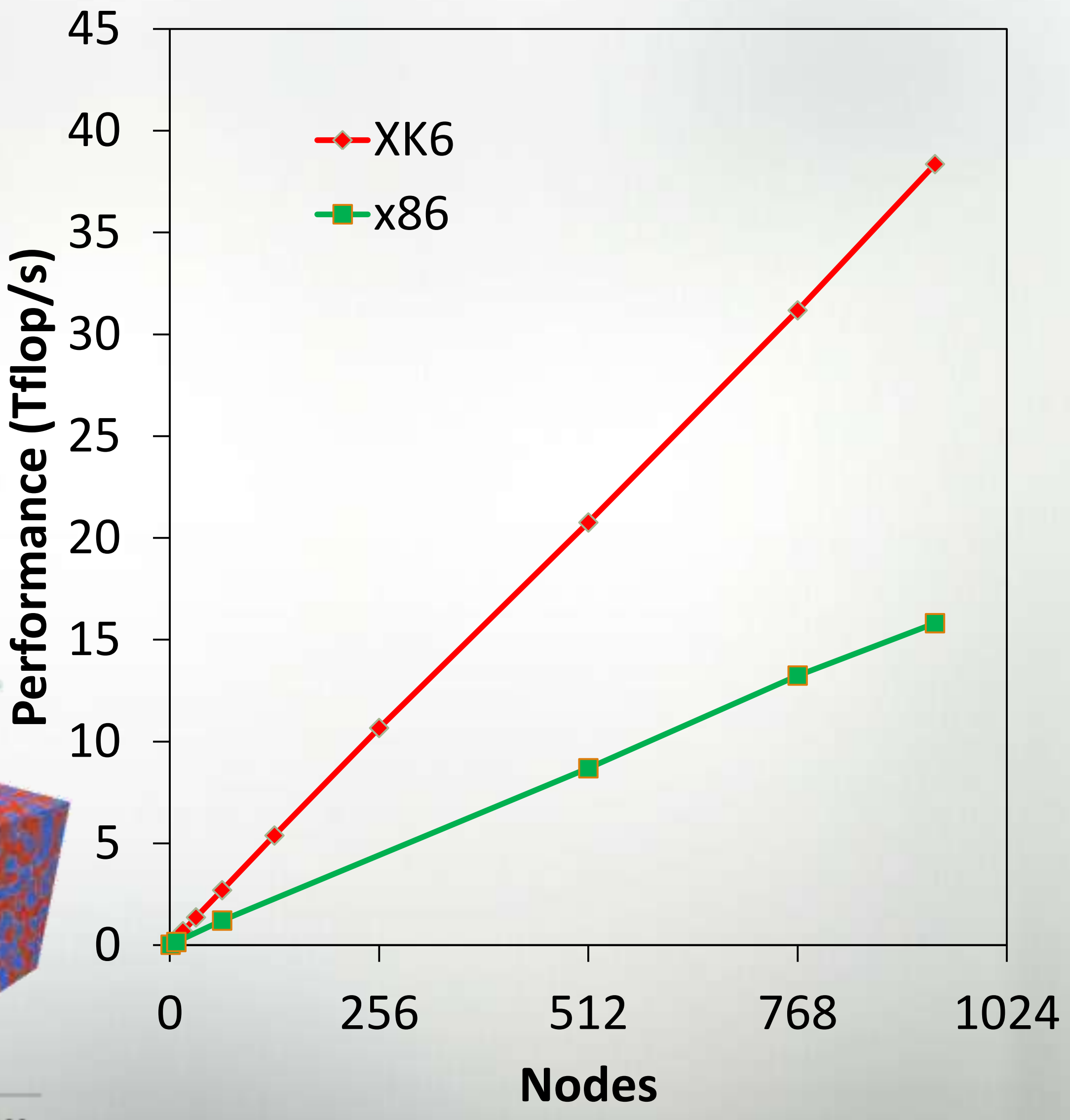
- If you work hard, you can get good parallel performance
- Ludwig Lattice Boltzmann code rewritten in CUDA
  - Reordered all the data structures (structs of arrays)
  - Pack halos on the GPU
  - Streams to overlap compute, PCIe comms, MPI halo swaps



## Ludwig weak scaling

- 10 cabinets of Cray XK6
    - 936 GPUs (nodes)
- Only 4% deviation from perfect scaling between 8 and 936 GPUs.
- Application sustaining 40+ Tflop/s



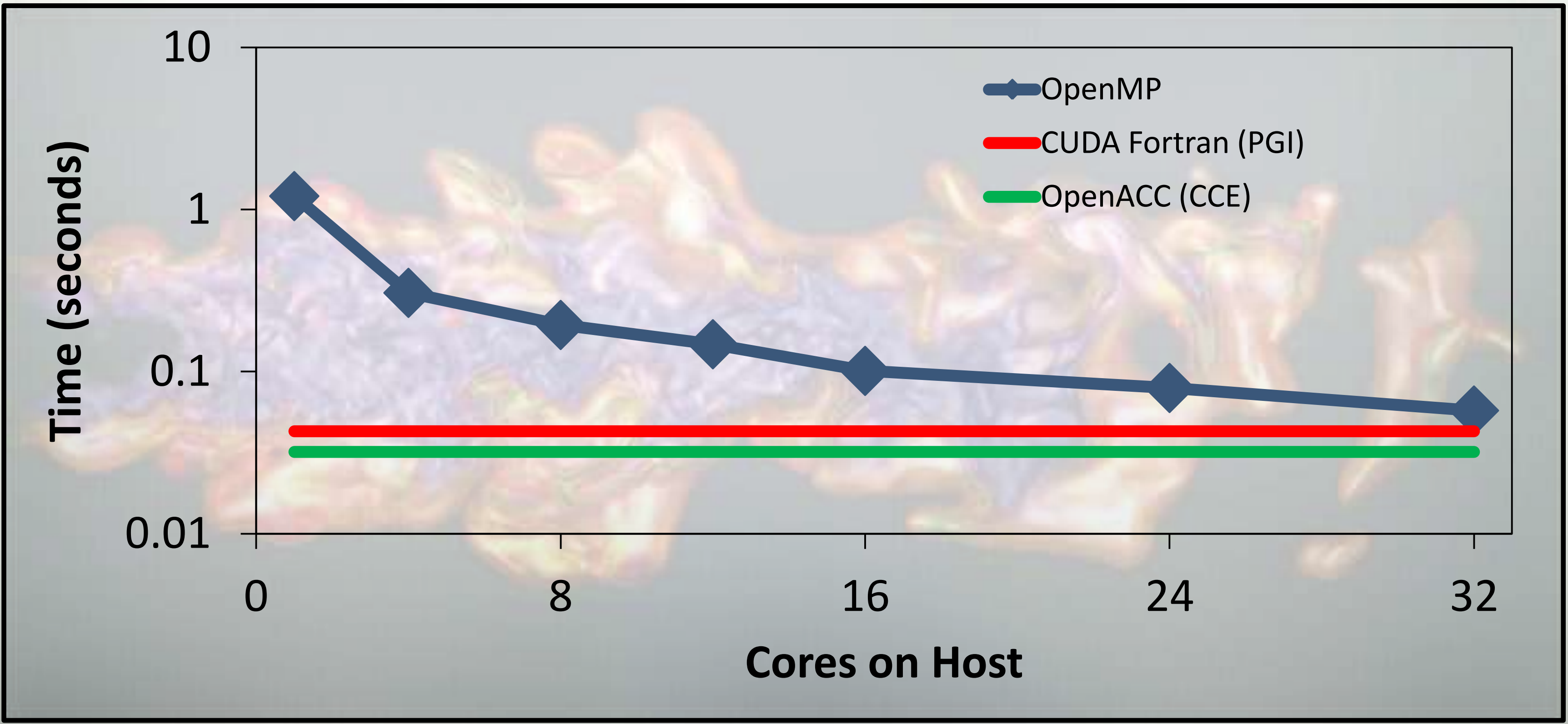


## Directive-based programming

- Most scientific applications will not have this level of developer support (Ludwig was special case)
- Directives provide high-level approach
  + Based on original source code (e.g. Fortran, C, C++)
    + Easier to maintain/port/extend code
    + Users with (for instance) OpenMP experience find it a familiar programming model
    + Compiler handles repetitive boilerplate code (cudaMalloc, cudaMemcpy...)
    + Compiler handles default scheduling; user can step in with clauses where needed
  – Possible performance sacrifice
    – Important to quantify this
    – Can then tune the compiler
    – Small performance sacrifice is an acceptable trade-off for portability and productivity
      – Who handcodes in assembler these days?
- Two relevant performance comparisons:
  - How does the performance compare to CUDA?
  - Can I justify buying a GPU instead of another CPU?

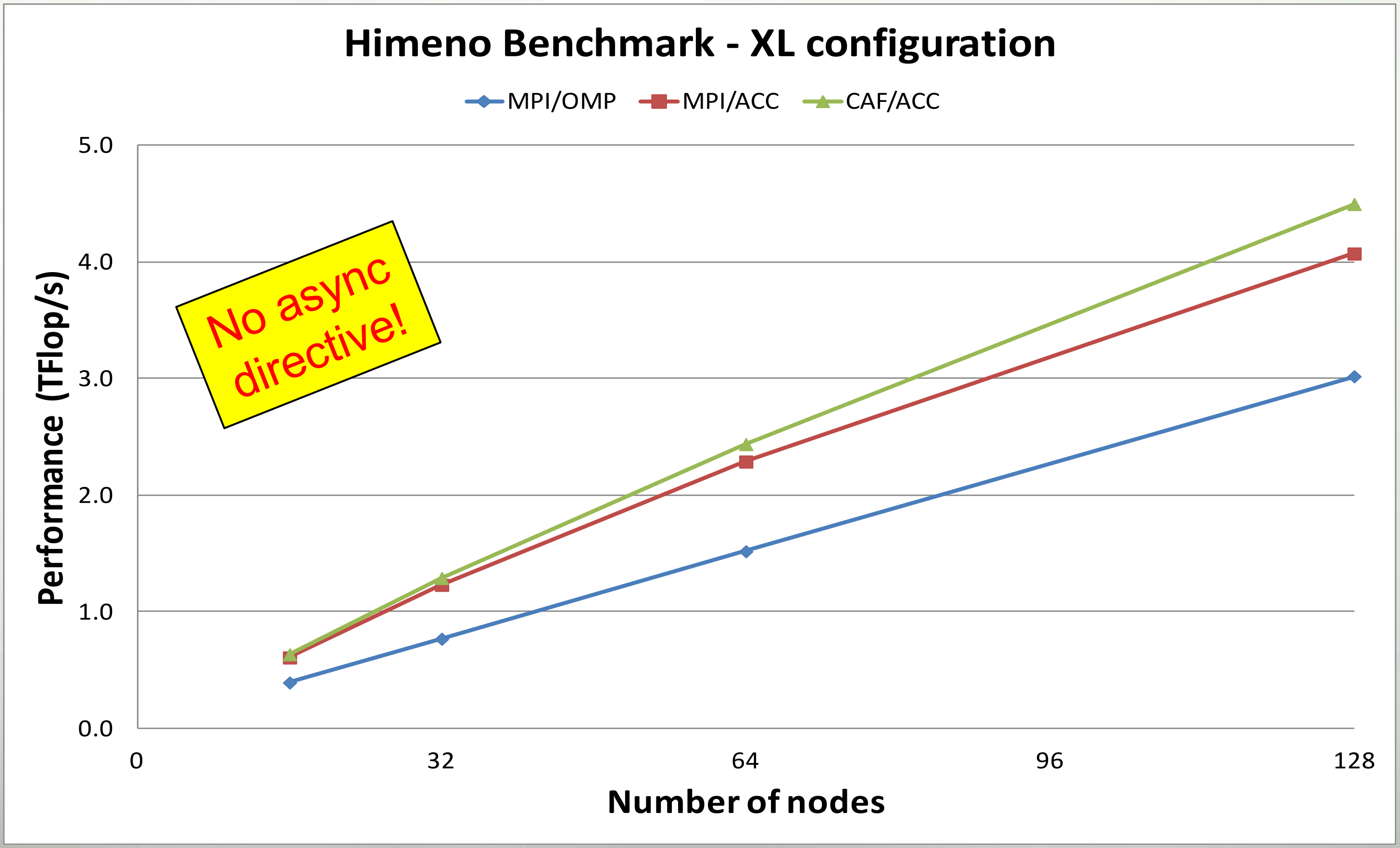
# Performance compared to CUDA

- Is there a performance gap relative to explicit low-level programming model? Typically 10-15%, sometimes none.
- Is the performance gap acceptable? Yes.
  - e.g. S3D comp_heat kernel (ORNL application readiness):



# Node-for-node performance comparison

- Does accelerated parallel application performance justify buying a GPU (Cray XK6) rather than another CPU (Cray XE6)?
  - For many codes, yes.



OpenACC.
DIRECTIVES FOR ACCELERATORS

- A common directive programming model for today's GPUs
  - Announced at SC11 conference
  - Offers portability between compilers
    - Drawn up by: NVIDIA, Cray, PGI, CAPS
    - Multiple compilers offer portability, debugging, permanence
  - Works for Fortran, C, C++
    - Standard available at www.OpenACC-standard.org
    - Initially implementations targeted at NVIDIA GPUs
- Current version: 1.0 (November 2011)
- Compiler support:
  - Cray CCE: partial now, complete in 2012
  - PGI Accelerator: released product in 2012
  - CAPS: released product in Q1 2012

**The OpenACC™ API**
**QUICK REFERENCE GUIDE**

The OpenACC Application Program Interface describes a collection of compiler directives to specify loops and regions of code in standard C, C++ and Fortran to be offloaded from a host CPU to an attached accelerator, providing portability across operating systems, host CPUs and accelerators.

Most OpenACC directives apply to the immediately following structured block or loop; a structured block is a single statement or a compound statement (C or C++) or a sequence of statements (Fortran) with a single entry point at the top and a single exit at the bottom.

Version 1.0, November 2011

All four companies intend to work within OpenMP organization to merge OpenACC and create a common specification that extends OpenMP to support accelerators.

© 2011 OpenACC-standard.org all rights reserved.

CAPS CRAY THE SUPERCOMPUTER COMPANY NVIDIA. The Portland Group

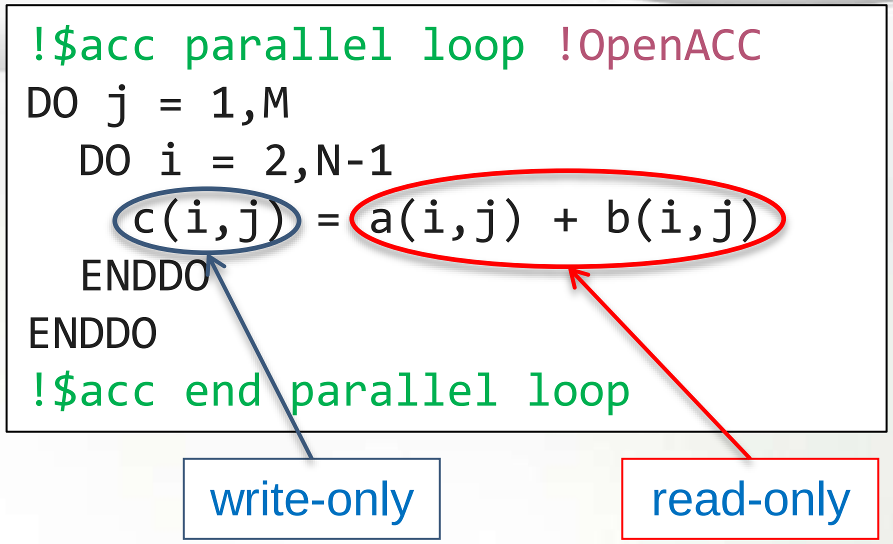
# Accelerator directives

- Modify original source code with directives
  - Non-executable statements (comments, pragmas)
    - Can be ignored by non-accelerating compiler
  - Sentinel: !$acc
  - Fortran:
    - Usually paired with !$acc end *
  - C/C++:
    - Structured block {...} avoids need for end directives
  - Continuation to extra lines allowed
- CPP macros defined to allow extra conditional compilation
  - E.g. around calls to runtime API functions
    - _OPENACC == yyyymm (currently 201111)

```
! Fortran example
!$acc *
<structured block>
!$acc end *
```

```
/* C/C++ example */
#pragma acc *
{structured block}
```

# A first example



```
!$acc parallel loop !OpenACC
DO j = 1,M
  DO i = 2,N-1
    c(i,j) = a(i,j) + b(i,j)
  ENDDO
ENDDO
!$acc end parallel loop
```

Execute a loop nest on the GPU

- Compiler does the work:
  - Data movement
    - allocates/frees GPU memory at start/end of region
    - moves of data to/from GPU
  - Loop schedule: spreading loop iterations over PEs of GPU

| **Parallelism** | **Nvidia GPU** | **SMT node** |
|---|---|---|
| Gang: | a threadblock | CPU |
| Worker: | warp (32 threads) | CPU core |
| Vector: | SIMT group of threads | SIMD instructions (SSE, AVX) |

  - Caching (explicitly use GPU shared memory for reused data)
    - automatic caching (e.g. NVIDIA Fermi) important
  - Tune default behaviour with optional clauses on directives

## Sharing GPU data between subprograms

```
PROGRAM main
  INTEGER :: a(N)
!$acc data copy(a)
!$acc parallel loop
  DO i = 1,N
   a(i) = i
  ENDDO
!$acc end parallel loop
  CALL double_me(a)
!$acc end data
END PROGRAM main
```

```
SUBROUTINE double_me(b)
  INTEGER :: b(N)
!$acc parallel loop present(b)
  DO i = 1,N
   b(i) = 2*b(i)
  ENDDO
!$acc end parallel loop
    END SUBROUTINE double_me
```

- data region spans two accelerator parallel regions
    - One happens to be inside a subroutine call here (which could be in separate source file)
- The present clause uses version of b on GPU without data copy
    - Can also call double_me() from outside a data region
        - Replace present with present_or_copy (can be shortened to pcopy)
- Original calltree structure of program can be preserved
- Similar data region constructs in other directive models

## Clauses for !$acc parallel loop

- Data clauses:
  - copy, copyin, copyout
    - copy moves data "in" to GPU at start of region and/or "out" to CPU at end
    - supply list of arrays or array sections (using Fortran ":" notation)
  - create
    - No copyin/out – useful for shared temporary arrays in loopnests
  - private: scalars private by default
  - present, present_or_copy*
- Tuning clauses:
  - !$acc loop [gang] [worker] [vector]
    - Targets specific loop (or loops with collapse clause) at specific level of hardware
  - num_gang, num_workers, vector_length
    - Tunes the amount of parallelism used (threadblocks, threads/block...)
  - seq: loop executed sequentially
  - independent: compiler hint (also use CCE !dir$ directives)

# More OpenACC directives

- Other !$acc parallel loop clauses:
  - if(logical)
    - Executes on GPU if .TRUE. at runtime, otherwise on CPU
  - reduction: as in OpenMP
  - cache: specified data held in software-managed data cache
    - e.g. explicit blocking to shared memory on Nvidia GPUs
- !$acc update [host|device]
  - Copy specified arrays (slices) within data region
- async[(handle)] clause for parallel, update directives
    - Launch accelerator region/data transfer asynchronously: allows CPU/GPU overlap
    - Operations with same handle will execute sequentially (as in CUDA streams)
    - !$acc wait[(handles)]: waits for completion
    - Runtime library functions can also be used to test/wait for completion
- host_data, deviceptr
    - Exposes GPU memory address in host code (e.g. for interoperability with CUDA)

## A porting strategy

- Preparation: add checksum(s) and high-res timer to code
    - Check for correctness very frequently
  - Profile code on the host
    - Use representative-sized problem, map calltree,
    - Ideally resolve profile by loopnest and measure typical loop iteration counts
- First optimise the data movements
  - Start in subprograms at bottom of callchain
    - Accelerate individual loopnests using parallel regions
      - Concentrate initially on most computationally expensive
    - Add data regions in subprograms
      - Minimise data movements, use create clause where possible
      - May need to accelerate insignificant loopnests to avoid data copies
  - Use available feedback to understand data movement
    - Compiler messages: -ra for CCE, -Minfo=accel for PGI
    - Runtime commentary: export CRAY_ACC_DEBUG=[1,2,3] for CCE
    - Nvidia compute profiler: export COMPUTE_PROFILE=1
    - CrayPAT performance measurement and analysis tool (Cray PE only)
- Code is probably going quite slowly at this point

## A porting strategy (2)

- Move progressively up callchain, adding data regions
  - Aim to further reduce data movements
    - No problem nesting data regions: use present clause on inner ones
    - May need to port insignificant subprograms to avoid data transfers
    - Use update for essential data transfers (e.g. data for halo swaps)
- Now optimise kernel performance (often trial and error)
  - Perfect loop nests schedule better than imperfect ones
    - e.g. Remove temporary arrays by manually inlining (eliminate array b)
    - Or manually privatise arrays and break loopnest (make b(i,j))

```
DO j = 1,N
 DO i = 0,M+1
  b(i) = a(i,j+1) + a(i,j-1)
 ENDDO
 DO i = 1,M
  c(i,j) = b(i+1) + b(i-1)
 ENDDO
ENDDO
```

```
DO j = 1,N
 DO i = 1,M
  c(i,j) = a(i+1,j+1) + a(i+1,j-1) &
         + a(i-1,j+1) + a(i-1,j-1)
 ENDDO
ENDDO
```

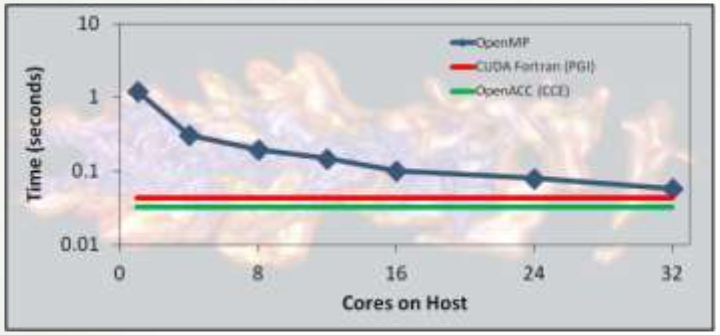
```
DO j = 1,N
 DO i = 0,M+1
  b(i,j) = a(i,j+1) + a(i,j-1)
 ENDDO
ENDDO
DO j = 1,N
 DO i = 1,M
  c(i,j) = b(i+1) + b(i-1)
 ENDDO
ENDDO
```

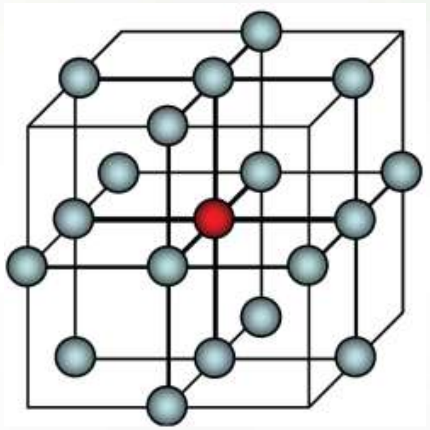
## A porting strategy (3)

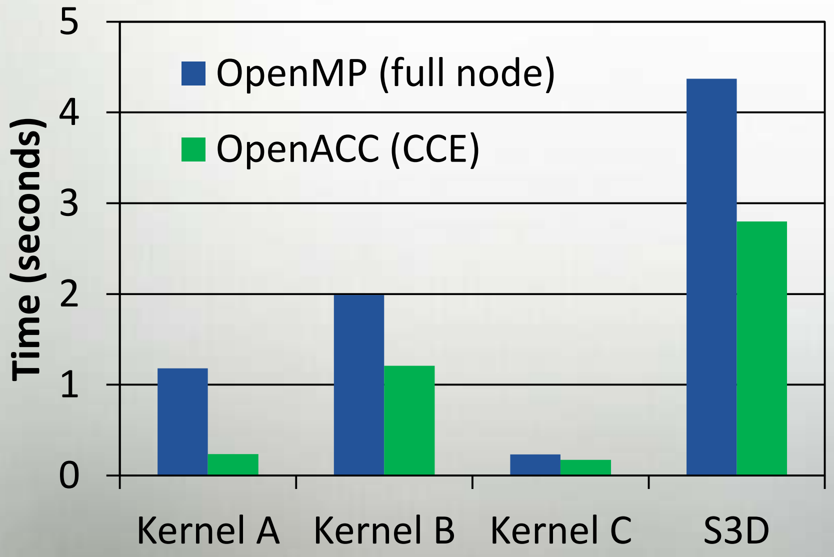
- Now look at tweaking the loop scheduling
  - Quick wins
    - Optimise loop scheduling
      - Make sure the right loops are vectorised (for coalesced memory loads)
        - And that they are vectorisable
      - Choose number of workers per gang (threads/block)
        - This number will vary by kernel and by problem size
        - Collapsing or blocking of loops may help (though compilers already do that)
    - See if caching can be used to reduce data loads from device memory
  - Longer term: can loops be migrated up the callchain?
    - E.g. Loop over sites, or blocks of sites ("blocking for cache")
    - If so, parallelise (gangs) over these
- Consider overlap of compute and communications using async
  - Don't do this until everything working
  - May require application restructuring

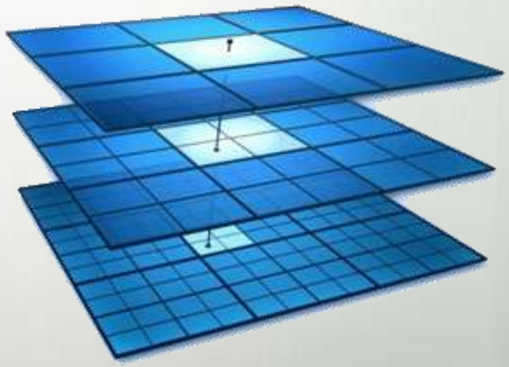
# Three example applications

1. S3D turbulent combustion code
2. Himeno
3. MultiGrid code (NAS & SPEC benchmarks)



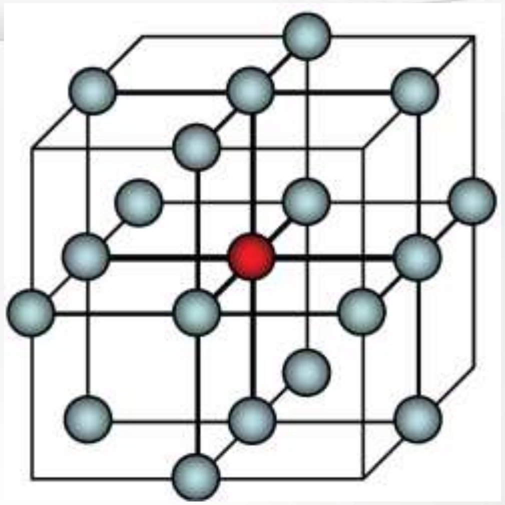


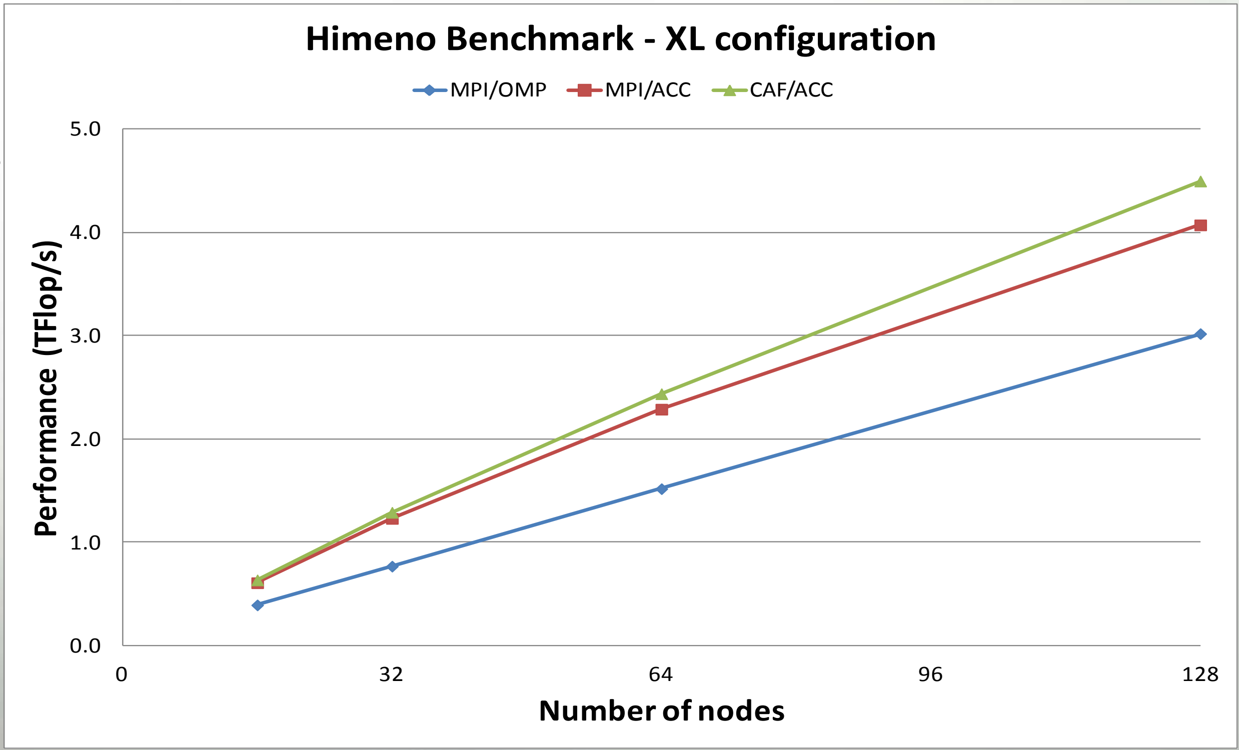




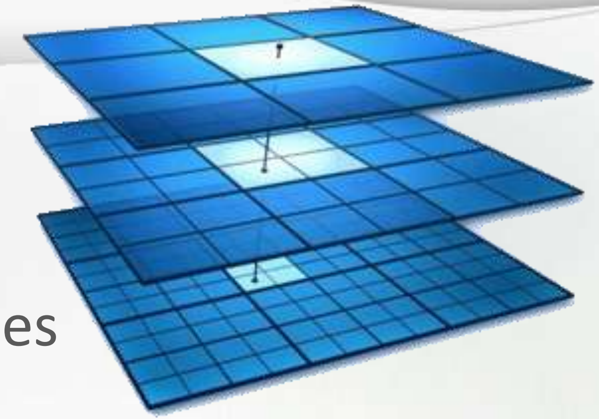
# Example: The Himeno Benchmark

- Parallel 3D Poisson equation solver
  - 19-point stencil
- MPI or CAF and/or OpenMP
  - available from here
  - ~600 lines of Fortran
  - Fully ported to accelerator using 27 directive pairs
- XL configuration:
  - 1024 x 512 x 512
  - Strong scaling
- More kernel tuning
- No use of async yet





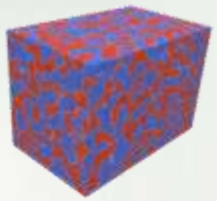
## Example: MultiGrid benchmark

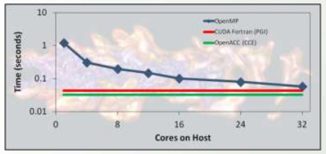


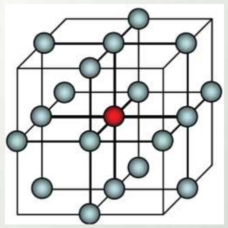
- NAS Parallel Benchmarks, also SPEC suite
- MG (multigrid) solves Laplacian on 3D grid
  - c. 1500 lines of Fortran, many subroutines
  - Three main hotspots:
    - **resid** (50% of runtime), **psinv** (25%), **rprj3** (9%)
  - Data arrays passed to/from subroutines at every iteration
- GPU 2x faster than CPU (16 cores)
  - Fully accelerated using 25 directive pairs (present essential)
- MPI-parallel version: Cray XK6 node faster than
- Further optimisations coming
  - Further use of shared memory
  - async clause support coming
    - CCE already launches kernels and data transfers asynchronously
    - More scope for overlap than in Himeno

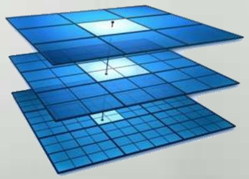
## In conclusion...

- Hybrid multicore has arrived and is here to stay
  - Fat nodes are getting fatter
  - GPUs have leapt into the top500 and accelerated nodes
- Programming accelerators efficiently is hard
  - When done well can give good performance (Ludwig)
- Accelerator directives offer a good alternative
  - Attractive (and familiar) programming model
  - Open standards for portability
  - Use original Fortran, C and C++ codes
- Presented a strategy for porting large codes
  - The performance penalty is small
  - The portability and productivity bonuses are huge
- Directives play nicely with other programming models
  - (so you don't need to throw away your prize CUDA kernels)









## Acknowledgments

Thank you to those that helped us get to grips with directives:

- Cray Exascale Research Initiative Europe team
  - Harvey Richardson, Roberto Ansaloni
- EPCC Exascale Technology Centre team
  - Alan Gray...
- Cray PE R&D team
  - Luiz DeRose, Suzanne LaCroix, James Beyer, David Oehmke...
- ORNL team
  - John Levesque...
- OpenMP subcommittee

For further info, ahart@cray.com