Writing Performance Portable OpenMP 4.5

Investigating the existing compiler technologies that support OpenMP 4.x

Matt Martineau - UoB (m.martineau@bristol.ac.uk)
James Price – UoB (j.price@bristol.ac.uk)
Simon McIntosh-Smith - UoB (cssnmis@bristol.ac.uk)
Wayne Gaudin – UK Atomic Weapons Establishment
Considering no nesting, the following is a highly simplified perspective of existing levels of parallelism
Intel provided the first commercial support for OpenMP 4.0

This implementation chooses to default to a single team when inside a *teams* region

In fact, when targeting a Xeon Phi, you can achieve good performance without explicitly using the *teams* directive

```
#pragma omp target
#pragma omp parallel for [simd]
```

The new Xeon Phi architectures don’t require offloading
Clang’s OpenMP has been fed into by multiple vendors, and an implementation targeting NVIDIA GPUs by IBM

We discuss the alpha version. Some (potentially outdated) information is available here https://clang-omp.github.io/

A new version is now OpenMP 4.5 feature complete and available at https://github.com/clang-ykt
```
#pragma omp target teams distribute
for(int ii = 0; ii < y; ++ii) {
    #pragma omp parallel for schedule(static, 1)
    for(int jj = 0; jj < x; ++jj)
        ...
}
```

Create a single team per streaming multiprocessor in the GPU e.g. 14 on K20X

Chunk the outer loop and pass to the master thread of each team

Execute the region in parallel as a team, work-sharing the inner jj loop

Schedule the threads in a round-robin fashion, enabling coalescence on NVIDIA GPUs
In version 4.0 scalar variables were implicitly mapped into device memory, and copied at the beginning and end of each **target** region.

```c
#pragma omp target teams distribute map(tofrom: localvars ... )
for(int ii = 0; ii < y; ++ii) {
    #pragma omp parallel for schedule(static, 1)
    for(int jj = 0; jj < x; ++jj)
        ...
}
```

Small performance improvements could be gained from overriding this, but this has an unfortunate conflict with the OpenMP 4.5 change to **firstprivate** scalars.

```c
#pragma omp target teams distribute map(to: localvars ... )
```
The Cray compilers represented the first commercial implementation to target NVIDIA GPUs, and was released towards the end of 2015.

The initial implementation was primarily OpenMP 4.0, but some features from version 4.5 were implemented too.

Cray again chooses a different approach to scheduling the target region on the target device.
A total of $y$ teams, containing 128 threads each, are initialised.

```c
#pragma omp target teams distribute
for(int ii = 0; ii < y; ++ii) {
    #pragma omp simd
    for(int jj = 0; jj < x; ++jj)
        ...
}
```

The outer loop iterations are chunked and given to the master thread of each team.

The inner loop is scheduled in a round robin order to each thread within the encountering team, as if each thread were a SIMD lane.

In the one dimensional loop case, or with a collapsed loop, the defaults change to 128 teams of 128 threads.
GCC 6.1 has full support for OpenMP 4.5, including an implementation for targeting AMD devices via HSAIL.

This implementation is highly constrained, allowing only the combined directive

\[\texttt{#pragma omp target teams distribute parallel for}\]

Further to this no clauses are supported, and \texttt{simd} is not supported.

The default for N iterations is N/64 teams of 64 threads.
Worst Case Differences

Four implementations, four subtly different potential approaches

```c
#pragma omp target teams distribute
for(int ii = 0; ii < y; ++ii)

#pragma omp target teams distribute \ parallel for schedule(static,1)
for(int ii = 0; ii < y; ++ii)

#pragma omp target
#pragma omp parallel for
for(int ii = 0; ii < y; ++ii)

#pragma omp target teams distribute \ parallel for
for(int ii = 0; ii < y; ++ii)
```

Cray C Compiler (v8.5)
Clang Compiler (alpha)
Intel C Compiler (v16.0)
GCC C Compiler (v6.1)
We collate our experiences into a number of suggestions for performance portability

• Use the most expressive combined construct applicable
  ```
  #pragma omp target teams distribute parallel for [simd]
  ```

• Even if the compiler doesn’t require teams / distribute / parallel for, they should be included where they make sense
• Using `simd` on those loops that require vectorisation

• The collapse and schedule clauses won’t harm portability, but likely need to be set on a per-architecture basis for performance – only use them if strictly necessary

• Although tuning the number of teams and threads will be important for performance critical it can severely harm performance portability
Performance Results for Simple Benchmarks

Results for kernels across multiple devices

Percentage of Peak (%)

- CUDA K20x
- CCE 8.5.0 K20x
- Clang K20x
- GCC 6.1 Kaveri
- CCE 8.5.0 BW
- ICC 16 KNL

Test Cases:
- vec_add_2d
- vec_add_sqr
- vec_add_and_mul
- reverse_indirect
- column_indirect
- 2pt_stencil
- 2pt_stencil_10_gap
- 5pt_stencil_2d
- 9pt_stencil_2d
- 7pt_stencil_3d
- 27pt_stencil_3d
- tealeaf_cheby_iter
- clover_energy_flux
- snap_sweep
- compute_bound

Matt Martineau, Simon McIntosh-Smith, James Price, Wayne Gaudin
Pragmatic Kernels, and Mini-apps including TeaLeaf, CloverLeaf, and SNAP
https://github.com/UK-MAC/
https://github.com/UoB-HPC/

Evaluating OpenMP 4.0's Effectiveness as a Heterogeneous Parallel Programming Model
Martineau, M., McIntosh-Smith, S., & Gaudin, W.
Parallel and Distributed Processing Symposium Workshops, 2016

Performance Analysis and Optimization of Clang’s OpenMP 4.5 GPU Support
Martineau, M., Bertolli, C., McIntosh-Smith, S., et al.
To be presented at PMBS’16 (November 2016)

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