Developing performance portable many-core codes

Simon McIntosh-Smith

UOB HPC

intel

An Intel Parallel Computing Center

University of BRISTOL

Twitter: @simonmcs

Simon McIntosh-Smith

- Graduated as Valedictorian in Computer Science from Cardiff University in 1991
- Joined Inmos to work for David May as a microprocessor architect
- Moved to Pixelfusion in 1999 – a high-tech start-up designing the first GPGPU, a many-core general purpose graphics processor
- Co-founded ClearSpeed in 2002 as Director of Architecture and Applications
- Joined the CS department at the University of Bristol in April 2009 to focus on High Performance Computing. Member of OpenCL standards body, Khronos, involved in running the UK's national HPC service, …

Bristol HPC industrial collaborations

- AWE / Sandia:
  - Co-developers of the Mantevo benchmark suite of mini-apps (http://mantevo.org)

AWE

Sandia National Laboratories

OAK RIDGE National Laboratory

- Los Alamos National Laboratory:
  - Jointly created OpenCL annual conference, IWOCL (http://iwocl.org)

Los Alamos NATIONAL LABORATORY

IWOCL 2014 INTERNATIONAL WORKSHOP ON OPENCL

OFFICIAL DATE & VENUE:
12-13 MAY, 2014
BRISTOL UNIVERSITY, ENGLAND

University of BRISTOL

UoB a partner in global HPC projects

Exascale:

- Mont Blanc (ARM+GPU for HPC)

National HPC services:

- Archer (Cray XC30, >100,000 cores)
- HECToR

European HPC projects:

- PRACE
- European Exascale Software Initiative (EESI)

University of BRISTOL

UoB contributing to HPC standards

Major contributor to OpenCL and its new Standard Portable Intermediate Representation (SPIR)

OpenCL

University of BRISTOL

Motivation

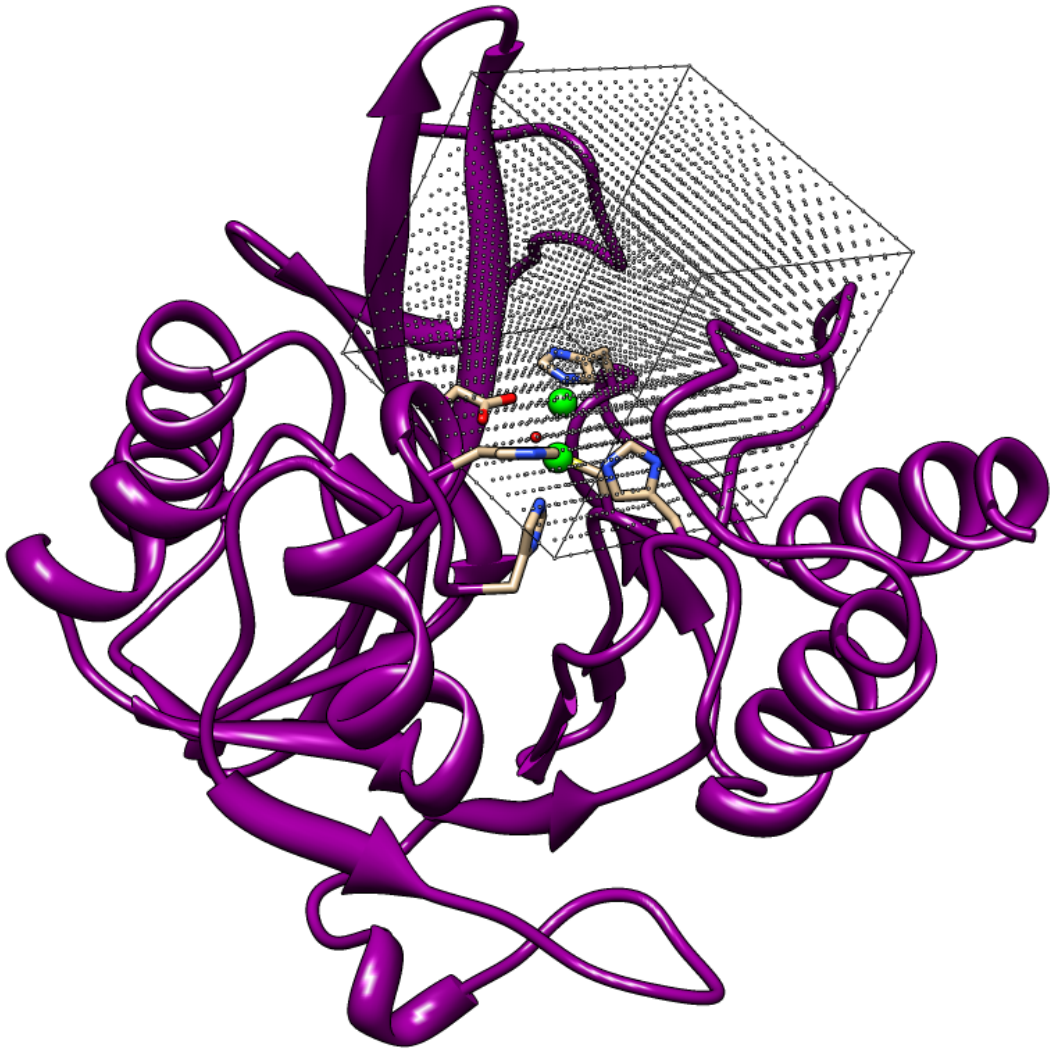
- All HPC processors going many-core / data parallel (Intel Xeon Phi, Nvidia, AMD)
- Initially very divergent parallel programming languages / APIs required:
  - Vector intrinsics
  - Cn (ClearSpeed)
  - Brook / Stream
  - CUDA
  - OpenMP / OpenACC
  - MPI
- This is a nightmare situation for software developers!
- Does it have to be this way?

BUDE – molecular docking

What is BUDE?

- Dr Richard Sessions, PI (Biochemistry)
- **B**ristol **U**niversity **D**ocking **E**ngine
- Designed for true *in silico* virtual drug screening by docking
- Employs a genetic algorithm-based search of the six degrees of freedom in the arrangement of the protein and drug molecules to reduce the search space
- Uses a tuned empirical free-energy forcefield for predicting the binding pose and energy of the ligand with the target protein

BUDE protein-ligand docking



University of BRISTOL

What did we do?

- Started with OpenCL
  - Supported by all the major vendors (even Nvidia!)
- Optimised initially for the most parallel device we had
- Kept checking that the optimisations weren't making things worse on the other devices

More specifically…

- Ported all the code to the accelerator
- Helped the compiler turn all the conditional branches into straight-line, predicated code
  - Involved eyeballing the generated PTX
- Did all the usual things to optimise memory accesses
  - Alignment, padding, coalescence etc.
- Chose sensible problem/work-group sizes

Optimising conditional branches

Conditional execution

```
// Only evaluate expression
// if condition is met
if (a > b)
{
  acc += (a - b*c);
}
```

Corresponding PTX

```
setp.gt.f32 %pred, %a, %b
@!%pred bra $endif
mul.f32 %f0, %b, %c
sub.f32 %f1, %a, %f0
add.f32 %acc, %acc, %f1
$endif:
```

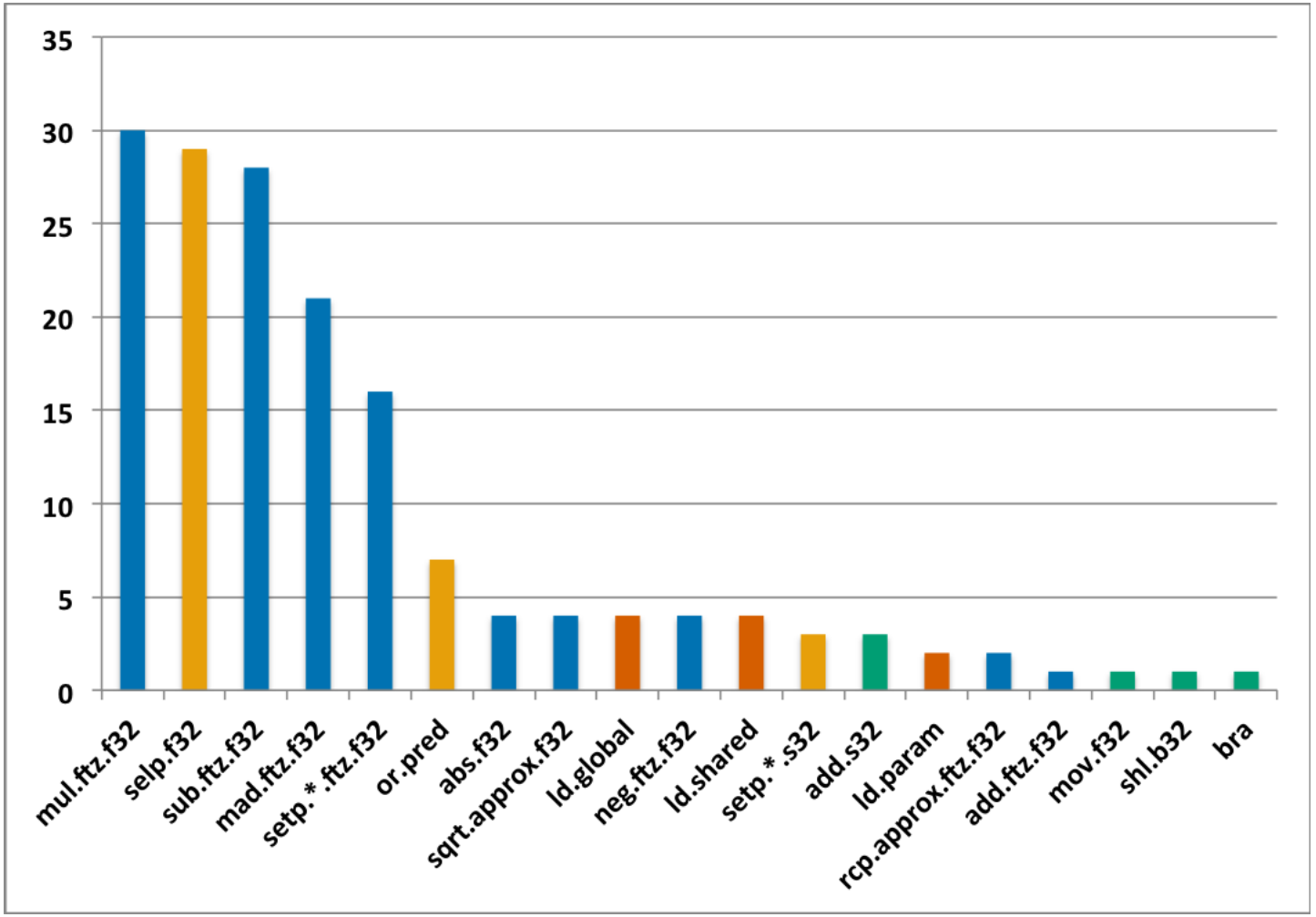
Selection and masking

```
// Always evaluate expression
// and mask result
temp = (a - b*c);
mask = (a > b ? 1.f : 0.f);
acc += (mask * temp);
```

Corresponding PTX

```
mul.f32 %f0, %b, %c
sub.f32 %temp, %a, %f0
setp.gt.f32 %pred, %a, %b
selp.f32 %mask, %one, %zero, %pred
mad.f32 %acc, %mask, %temp, %acc
```

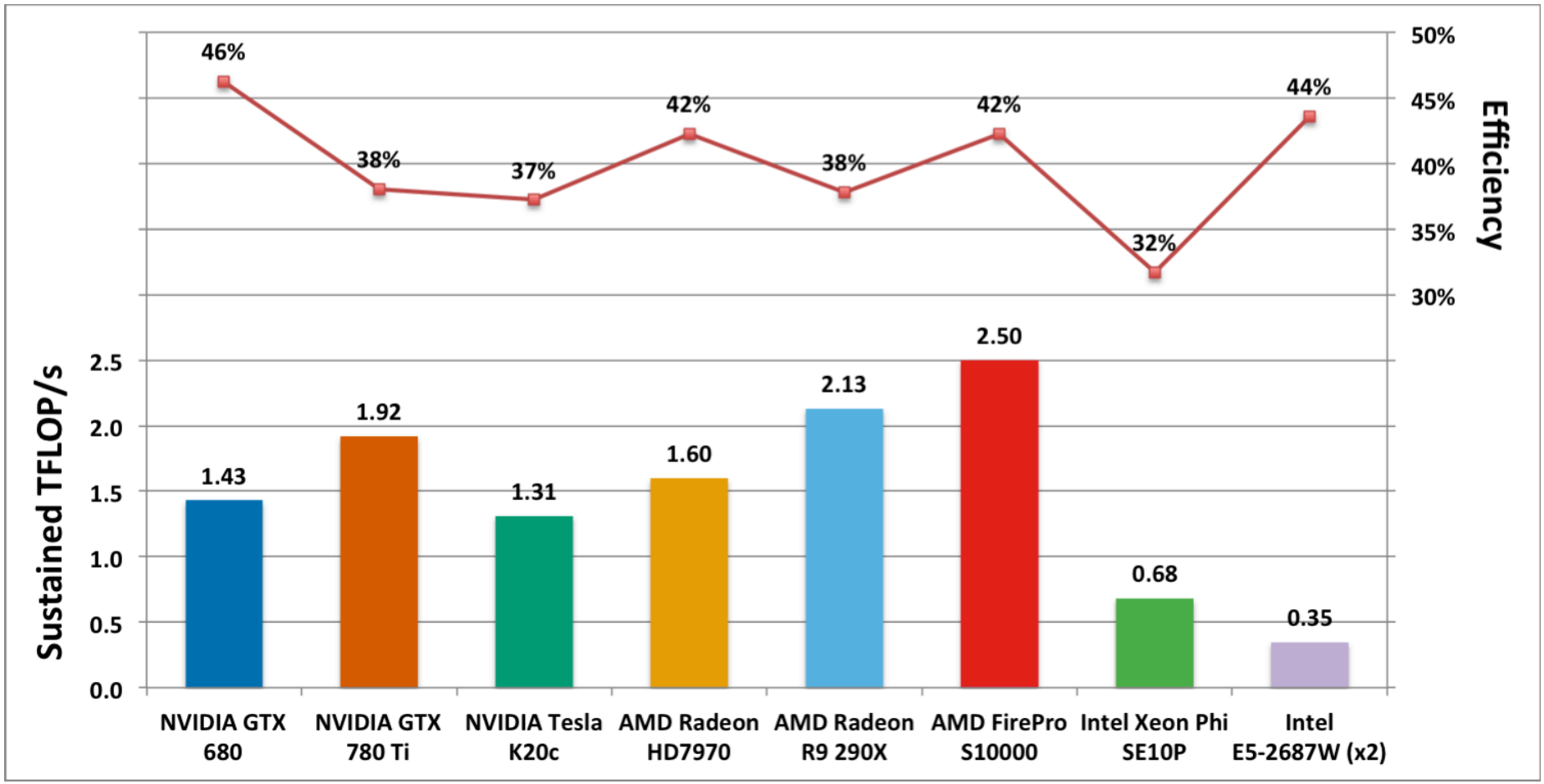
Instruction mix



Target hardware

Platform	Clock (GHz)	RAM (GB)	Memory B/W (GB/s)	S.P. TFLOP/s	D.P. TFLOP/s	TDP (W)
AMD FirePro S10000	0.825	6	480	5.91	1.48	375
AMD Radeon HD 7970	0.925	3	264	3.78	0.95	230
AMD Radeon R9 290X	1.000	4	320	5.63	0.70	250
Intel Xeon E5-2687W (x2)	3.100	32	102	0.79	0.40	300
Intel Xeon Phi SE10P	1.100	8	320	2.15	1.07	300
NVIDIA GTX 780 Ti	0.928	3	336	5.05	0.21	250
NVIDIA GTX 680	1.006	2	192	3.00	0.13	195
NVIDIA Tesla K20	0.706	6	208	3.52	1.17	225
NVIDIA Tesla M2090	0.650	6	177	1.33	0.66	225

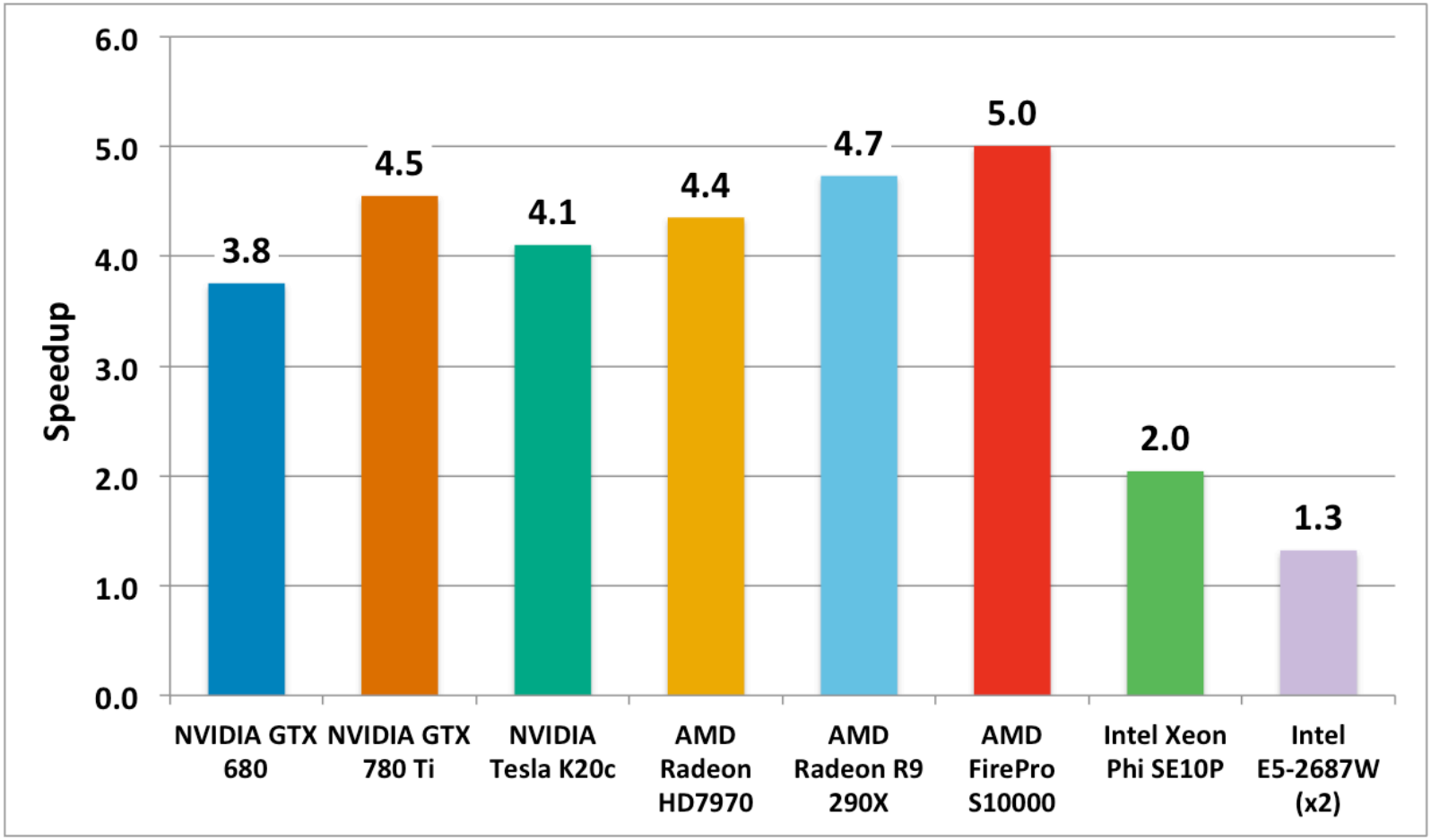
BUDE results



"High Performance *in silico* Virtual Drug Screening on Many-Core Processors", S. McIntosh-Smith, J. Price, R.B. Sessions, A.A. Ibarra, IJHPCA 2014
DOI: 10.1177/1094342014528252

University of BRISTOL

How much the optimisations helped

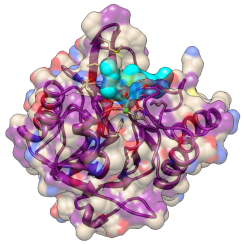


"High Performance *in silico* Virtual Drug Screening on Many-Core Processors", S. McIntosh-Smith, J. Price, R.B. Sessions, A.A. Ibarra, IJHPCA 2014
DOI: 10.1177/1094342014528252

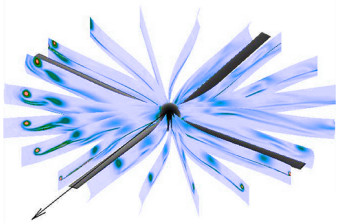
University of BRISTOL

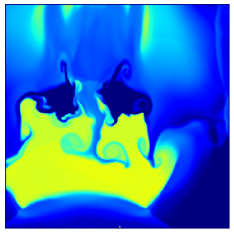
Performance portability

- BUDE was highly performance portable
  - Compute intensive, N-body / Monte Carlo



- Bandwidth intensive codes next
  - Structured grid codes:
    - Lattice Boltzmann
    - CloverLeaf (hydrodynamics)
    - ROTORSIM (CFD)



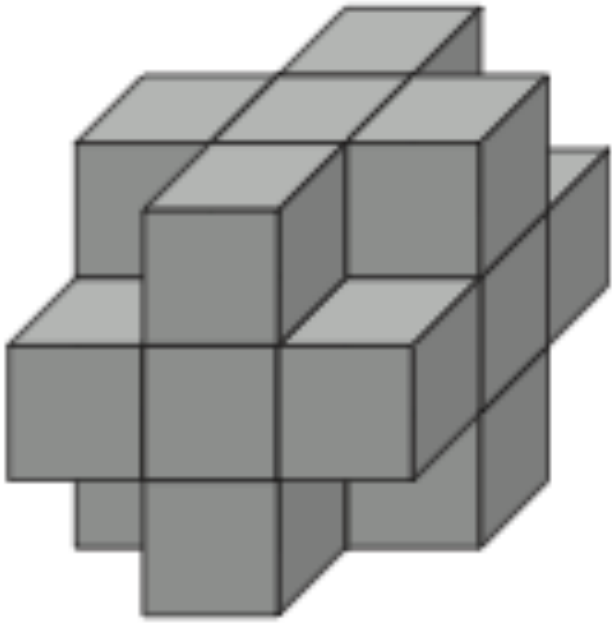


Structured Grid Codes

Lattice Boltzmann (LBM)

- A versatile approach for solving incompressible flows based on a simplified gas-kinetic description of the Boltzmann equation (used for CFD etc)
- Ports well to most parallel architectures
- We targeted one of the most widely used variants, **D3Q19-BGK**

D3Q19-BGK LBM

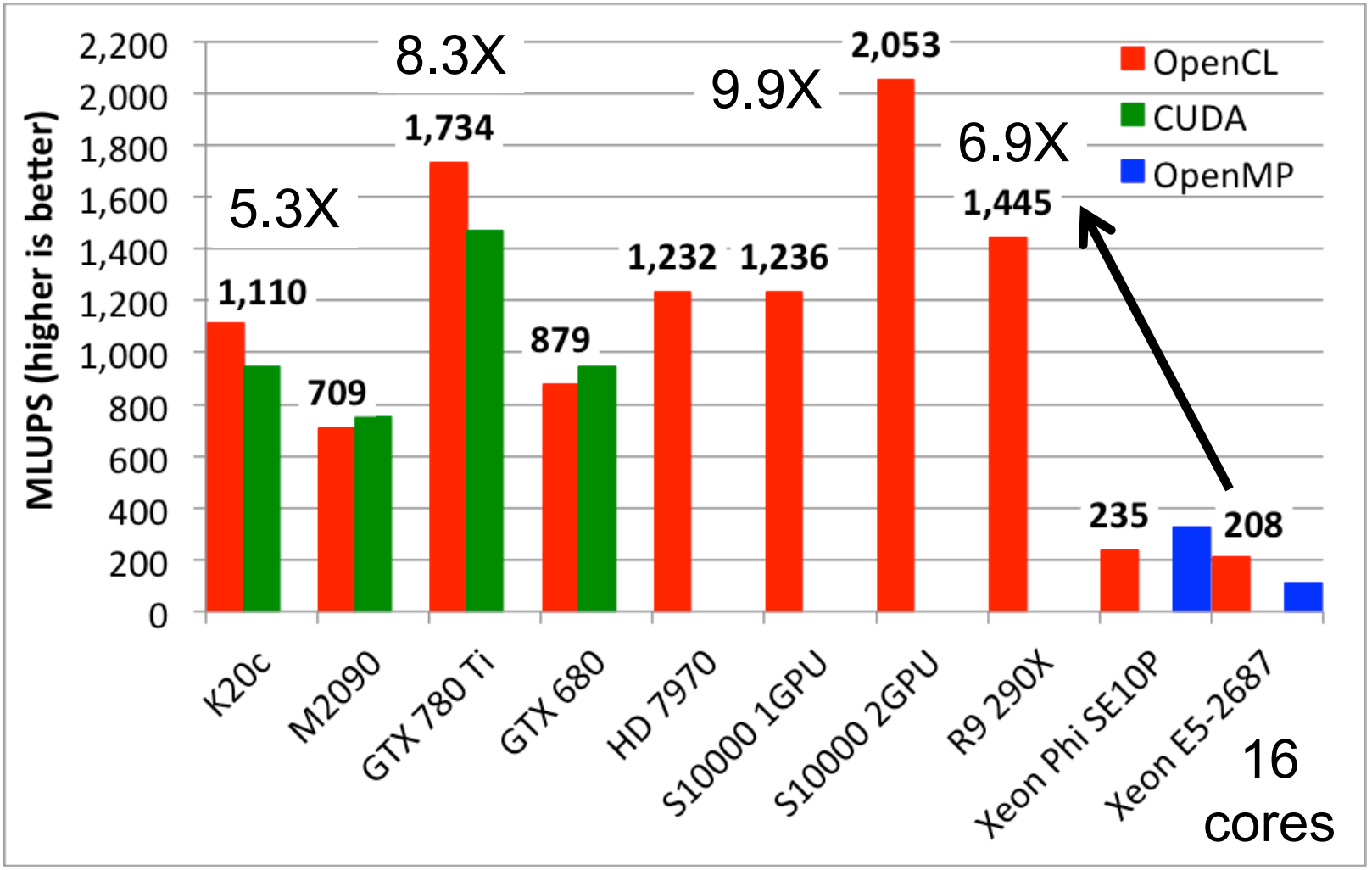


- To update a cell, need to access 19 of the 27 surrounding cell values in the 3D grid

Methodology

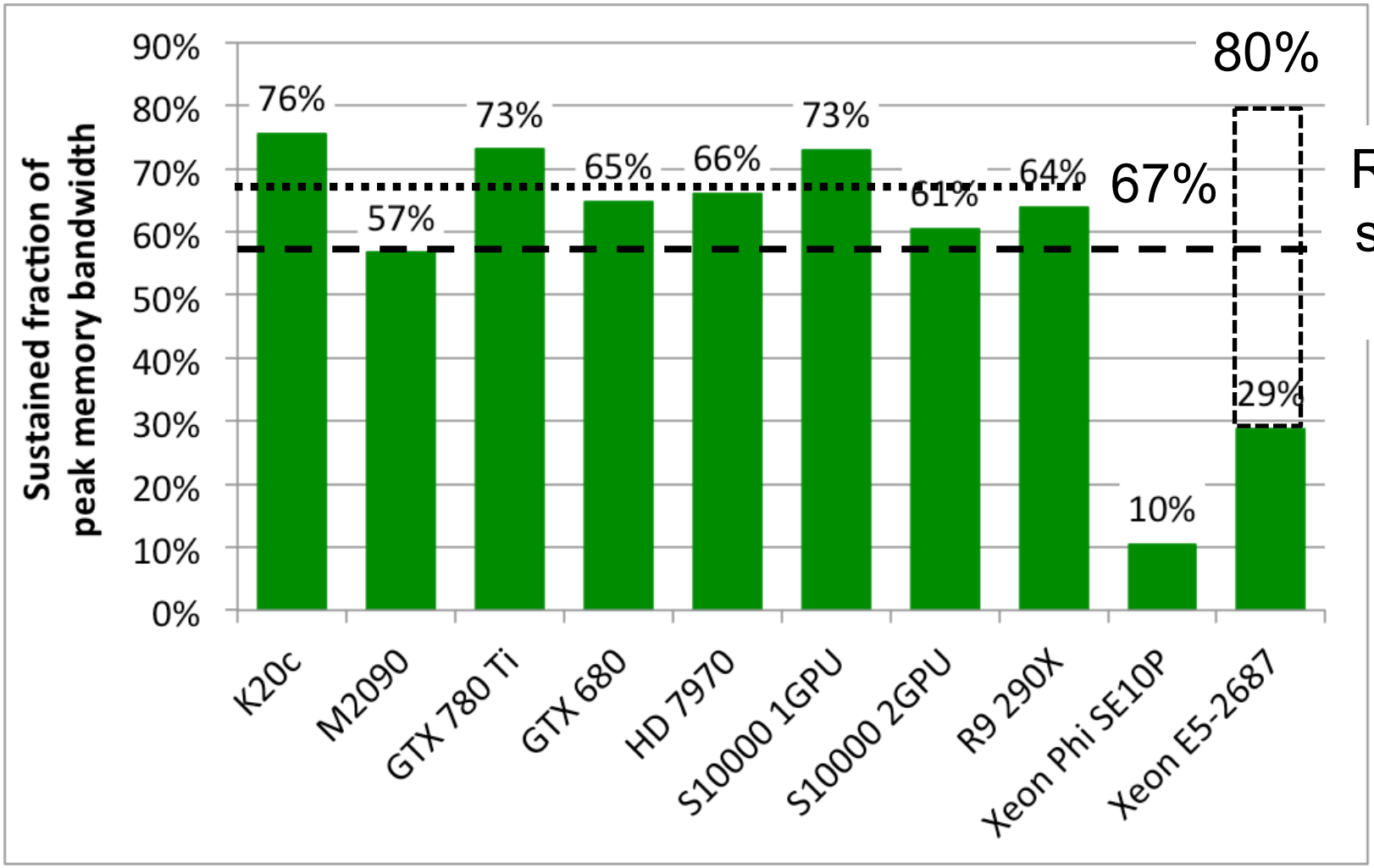
- Developed a code that was efficient but not over complicated
- "Identical" versions in OpenCL and CUDA
  - Single precision grid 128^3 (~2m grid points, 304 MBytes)
  - The OpenCL three dimensional work-group size was fixed at (128,1,1) for ***all*** OpenCL runs on ***all*** devices
  - Same arrangement for CUDA version
- The OpenMP code was as close as possible to the OpenCL/ CUDA versions
- Ensured the OpenMP code was being vectorised by the compiler (latest Intel icc)

Performance results for 128^3



Single precision results

Performance results for 128^3

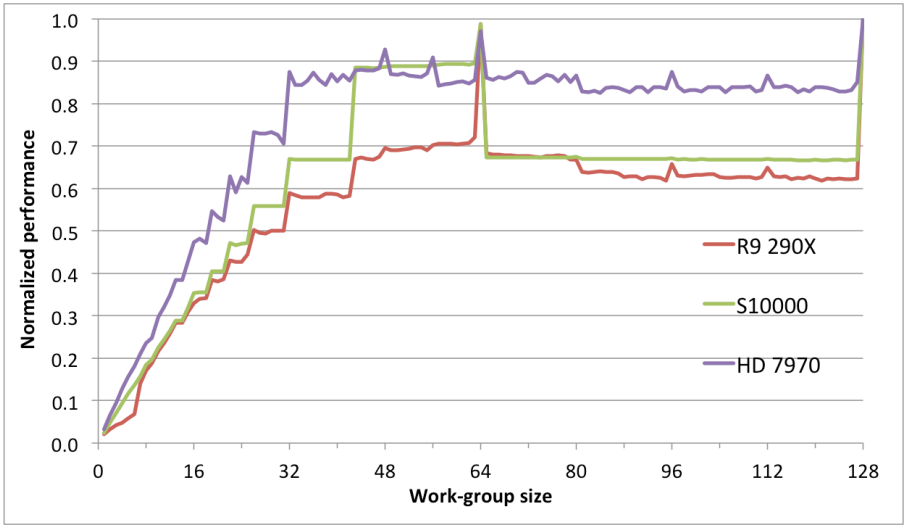


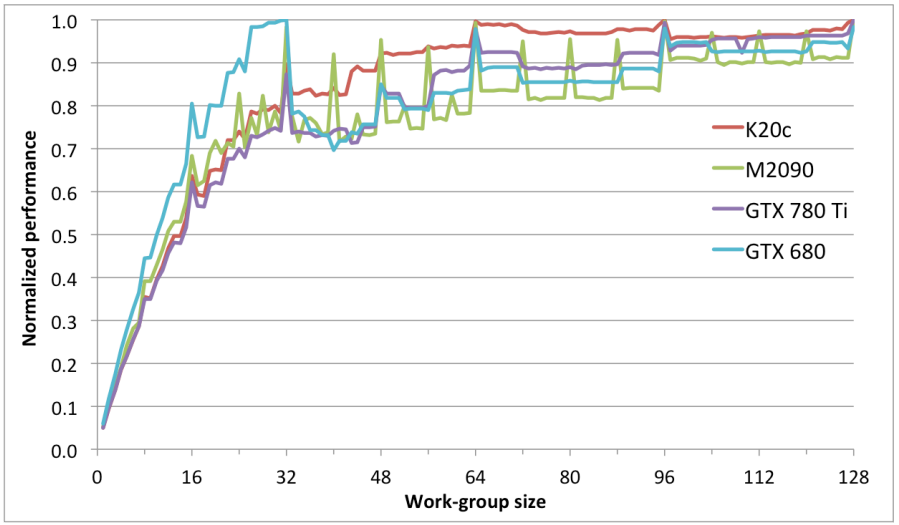
OpenCL single precision results

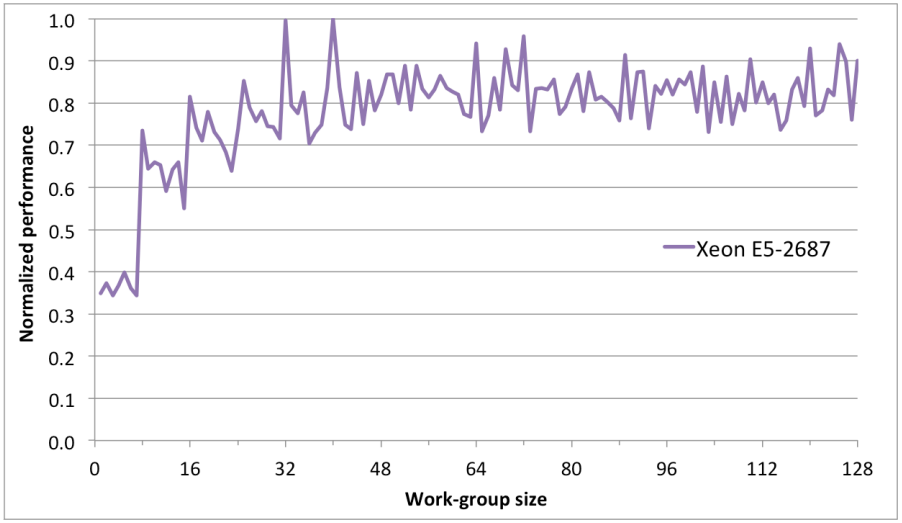
So perf. portable, but is it fast?

- On an Nvidia K20, our best 128^3 single precision performance in OpenCL was **1,110** MLUPS
- In the literature, the fastest quoted results are **~1,000** MLUPS (Januszewski and Kostur's *Sailfish* program) and **982** MLUPS (Mawson and Revell)
- Our results are a **13%** improvement over Mawson-Revell and a **10%** improvement over Januszewski-Kostur

Impact of work-group sizes







University of BRISTOL

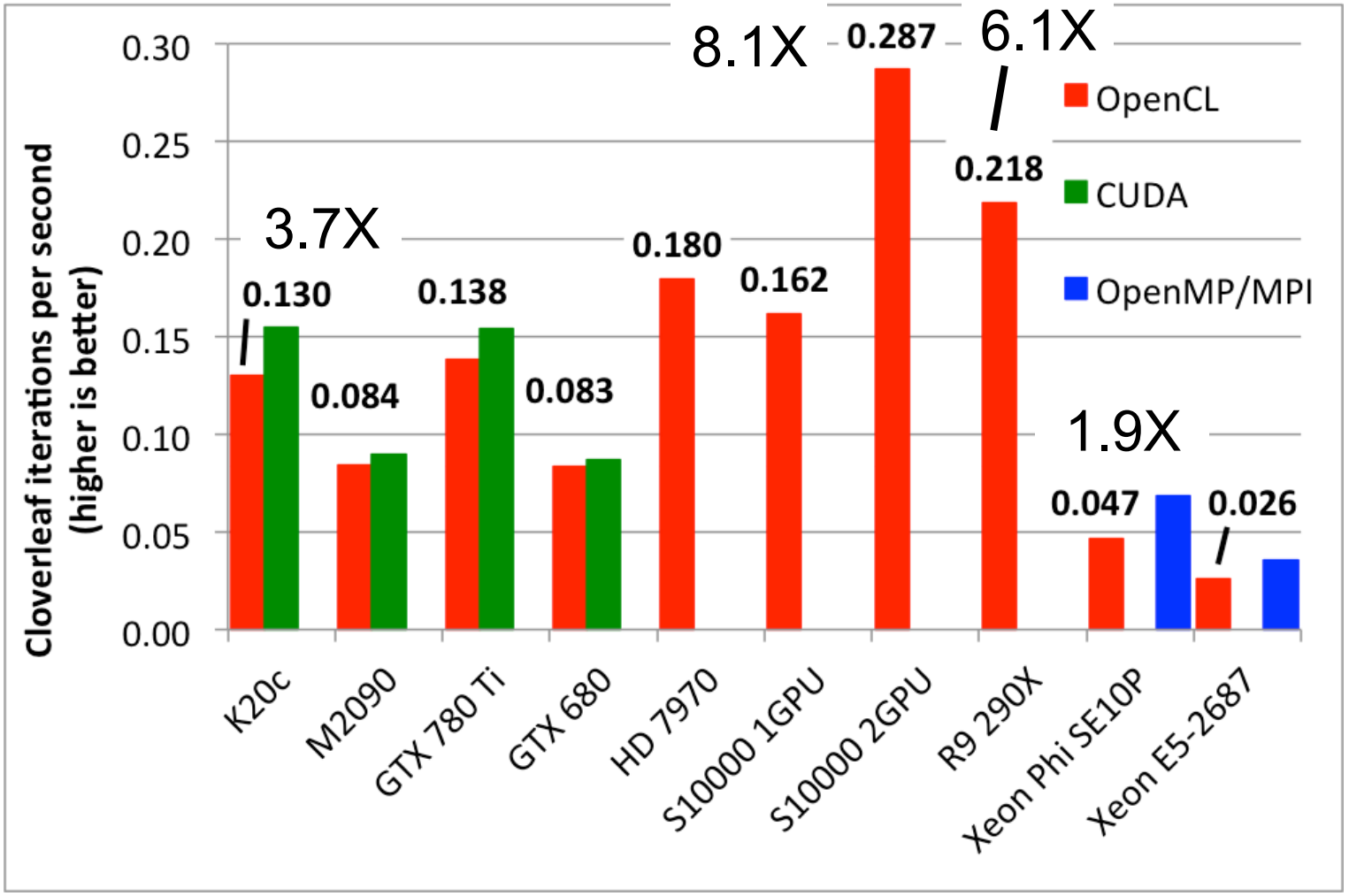
CloverLeaf: A Lagrangian- Eulerian hydrodynamics benchmark

- A collaboration between AWE, Warwick & Bristol
- CloverLeaf is a bandwidth-limited, structured grid code and part of Sandia's "Mantevo" benchmarks.
- Solves the compressible Euler equations, which describe the conservation of energy, mass and momentum in a system.
- These equations are solved on a Cartesian grid in 2D with second-order accuracy, using an explicit finite-volume method.
- Optimised parallel versions exist in OpenMP, MPI, OpenCL, OpenACC, CUDA and Co-Array Fortran.

CloverLeaf benchmark parameters

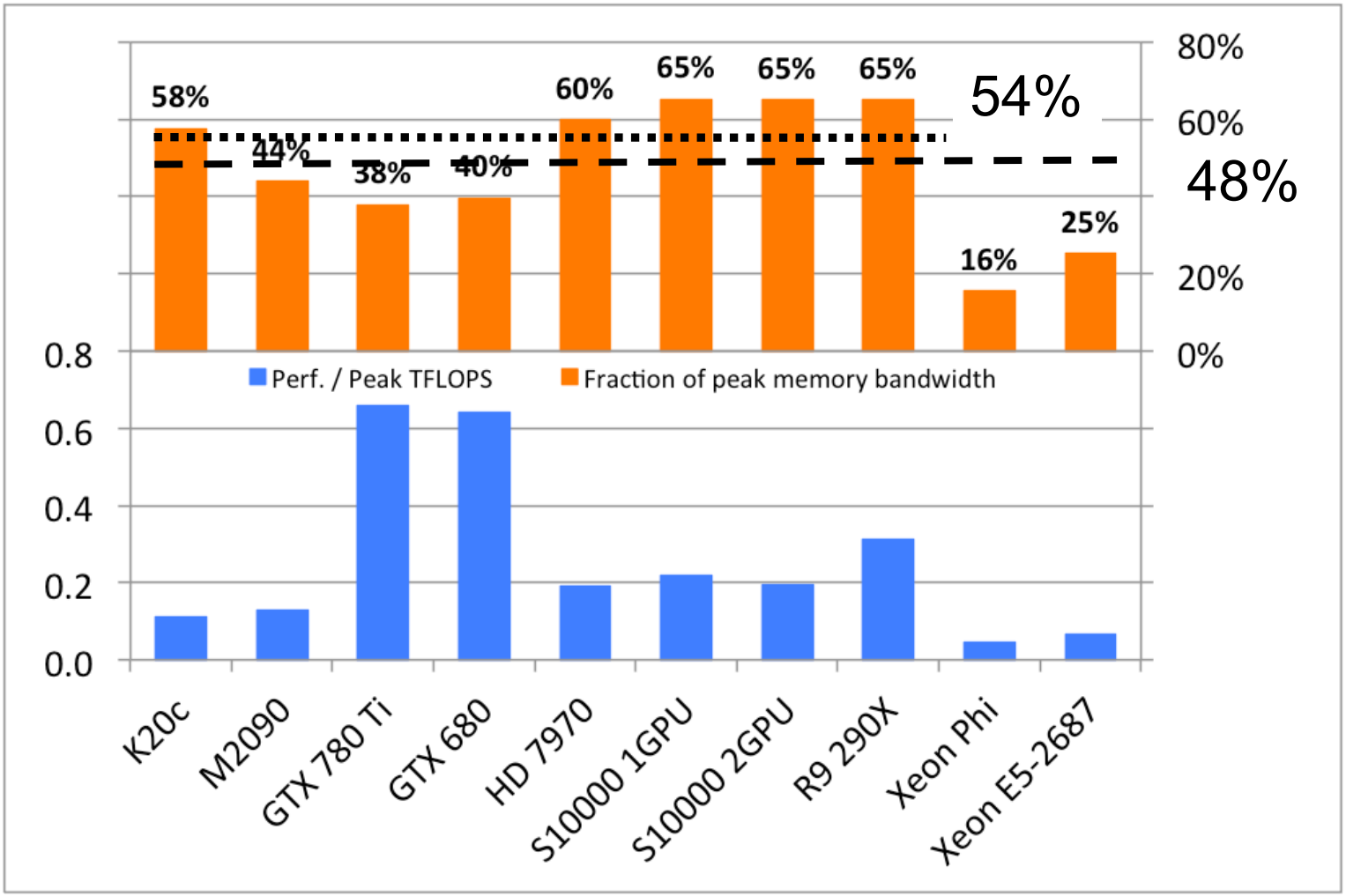
- Double precision grid of size 1920×3840
  - ~7.4m grid points, 25 values per grid point →~1.5 Gbytes in working dataset
- The OpenCL and CUDA parallelisations were performed in an identical manner
  - One work-item/thread for each grid point
  - Identical arrangements for work-group sizes and layouts
  - E.g. 2D work-groups of (128, 1) for OpenCL

Results – performance



S.N. McIntosh-Smith, M. Boulton, D. Curran, & J.R. Price, "On the performance portability of structured grid codes on many-core computer architectures", ISC, Leipzig, June 2014. DOI: 10.1007/978-3-319-07518-1_4

Results – sustained bandwidth



University of BRISTOL

S.N. McIntosh-Smith, M. Boulton, D. Curran, & J.R. Price, "On the performance portability of structured grid codes on many-core computer architectures", ISC, Leipzig, June 2014. DOI: 10.1007/978-3-319-07518-1_4

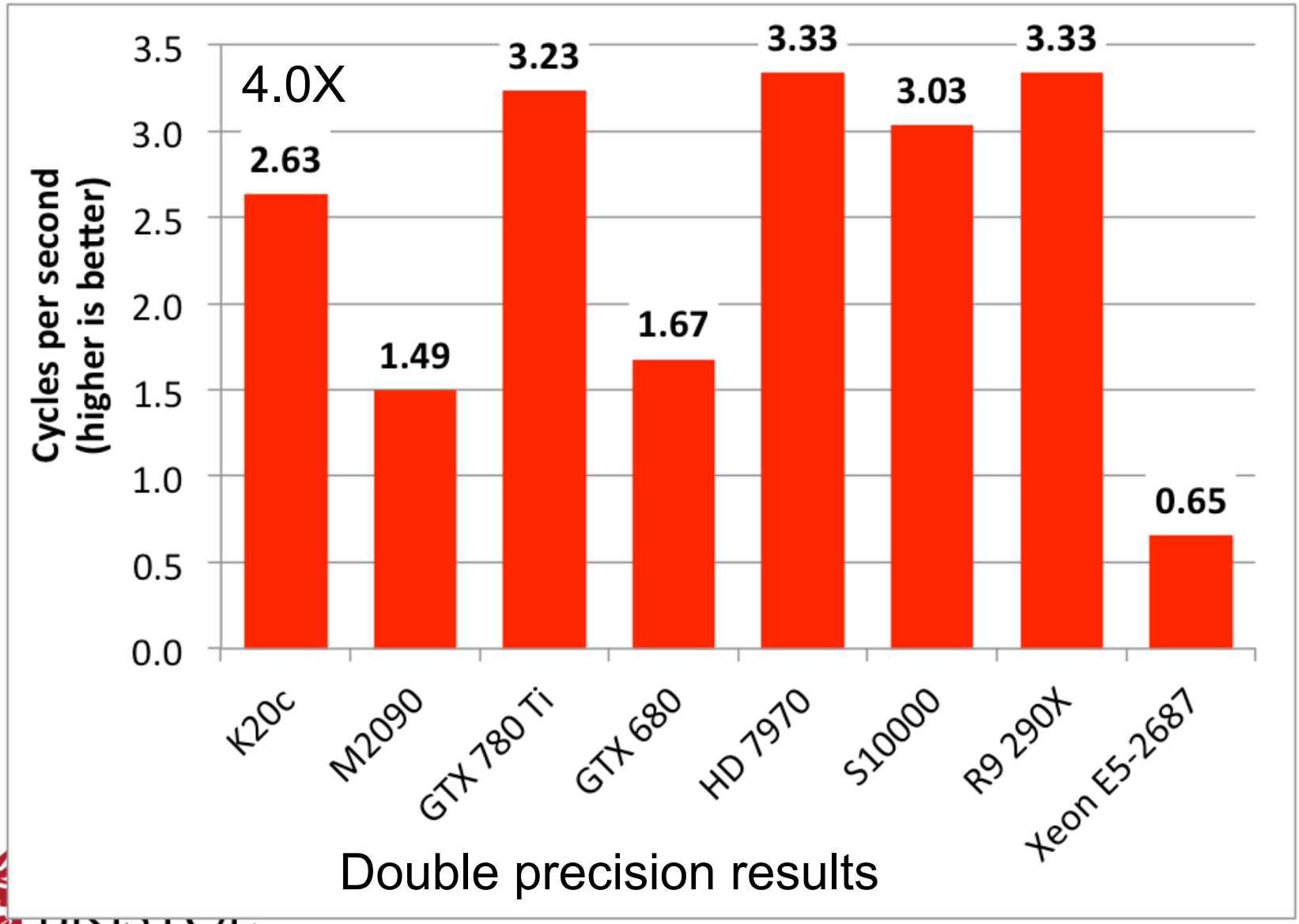
ROTORSIM

MONT-BLANC

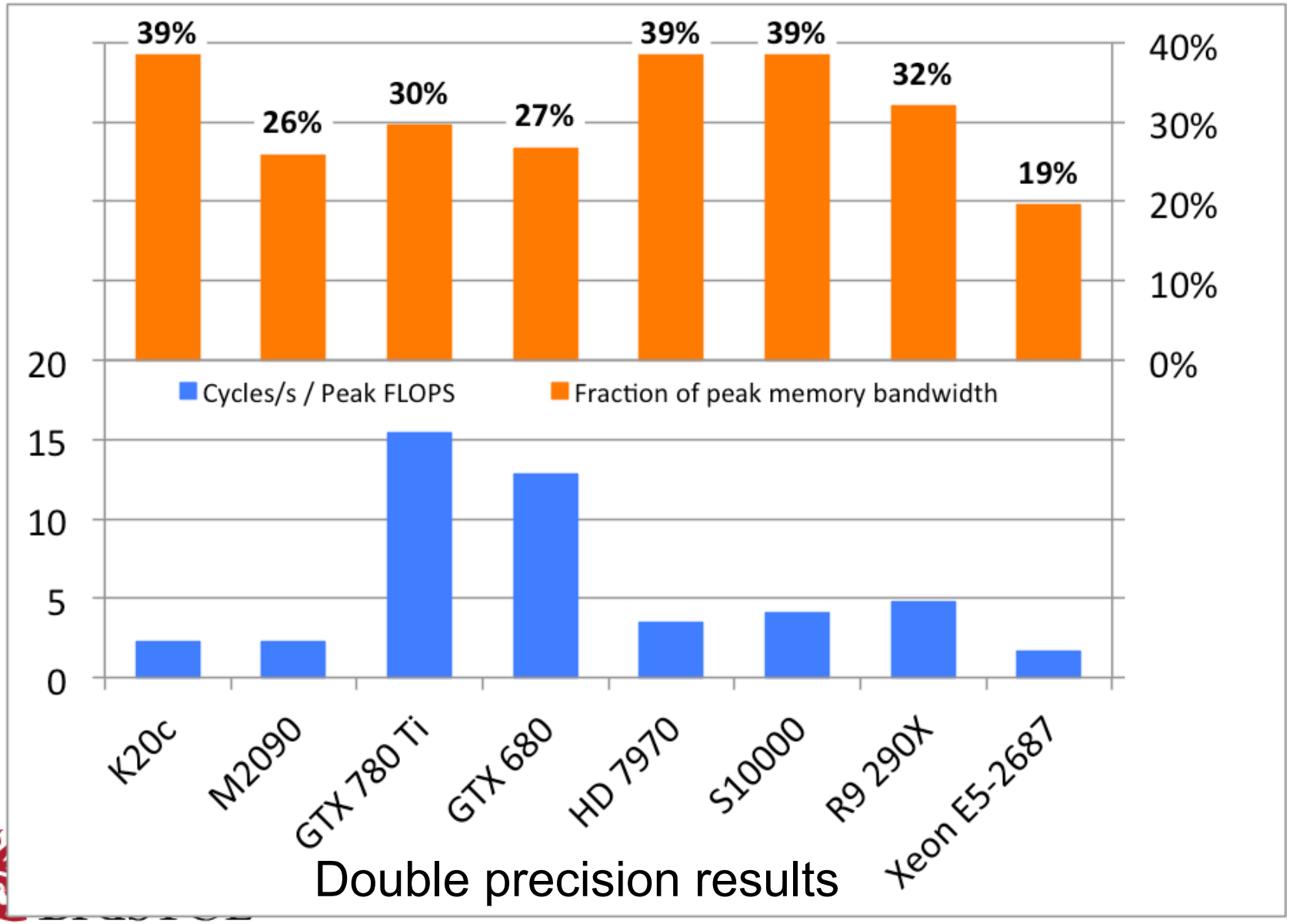
- A production multiblock, compressible finite-volume CFD code
- Developed in Bristol by Prof. Chris Allen
- Upwind, third-order accurate spatial stencil, with an explicit time integration scheme for steady flows
- Implicit dual-time approach for time-accurate calculations
- Optimised versions in MPI and OpenCL

University of BRISTOL

Results – performance



Results – sustained bandwidth



Double precision results

Performance portability isn't what we expect

But why not?

Why don't we expect perf. portability?

- Historical reasons
  - Started with immature drivers
  - Started with immature architectures
  - Started with immature applications

- But things have ***changed***
  - Drivers now mature / maturing
  - Architectures now mature / maturing
  - Applications now mature / maturing

Is anyone else achieving performance portability?

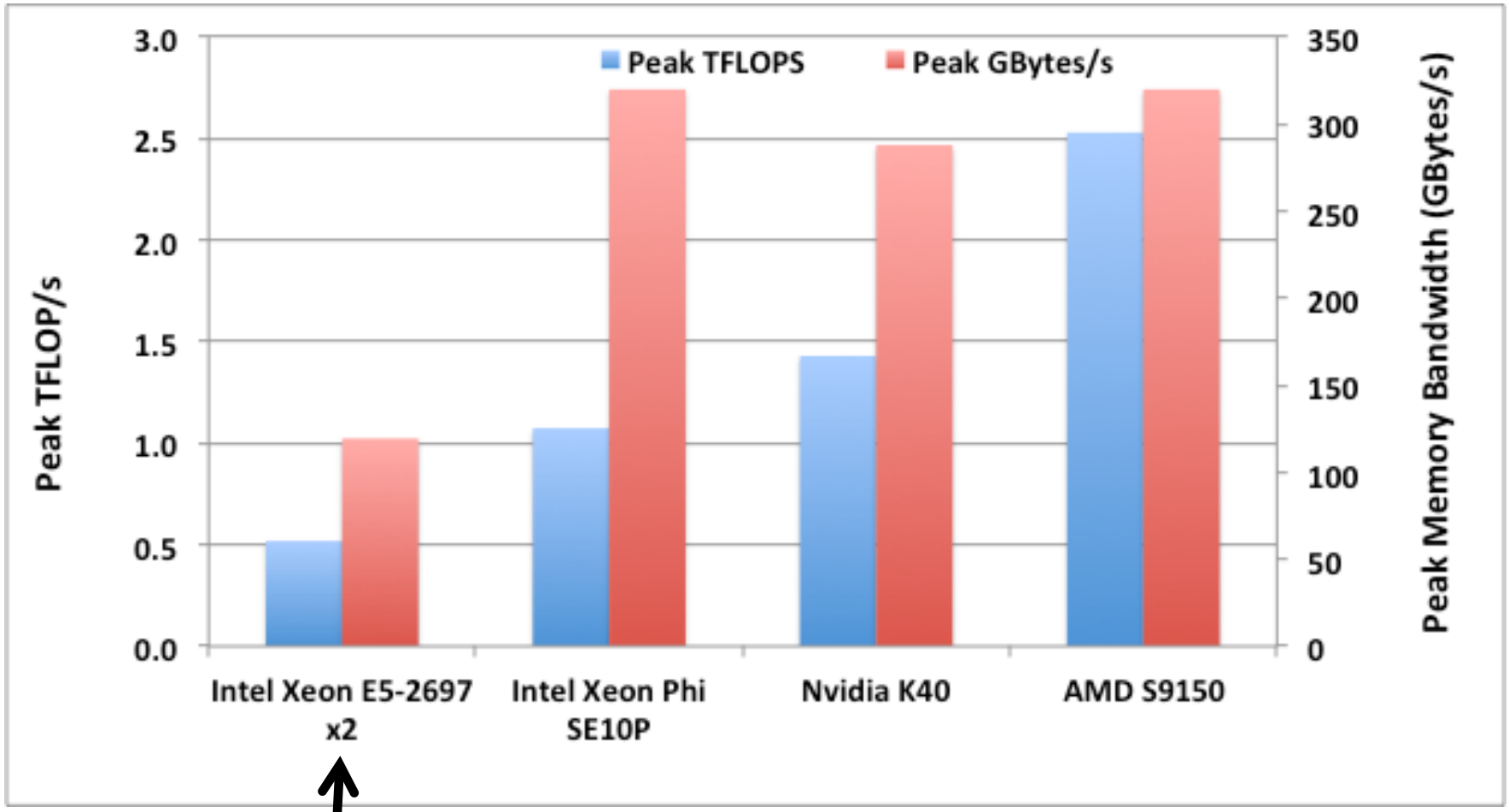
ViennaCL
http://viennacl.sourceforge.net/

- Some very interesting data from Karl Rupp's ViennaCL
- Historical study into performance portability of BLAS functions (L1 to L3)
- His results show that performance was much more "peaky" on older devices and older drivers
- Much smoother today – and indeed possible to tune on one device to represent many
- To appear in "Performance Portability Study of Linear Algebra Kernels in OpenCL", Karl Rupp et al., International Workshop on OpenCL (IWOCL), May 12-13th 2014, Bristol, UK. ACM ICPS, ISBN 978-1-4503-3007-7

Performance portability techniques

- Use a platform portable parallel language
- Aim for 80-90% of optimal
- Avoid platform-specific optimisations
- ***Most*** optimisations make the code faster on ***most*** platforms

Other motivations…



16 cores of Sandy Bridge at 3.1 GHz

University of BRISTOL

Conclusions

- We have demonstrated that it's possible to achieve good performance portability for two classes of application so far:
  - **N-body / Monte Carlo codes** – *BUDE*
  - **Structured grid codes** - *lattice Boltzmann, CloverLeaf* and *ROTORSIM*
- **OpenCL** can straightforwardly enable a much better degree of performance portability than you might expect

University of BRISTOL

Google: **HandsOnOpenCL** on *Github*

Related Publications

[1] "High Performance *in silico* Virtual Drug Screening on Many-Core Processors", S. McIntosh-Smith, J. Price, R.B. Sessions, A.A. Ibarra, IJHPCA 2014. DOI: 10.1177/1094342014528252

[2] "On the performance portability of structured grid codes on many-core computer architectures", S.N. McIntosh-Smith, M. Boulton, D. Curran and J.R. Price. ISC, Leipzig, June 2014. DOI: 10.1007/978-3-319-07518-1_4

[3] "Evaluation of a performance portable lattice Boltzmann code using OpenCL", S.N. McIntosh-Smith and D. Curran. International Workshop on OpenCL (IWOCL), May 12-13th 2014, Bristol, UK. ACM ICPS, ISBN 978-1-4503-3007-7

[4] "Accelerating hydrocodes with OpenACC, OpenCL and CUDA", Herdman, J., Gaudin, W., McIntosh-Smith, S., Boulton, M., Beckingsale, D., Mallinson, A., Jarvis, S. In: High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:. (Nov 2012) 465-471. DOI: 10.1109/SC.Companion.2012.66

Backup

"Isn't OpenCL much harder than CUDA?"

- Nope
- Kernels are almost identical
- Latest OpenCL C++ API does the same for the host code
- OpenCL is being very strongly supported by Intel and AMD (and ARM, IMG, …)
- Nvidia's OpenCL implementation is surprisingly good, though they hate to admit it!

Enqueue a kernel (C++)

CUDA C

```
dim3
threads_per_block(30,20);

dim3 num_blocks(10,10);


kernel<<<num_blocks,
  threads_per_block>>>(…);
```

OpenCL C++

```
const cl::NDRange
      global(300, 200);

const cl::NDRange
      local(30, 20);

kernel(
  EnqueueArgs(global, local),
  …);
```

University of BRISTOL

Allocating and copying memory

	CUDA C	OpenCL C++

Allocate	`float* d_x;`
	`cudaMalloc(&d_x,`
	`  sizeof(float)*size);`
		`cl::Buffer`
		`  d_x(begin(h_x), end(h_x), true);`

Host to Device	`cudaMemcpy(d_x, h_x,`
	`  sizeof(float)*size,`
	`  cudaMemcpyHostToDevice);`
		`cl::copy(begin(h_x), end(h_x),`
		`  d_x);`

Device to Host	`cudaMemcpy(h_x, d_x,`
	`  sizeof(float)*size,`
	`  cudaMemcpyDeviceToHost);`
		`cl::copy(d_x,`
		`  begin(h_x), end(h_x));`