Programming hybrid architecture with OpenMP accelerator directives

Alistair Hart

Cray Exascale Research Initiative Europe

Future computing for particle physics

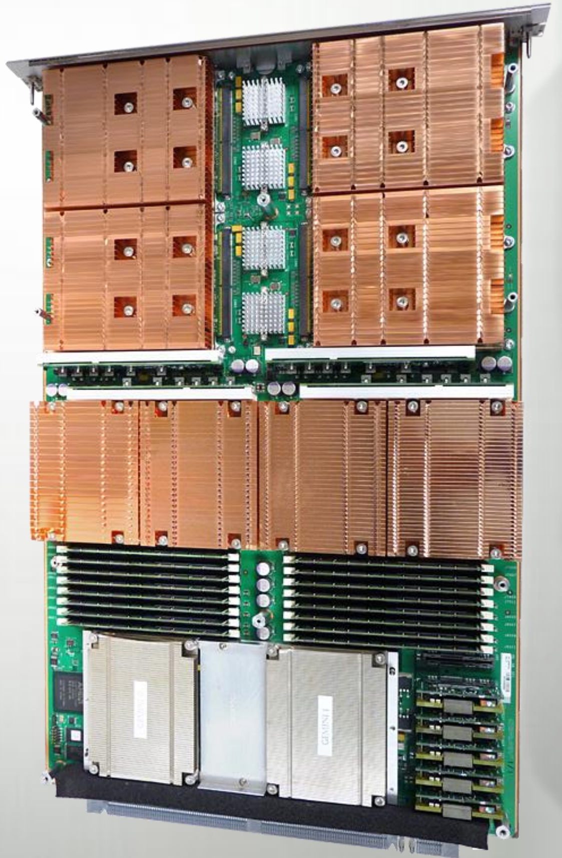
e-Science Institute (Edinburgh)

17.Jun.11

ahart@cray.com

"Accelerating the Way to Better Science"

- Cray has now announced the new Cray XK6
  - Next generation Nvidia Fermi X2090 GPU
    - 20% better performance than 2070
    - compute: 448→512 cores; 1.15→1.30 GHz clock
    - memory: 6GB; 150→178GB/s bandwidth
  - Next generation AMD Interlagos CPU
  - Cray Gemini interconnect
    - high bandwidth/low latency scalability
  - Fully blendable with Cray XE6 product line
  - Fully upgradeable from Cray XT/XE systems



- Longer term, GPUs are template a

  for Exascale HPC architectures

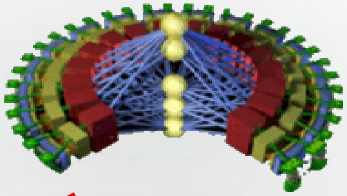
Cray Exascale Research Initiative Europe

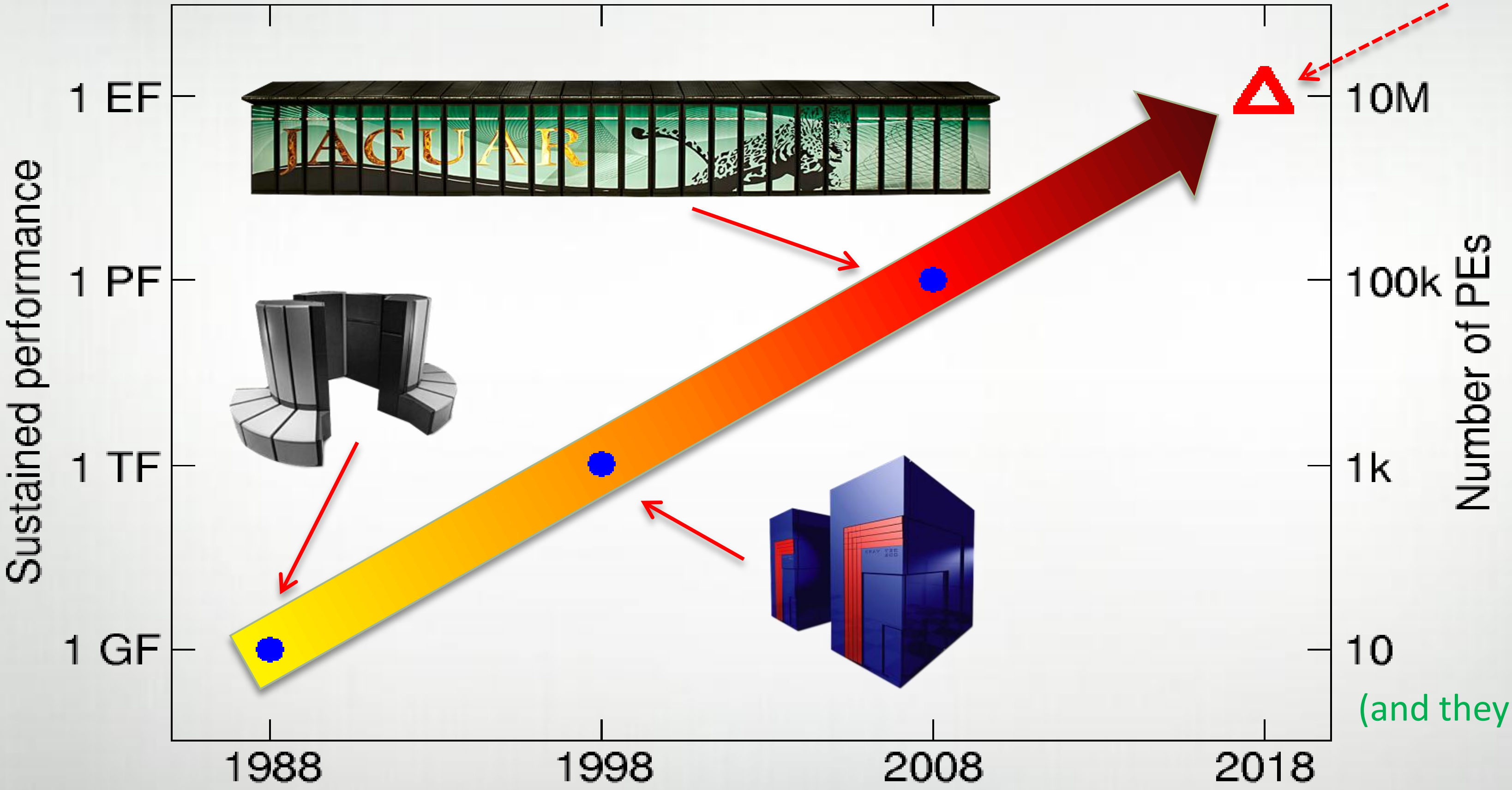
- Launched December 2009
  - Initial research partners are EPCC, CSCS and HLRS
  - Cray increased EMEA team in Edinburgh, Lugano and Stuttgart
- Exploring how real applications can exploit future Exascale architectures, specifically:
  - Programming models for PGAS languages
  - Programming GPU accelerators
  - Improved algorithms for FFTs
  - Network and I/O profiling
- Strong interactions with Cray R&D
- 2011: Cray in negotiation with the EU for "Exascale computing, software and simulation" network

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

The road to Exascale

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





(and they're all Crays)

- Can't scale power the same (US DOE: "6→20MW only")
  - So we need lower-power, higher-performing PEs, e.g. GPUs

Unified X86/GPU programming environment

- Cray XK6 includes the first-generation Cray Unified X86/GPU PE

- Why is Cray putting so much effort into this?
  - It is hard to get good performance from hybrid systems
  - A good PE narrows the gap between observed and ~~peak~~ achievable performance

- It will support three classes of users:
  1. "hardcore" GPU programmers with existing CUDA ports
  2. users with parallel codes and OpenMP experience, but less GPU knowledge
  3. users with serial codes looking for portable parallel performance with and without GPUs

Cray XK6 PE: accelerator-specific components

Cray supplied
(Cray specific)

- Compilers: Cray Compilation Environment (CCE)
  - Standards-compliant: Fortran (incl. CAF), C (incl. UPC), C++
  - Accelerator usage: automatic and via OpenMP accelerator directives
  - 30 years' vectorising experience in CCE is great for generating GPU code
- Libraries: e.g. Cray libsci (BLAS, FFT)
  - Cray's Autotuning Framework especially important for tuning GPUs
  - e.g. DGEMM 30% faster than cuBLAS
- Performance Analysis: Cray Optimisation Explorer, CrayPAT
  - Scoping tool to help users port and optimise applications
  - Loop statistics information in CrayPAT (important for tuning OpenMP)
  - Whole application profiling: coherent view of: CPU, GPU, comms, I/O

Cray supported
(third party)

- Compilers: NVIDIA CUDA, PGI Accelerator
- Debuggers: Allinea DDT, Totalview
- Libraries: cuBLAS, cuFFT, MAGMA, PETSc, Trilinos etc.
- Performance tools: NVIDIA Compute and Visual profilers

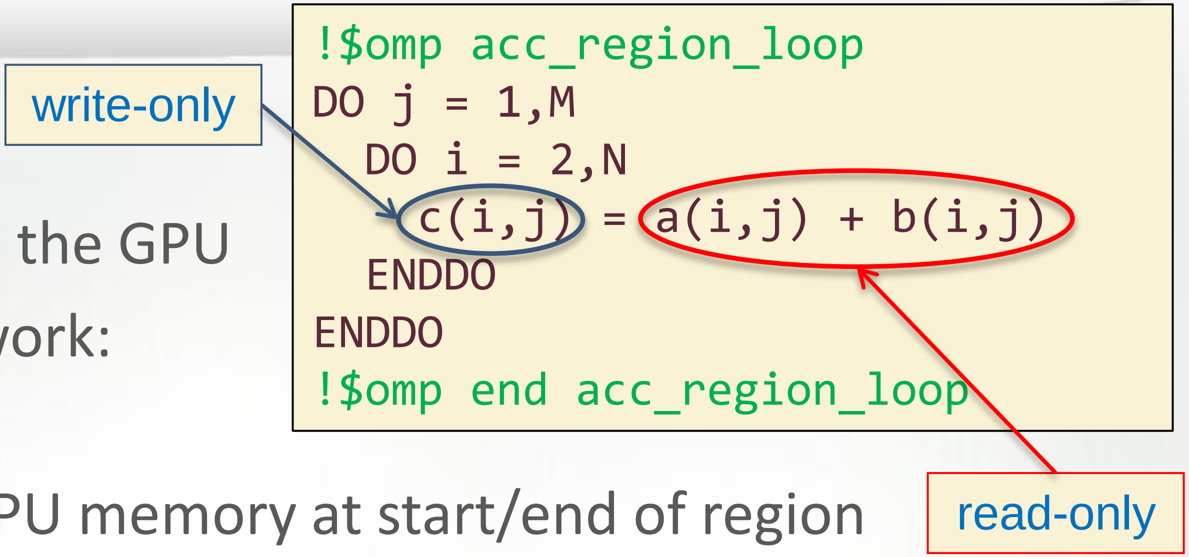
Accelerator programming

- Why a new model? There are already many ways to program:
  - CUDA, PGI CUDA Fortran, OpenCL...
  - All are quite low-level and closely coupled to the GPU
- User needs to write specialist kernels:
  - Hard to write and debug
  - Hard to optimise for specific GPU
  - Hard to update (porting/functionality)
- Directives provide high-level approach
  - + Based on original source code
    - + Easier to maintain/port/extend code (especially if original developer has moved on)
    - + Users with OpenMP experience find it a familiar programming model
    - + You run the same code on multicore CPU
  - – Possible performance sacrifice
    - A ***small*** performance gap is acceptable (do you still hand-code in assembler?)
    - + Goal is within 10% of CUDA (already seeing this in many cases, more tuning ongoing)

OpenMP accelerator directives

- An open standard is the most attractive for developers
  - portability; multiple compilers for debugging; permanence
- An established standards committee is better than a new body
- Subcommittee of OpenMP ARB, aiming for OpenMP 4.0
  - includes most major vendors
    - PGI, CAPS, Intel, IBM... + other interested parties (e.g. EPCC)
  - co-chaired by Cray (James Beyer)
- Cray is an enthusiastic supporter
  - CCE is first full implementation
  - Fortran, C, C++
  - Will track standard as it evolves
- Compiles straight to PTX
  - Preserves GPU debugging symbols (DDT...)
- Expect other vendors to follow lead

A first example



```
!$omp acc_region_loop
DO j = 1,M
  DO i = 2,N
    c(i,j) = a(i,j) + b(i,j)
  ENDDO
ENDDO
!$omp end acc_region_loop
```

write-only

read-only

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
  - division of iterations between SIMT/MIMD units of GPU
- Cache usage
  - Explicit use of GPU shared memory for reused data
    - automatic caching (e.g. NVIDIA Fermi) important
- Tune default behaviour with optional clauses on directives

Directive clauses

- Data clauses:
  - acc_copy, acc_copyin, acc_copyout, acc_shared
    - e.g. copy moves data "in" to GPU at start of region and "out" to CPU at end
    - supply list of arrays or array sections (using Fortran ":" notation)
  - present: share GPU-resident data between kernels
- Tuning clauses:
  - num_pes, cache, collapse...
    - optimise GPU occupancy, register and shared memory usage, loop scheduling...
- Some other important clauses:
  - async: Launch accelerator region asynchronously
    - allows overlap of GPU computation/PCI transfers with CPU computation/network
  - acc_call: Call external libraries or CUDA kernels
    - optionally using data already resident on the GPU
  - hetero: split loop iterations between CPU and GPU

Data regions to hold data on GPU

```
PROGRAM main
  REAL :: a(N)

!$omp acc_data acc_shared(a)
!$omp acc_region_loop
  DO i = 1,N
    a(i) = i
  ENDDO
!$omp end acc_region_loop
  CALL double_me(a)
!$omp end acc_data
END PROGRAM main
```

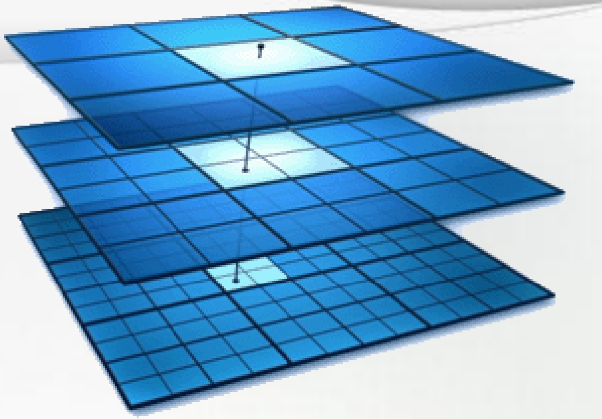
```
SUBROUTINE double_me(b)
  REAL :: b(N)

!$omp acc_region_loop present(b) &
!$omp&                acc_copy(b)
  DO i = 1,N
    b(i) = 2*b(i)
  ENDDO
!$omp end acc_region_loop


END SUBROUTINE double_me
```

- data region spans two accelerator regions
  - The acc_region checks at runtime if b is already on GPU:
    - yes: it uses this without copies; no: it follows the acc_copy(b) clause
  - Can also call double_me() from outside a data region
- Do not need to inline the subroutine (manually or by compiler)
  - Can even be in different source file

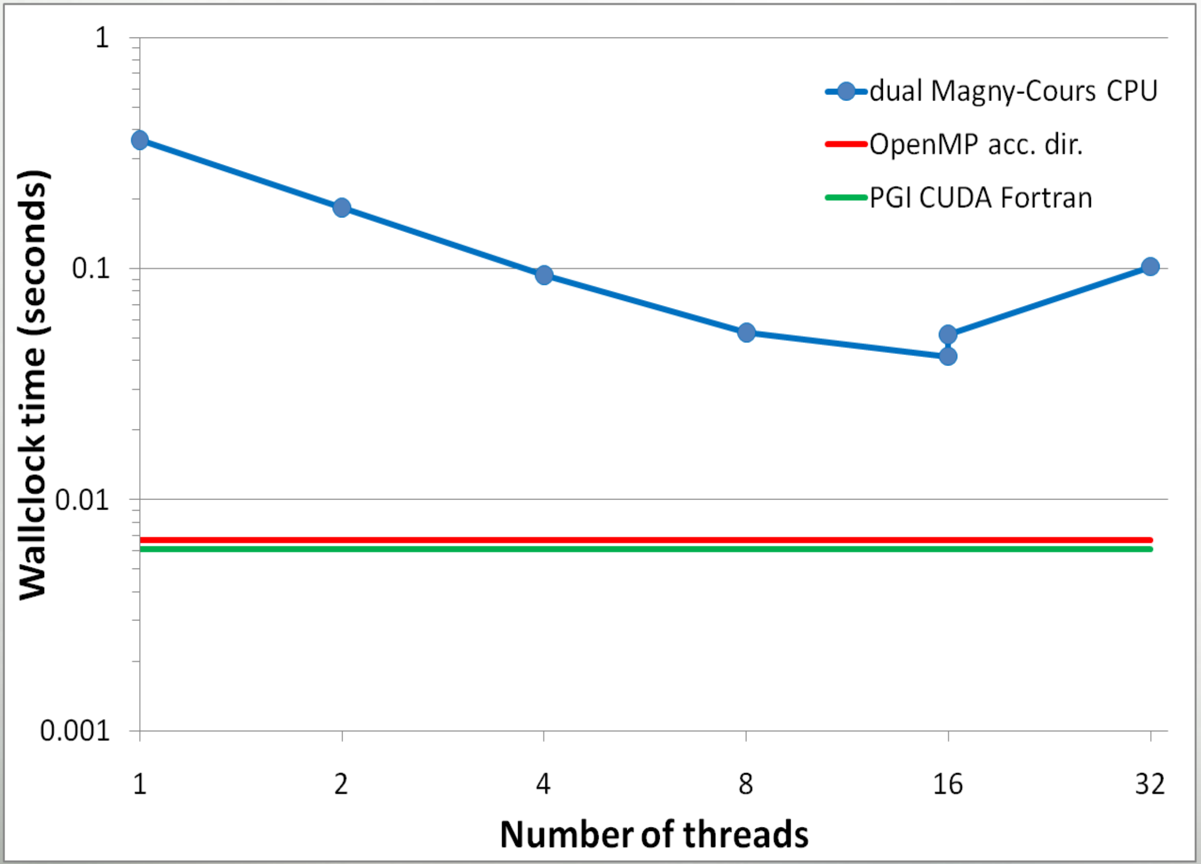
Scalar examples: benchmarks

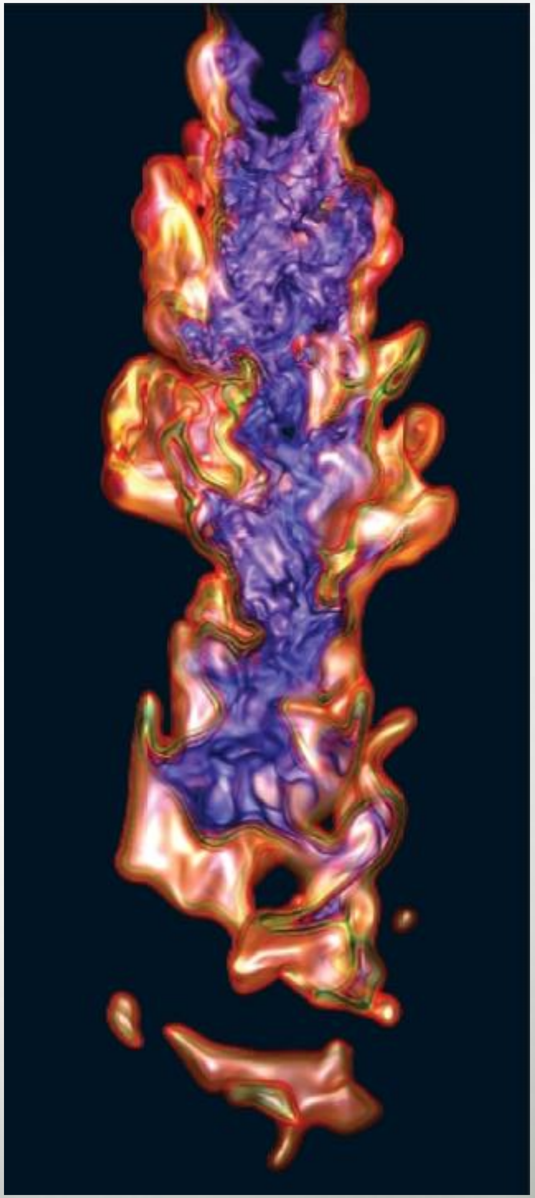


- NAS Parallel Benchmarks and SPEC suite
- MG (multigrid) solves Laplacian on 3D grid
  - Hotspots: resid (50% of runtime); psinv (25%); rprj3 (9%)
    - Data arrays passed to/from subroutines at every iteration
  - Whole application ported (25 directive pairs for 1500 lines)
    - present clause essential to eliminate data movement costs
  - GPU 50% faster than 12-core AMD Magny-Cours CPU
    - Even before compiler starts to use GPU shared memory
- CG (conjugate gradient)
  - whole application ported (19 directive pairs for 1200 lines)
    - less than 1 hour's work (from first sight of code)
  - GPU 15% faster than 12-core AMD Magny-Cours CPU
    - more tuning is possible

Real kernel example: S3D turbulent combustion

- 3d simulation of HCCI combustion
- detailed chemical kinetics (60 species)
- very important for low emission engines burning second-generation biofuels





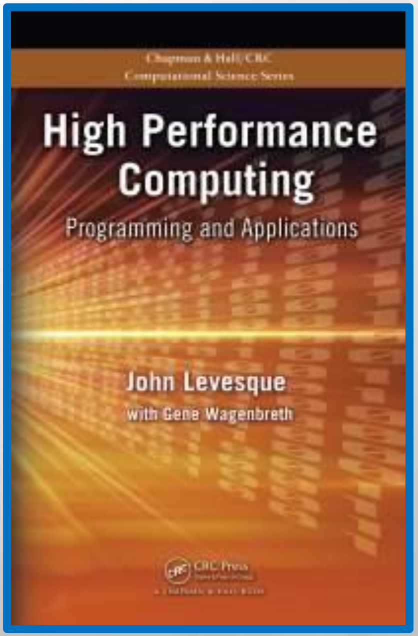
Parallel example: Himeno Poisson solver

- Real problems have comms (e.g. halo swaps) with PCIe transfers
- Do comms kill the node-for-node performance comparison? No!
- OMP ACC directives work with MPI, CAF, SHMEM (& OpenMP!)

- Again, this code will benefit from ongoing GPU stencil tuning

Porting a pure-MPI code to hybrid multicore

- A good hybrid OpenMP code can (should) outperform a pure MPI code on modern multi-/many-core CPUs
  - (once MPI scaling is exhausted)
- Given a good OpenMP code, a compiler can (should) generate efficient GPU code
- "Good" is measured as:
  - OpenMP loops moved as far up the call-chain as possible
  - Low-level loops vectorisable
- We have a **four task strategy** to achieve this:
  1. Identification of potential accelerator kernels
  2. Parallel Analysis, Scoping and Vectorisation
  3. Correctness Debugging
  4. Fine tuning of the accelerated program
- Users need tools to help with this



Chapman & Hall/CRC Computational Science Series

High Performance Computing

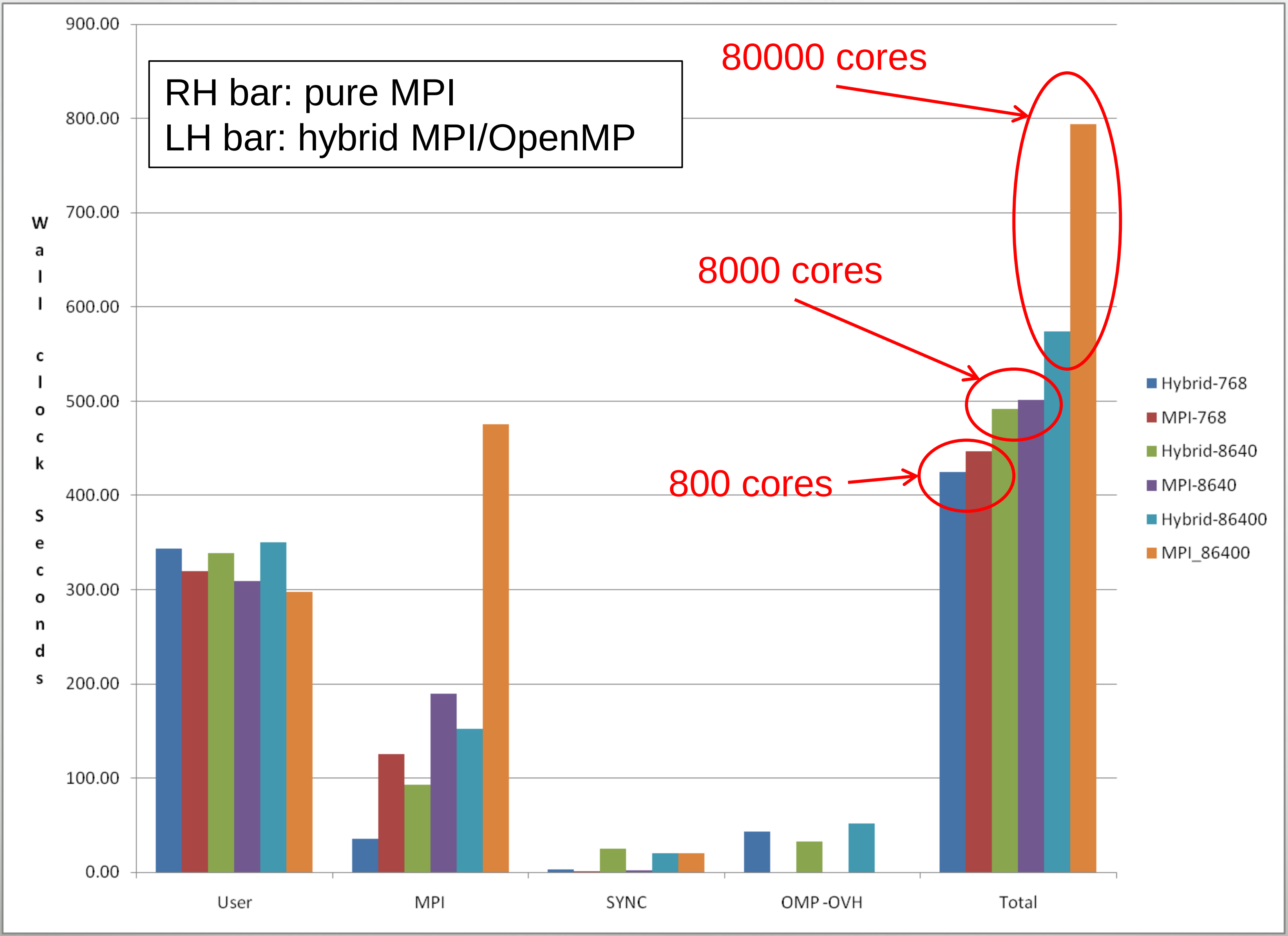
Programming and Applications

John Levesque

with Gene Wagenbreth

CRC Press

Hybrid vigour: S3D performance (CPU)



RH bar: pure MPI
LH bar: hybrid MPI/OpenMP

80000 cores

8000 cores

800 cores

Cray GPU Programming Environment

- Objective: Enhance productivity related to porting applications to hybrid multi-core systems
  - Provide all the information needed for the porting strategy already described here
- Five core components
  - Cray Statistics Gathering Facility on host and GPU
  - Cray Optimization Explorer – Scoping Tools (COE)
  - Cray Compilation Environment (CCE)
  - Cray GPU runtime library
  - Cray GPU Libraries

In conclusion...

- Cray XK6 integrates GPU accelerators
  - Programming accelerators efficiently is hard
    - whether measured in wallclock or developer time
  - a unified X86/GPU PE allows users to exploit Cray XK6 efficiently (performance and productivity)
- OpenMP accelerator directives are a key part of this
  - Attractive, familiar programming model
  - Open standard for vendor portability
  - Use original Fortran, C, C++ source code
  - Performance penalty is small

- An integrated porting strategy for hybrid multicore systems benefits both CPU and GPU performance
  - Users need new tools to achieve this
  - Cray unifed PE supplies such tools

Acknowledgments

- John Levesque and Roberto Ansaloni (Cray)
- Cray R&D: Luiz DeRose, Suzanne LaCroix, David Oehmke, James Beyer, Vince Graziano...
- EPCC Exascale team
- ORNL team
- OpenMP subcommittee

For further info, ahart@cray.com or see our paper this week at IWOMP11...

IWOMP 2011 international workshop on OpenMP

JAGUAR
APPLIED MECHANICS
Railway Technical Research Institute
HLRS
CRAY XT5
KRAKEN

Thank You!