Xeon Phi experiences for the CloverLeaf and TeaLeaf benchmarks

Michael Boulton

June 5, 2014

Cloverleaf and TeaLeaf

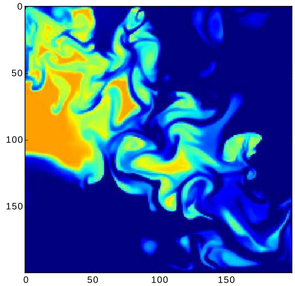
Both work on the same 2 dimensional structured grid with a reflective boundary. Parallel versions of both have been written in FORTRAN/C/OpenMP and OpenCL.

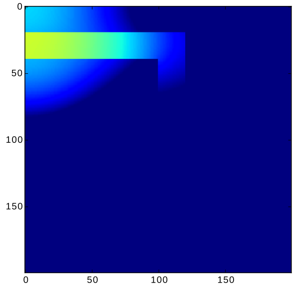
Cloverleaf

- Eulerian/Lagrangian hydrodynamics - solves Euler equations
- Explicit update

TeaLeaf

- Heat diffusion - solves system of linear equations using matrix free method
- Implicit update





Cloverleaf and TeaLeaf

Mesh is defined from `x_min,y_min` to `x_max,y_max`, with 2 halo cells around the outside. A typical loop looks like:

```
!$OMP PARALLEL
!$OMP DO REDUCTION(+:pw)
DO k=y_min,y_max
    DO j=x_min,x_max
        w(j, k) = (1.0_8                                          &
            + ry*(Ky(j, k+1) + Ky(j, k))                          &
            + rx*(Kx(j+1, k) + Kx(j, k)))*p(j, k)                 &
            - ry*(Ky(j, k+1)*p(j, k+1) + Ky(j, k)*p(j, k-1))      &
            - rx*(Kx(j+1, k)*p(j+1, k) + Kx(j, k)*p(j-1, k))

        pw = pw + w(j, k)*p(j, k)
    ENDDO
ENDDO
!$OMP END DO
!$OMP END PARALLEL
```

FORTRAN/C/OpenMP on Xeon Phi

Tests were run on an 'SE10P' card, roughly equivalent to a '5110P' (61 cores @ 1.1GHz)

- Slight problem with FORTRAN not being fully vectorized originally, but fixed in the latest version of compiler
- Running OpenMP across whole device quite slow, flat MPI quite slow. Best balance was to use one MPI task per CPU with one OpenMP task per core
- Slight speed difference between C and FORTRAN but nothing very noticeable

Worked immediately without any other changes needed, and was 40-45% faster than on a dual socket Ivy Bridge Xeon CPU.

OpenCL on Xeon Phi

- Useful information as to whether the kernels were vectorised or not before having to profile them. Vectorised differently on CPU/Phi?

  ```
  Build started
  Kernel <viscosity> was successfully vectorized
  Done.
  ```

- Had to change kernel launches to use offsets:

  ```
   if(row >= (y_min + 1) && row <= (y_max + 1)
  && column >= (x_min + 1) && column <= (x_max + 1))
  ```

  Managed to get a ∼30% speed up by removing checks that depended on the x dimension

- Source level profiling helped find some slow bits which I didn't think were slow

PdV kernel

tlb

Tools/libraries

- Vec reporting helps a lot, especially in huge programs - vectorisation is as good as if not better than gcc (fixed with recent SLP loop vectorisation?). Next version of compiler has better reporting
- OpenMP 4.0 lets you specify loops as SIMD in a portable manner instead of using `#pragma simd`
- Source level profiling incredibly useful, especially for OpenCL where it can find unexpected problems

Experiences

Good things:

- Performance per Watt/volume/cost is better than a CPU
- A lot easier to get things running on it - anything that helps on the CPU helps on the Xeon Phi and vice versa so it's not wasted effort
- Once you can properly exploit parallelism in code it does have quite good performance

Bad things:

- Feels immature - setting environment variables to improve speed, driver updates might speed up code, might have to spend a bit of time aligning loops, finding optimal MPI/OpenMP balance, etc
- FP performance is good for memory bandwidth isn't so good
- Streaming store instructions not generated enough

Most of these seem to be fixed in newer drivers/compilers/next gen hardware