Evaluating OpenMP’s Effectiveness in the Many-Core Era

Prof Simon McIntosh-Smith
HPC Research Group
simonm@cs.bris.ac.uk
Bristol, UK

- 10th largest city in UK
- Aero, finance, chip design
- HQ for Cray EMEA
- 100 miles west of London
Bristol's long HPC history
What does my group do?

- **Performance portability**
  - Programming model evaluations
  - Code design strategies
  - Hardware evaluations
  - "Cross-X", where X = vendor, language, …

- **Fault tolerance**
  - Application-based fault tolerance
  - Reliable computing on unreliable hardware
PERFORMANCE
PORTABILITY
BUDE – MOLECULAR DOCKING (2013)
What is BUDE?

• **Bristol University Docking Engine**
  • Dr Richard Sessions, PI (Biochemistry)
• *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
BUDE protein-ligand docking
## Target hardware

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

- **BUDE**'s OpenCL implementation proved to be highly performance portable
  - Compute intensive, N-body / Monte Carlo

- Looked at bandwidth intensive codes next, such as the **CloverLeaf** structured grid hydrodynamics mini-app
CloverLeaf: A Lagrangian-Eulerian hydrodynamics benchmark

- 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.
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
(Ninja level) 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

• This was only possible in OpenCL in 2014…
HIGHER-LEVEL PERFORMANCE PORTABILITY (2014-)
Moving on up

• Low-level programming in OpenCL or CUDA is all very well ...

• ... But we don't expect most scientific codes to be re-written in these languages

• What are the emerging options?
  • Directive-based: OpenMP 4.x, OpenACC, OmpSs, ...
  • C++ based: RAJA, Kokkos, SYCL, ...
Investigating the Performance Portability Capabilities of OpenMP 4, Kokkos and Raja

Using TeaLeaf and other mini-apps to assess how performance portable modern parallel programming models are

Matt Martineau - UoB (m.martineau@bristol.ac.uk)
Simon McIntosh-Smith - UoB (cssnmis@bristol.ac.uk)
Wayne Gaudin – UK Atomic Weapons Establishment

DOE performance portability workshop, Arizona, April 2016.
TeaLeaf – Heat Conduction

- Mini-app from Mantevo suite of benchmarks

- Implicit, sparse, matrix-free solvers on structured grid
  - Conjugate Gradient (CG)
  - Chebyshev
  - Preconditioned Polynomial CG (PPCG)

- Memory bandwidth bound
- Good strong and weak scaling on Titan & Piz Daint
The Performance Experiment

• Performance tested on **CPU, GPU, and KNC**

• Single node only (multi-node scaling proven)

• All ports were optimised as much as possible, while ensuring performance portability

• Solved 4096x4096 problem, the point of mesh convergence, for *single iteration*
All the programming models get to performance within 25% of OpenCL / CUDA hand-optimised code
Martineau, M., McIntosh-Smith, S. Gaudin, W., Assessing the Performance Portability of Modern Parallel Programming Models using TeaLeaf, 2016, CC-PE
TeaLeaf conclusions

- RAJA and Kokkos both looking promising
  - For GPU (NVIDIA) and CPU (Intel, IBM)
  - What about other architectures though?
    - AMD GPUs, ARM CPUs, …
- Big question is: who maintains these in the long-term?
- OpenMP 4.x also looking good for GPUs
  - Still lots of Fortran out there
What we did next

• Based on these early successes we decided to do a detailed assessment of OpenMP 4.x compiler implementations

• Started with something simple – a modern port of STREAM to OpenMP 4.x and other parallel programming languages

• Then looked at a range of codes and mini-apps

• See Matt Martineau's talk on Friday for all the details: "Pragmatic Performance Portability with OpenMP 4.x"
OpenMP 4: STREAM Triad

template <class T>
void OMP45Stream<T>::triad()
{
    const T scalar = 0.3;

    unsigned int array_size = this->array_size;
    T *a = this->a;
    T *b = this->b;
    T *c = this->c;

    #pragma omp target teams distribute parallel for simd \
        map(to: a[0:array_size], b[0:array_size], c[0:array_size])
    for (int i = 0; i < array_size; i++)
    {
        a[i] = b[i] + scalar * c[i];
    }
}
GPU-STREAM 2 performance

Fraction of theoretical peak

<table>
<thead>
<tr>
<th></th>
<th>K20X</th>
<th>K40</th>
<th>K80</th>
<th>GTX 980 Ti</th>
<th>S9150</th>
<th>Fury X</th>
<th>Sandy Bridge</th>
<th>Ivy Bridge</th>
<th>Haswell</th>
<th>Broadwell</th>
<th>KNL</th>
<th>Power 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>McCalpin</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>63%</td>
<td>69%</td>
<td>85%</td>
<td>81%</td>
<td>88%</td>
<td>64%</td>
</tr>
<tr>
<td>SYCL</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>84%</td>
<td>43%</td>
<td>X</td>
<td>X</td>
<td>41%</td>
<td>N/A</td>
</tr>
<tr>
<td>RAJA</td>
<td>-73%</td>
<td>67%</td>
<td>74%</td>
<td>80%</td>
<td>N/A</td>
<td>N/A</td>
<td>53%</td>
<td>54%</td>
<td>67%</td>
<td>64%</td>
<td>58%</td>
<td>73%</td>
</tr>
<tr>
<td>Kokkos</td>
<td>-73%</td>
<td>67%</td>
<td>75%</td>
<td>80%</td>
<td>N/A</td>
<td>N/A</td>
<td>53%</td>
<td>53%</td>
<td>65%</td>
<td>63%</td>
<td>57%</td>
<td>78%</td>
</tr>
<tr>
<td>OpenMP</td>
<td>-70%</td>
<td>63%</td>
<td>70%</td>
<td>71%</td>
<td>N/A</td>
<td>N/A</td>
<td>52%</td>
<td>52%</td>
<td>67%</td>
<td>64%</td>
<td>58%</td>
<td>55%</td>
</tr>
<tr>
<td>OpenACC</td>
<td>-70%</td>
<td>63%</td>
<td>71%</td>
<td>79%</td>
<td>83%</td>
<td>N/A</td>
<td>27%</td>
<td>70%</td>
<td>40%</td>
<td>30%</td>
<td>47%</td>
<td>X</td>
</tr>
<tr>
<td>CUDA</td>
<td>-72%</td>
<td>66%</td>
<td>73%</td>
<td>80%</td>
<td>N/A</td>
<td>N/A</td>
<td>23%</td>
<td>54%</td>
<td>31%</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>OpenCL</td>
<td>-73%</td>
<td>66%</td>
<td>76%</td>
<td>80%</td>
<td>84%</td>
<td>86%</td>
<td>29%</td>
<td>45%</td>
<td>32%</td>
<td>32%</td>
<td>46%</td>
<td>N/A</td>
</tr>
</tbody>
</table>

On those supported target architectures, OpenMP 3.0 achieves good performance.

OpenMP 4.x performance

Deakin, T., Price, J., Martineau, M., McIntosh-Smith, S., GPU-STREAM v2.0 Benchmarking the achievable memory bandwidth of many-core processors across diverse parallel programming models, P³MA, ISC’16

Third party names are the property of their owners.
Current Compiler Support

- **Intel** began support for OpenMP 4.0 targeting the Xeon Phi coprocessors in 2013.

- **Cray** provided the first vendor implementation targeting NVIDIA GPUs in late 2015. Now supports OpenMP 4.0, and a subset of OpenMP 4.5.

- **IBM** has recently completed a compiler implementation using Clang, that fully supports OpenMP 4.5. This is being introduced into the Clang main trunk.

- **GCC 6.1** introduced full support for OpenMP 4.5, and can target Intel Xeon Phi, or HSA enabled AMD GPUs. Still very immature.
Performance?

• To test performance we use a mixture of synthetic benchmarks and mini-apps.

• We compare against device-specific code written in OpenMP 3.0 and CUDA.

• We use OpenMP 4.x to run on a diverse range of currently supported architectures.

• Our initial expectations were low for GPUs…
Performance?

Real need to target local/shared memory for BUDE – OpenMP 5?

OpenMP 4.0 nearly as fast as hand-optimised CUDA with CCE

Martineau, M., McIntosh-Smith, S. Gaudin, W., *Evaluating OpenMP 4.0’s Effectiveness as a Heterogeneous Parallel Programming Model*, 2016, HIPS’16
Achieving good OpenMP 4.x performance on GPUs

• Our findings so far:
  • You **can** achieve good performance with OpenMP 4.x.

• Achieve this by:

  • Keeping data resident on the device for the greatest possible time.

  • Collapsing loops with the **collapse** clause, creating a large enough iteration space to saturate a device such as a GPU.

  • Using the **simd** directive to vectorize inner loops.

  • Using **schedule**(static, 1) for coalescence (obsolete).

  • Using profilers such as **nvprof**.
Can we do even better?

Through extensive tuning of the clang/llvm compiler implementation we achieved CloverLeaf performance within 9% of hand optimized CUDA code…

Martineau, M., Bertolli, C., McIntosh-Smith, S., et al. *Broad Spectrum Performance Analysis of OpenMP 4.5 on GPUs*, 2016, PMBS’16
An important observation

OpenMP can express more levels of parallelism than we often need
• Leaves ambiguity when not all levels needed
• How the "simd" directive is implemented now and in the future will have a big impact on success…
Pragmatic portability

// CCE targeting NVIDIA GPU
#pragma omp target teams distribute simd
for(...) {
}

// Clang targeting NVIDIA GPU
#pragma omp target teams distribute parallel for schedule(static, 1)
for(...) {
}

// GCC 6.1 target AMD GPU
#pragma omp target teams distribute parallel for
for(...) {
}

// ICC targeting Intel Xeon Phi
#pragma omp target if(offload)
#pragma omp parallel for simd
for(...) {
}

Four different styles of pragma for the same kernel is not what anyone wants ... (See Martineau's talk for the answer!)
What are the main issues?

• Dealing with the host-device data movement got a lot easier in OpenMP 4.5
• We want to write performance portable OpenMP
  • Ideally the same set of pragmas on all hardware platforms (CPUs, GPUs, …)
• C++ still causing all sorts of performance problems
• Need to be able to exploit the emerging memory hierarchies (HBM etc)
• Then there's still multi-device support, heterogeneous computing, dynamic load balancing across devices...
Summary

• Quite a few OpenMP 4.x implementations now emerging (Intel, Cray, PGI, gcc, clang/llvm…)
• Levels of maturity are quite mixed
• Demonstrated OpenMP 4.x can achieve GPU performance similar to OpenCL/CUDA
• Can also achieve a reasonable degree of performance portability, although need to jump through hoops with the pragmas
• The signs are promising for OpenMP 4.x!
Performance portability refs

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

- Assessing the Performance Portability of Modern Parallel Programming Models using TeaLeaf
  Martineau, M., McIntosh-Smith, S. & Gaudin, W.
  Concurrency and Computation: Practice and Experience (April 2016), to appear

- GPU-STREAM v2.0: Benchmarking the achievable memory bandwidth of many-core processors across diverse parallel programming models
  Deakin, T. J., Price, J., Martineau, M. J. & McIntosh-Smith, S. N.
  First International Workshop on Performance Portable Programming Models for Accelerators (P^3MA), ISC 2016

- https://github.com/UoB-HPC/
For related software and papers

See: [http://uob-hpc.github.io](http://uob-hpc.github.io)

GPU-STREAM:
[https://github.com/UoB-HPC/GPU-STREAM](https://github.com/UoB-HPC/GPU-STREAM)

CloverLeaf:
[https://github.com/UoB-HPC/CloverLeaf-OpenMP4](https://github.com/UoB-HPC/CloverLeaf-OpenMP4)

TeaLeaf:
[https://github.com/UoB-HPC/TeaLeaf](https://github.com/UoB-HPC/TeaLeaf)

SNAP:
[https://github.com/UoB-HPC/SNAP_MPI_OpenCL](https://github.com/UoB-HPC/SNAP_MPI_OpenCL)
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>System details</th>
</tr>
</thead>
<tbody>
<tr>
<td>K20X</td>
<td>Cray® XC40, NVIDIA® K20X GPU, Cray compilers version 8.5, gnu 5.3, CUDA 7.5</td>
</tr>
<tr>
<td>K40</td>
<td>Cray® CS cluster, NVIDIA® K40 GPU, Cray compilers version 8.4, gnu 4.9, CUDA 7.5</td>
</tr>
<tr>
<td>K80</td>
<td>Cray® CS cluster, NVIDIA® K40 GPU, Cray compilers version 8.4, gnu 4.9, CUDA 7.5</td>
</tr>
<tr>
<td>S9150</td>
<td>AMD® S9150 GPU. Codeplay® computeCpp compiler 2016.05 pre-release. AMD-APP OpenCL 1.2 (1912.5) drivers for SyCL. PGI® Accelerator™ 16.4 OpenACC</td>
</tr>
<tr>
<td>GTX 980 Ti</td>
<td>NVIDIA® GTX 980 Ti. Clang-ykt fork of Clang for OpenMP. PGI® Accelerator™ 16.4 OpenACC. CUDA 7.5</td>
</tr>
<tr>
<td>Fury X</td>
<td>AMD® Fury X GPU (based on the Fiji architecture).</td>
</tr>
<tr>
<td>Sandy Bridge</td>
<td>Intel® Xeon® E5-2670 CPU. Intel® compilers release 16.0. PGI® Accelerator™ 16.4 OpenACC and CUDA-x86. Intel® OpenCL runtime 15.1. Codeplay® computeCpp compiler 2016.05 pre-release.</td>
</tr>
<tr>
<td>Haswell</td>
<td>Cray® XC40, Intel® Xeon® E5-2698 CPU. Intel® compilers release 16.0. PGI® Accelerator™ 16.3 OpenACC and CUDA-x86. Gnu 4.8 for RAJA and Kokkos</td>
</tr>
<tr>
<td>Broadwell</td>
<td>Cray® XC40, Intel® Xeon® E5-2699 CPU. Intel® compilers release 16.0. PGI® Accelerator™ 16.3 OpenACC and CUDA-x86. Gnu 4.8 for RAJA and Kokkos</td>
</tr>
<tr>
<td>KNL</td>
<td>Intel® Xeon® Phi™ 7210 (knights landing) Intel® compilers release 16.0. PGI® Accelerator™ 16.4 OpenACC with target specified as AVX2.</td>
</tr>
<tr>
<td>Power 8</td>
<td>IBM® Power 8 processor with the XL 13.1 compiler.</td>
</tr>
</tbody>
</table>