Offload Mode Case Study

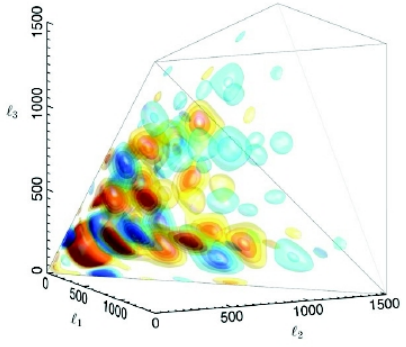
James Briggs

[1]COSMOS DiRAC

April 28, 2015

Case Study: Modal2d

- MODAL is an early universe simulation and analysis code used to probe the Cosmic Microwave Background (CMB).
- Analyses higher-order correlation functions beyond the power spectrum.
- Novel algorithm for efficient mode expansion to measure reconstruct the CMB **bispectrum** for the first time.
- Fast and efficient way to probe cosmological data for hints of *new physics* in the early universe.



Bispectrum of CMB. Source: *Planck 2013 results. XXIV. Constraints on primordial non-Gaussianity*

Surveying the Code

- Original code is pure C and parallelised with **MPI only**.
- Already vectorised the code on Xeon to great success and there is enough potential parallelism for threads $\Rightarrow$ great Xeon Phi potential?
- Library dependencies – GSL, iniparser, FFTW – for initialisation and I/O. (Outside of main loop).
  - Compiling for native with `-mmic` tedious because I need to compile the external libraries for Xeon Phi too.
- Likely less tedious to test Xeon Phi with offload than native.

Pseudo-code

- Want to offload the computationally most expensive part.
- Pseudo-code for main loop:

```
MPI_for n in primoridal_modes:
    MPI_for m in late_modes:
        y = double[xsize]
        for x in range(0,xsize):
            y[i] += x[i]*x[i] * gamma_pt(n,m,i);
        gamma[n][m] = gsl_integrate(x[], y[]);

MPI_Reduce(gamma[][]);
```

- Output = `gamma[][]`.
- The n and m loops are decomposed over MPI tasks. Typical size $\mathcal{O}(1000)$.
- `gamma_pt` routine has a *lot of work* and is well vectorised.

Making it Offloadable (1/3)

```
MPI_for n in primoridal_modes:
    MPI_for m in late_modes:
        y = double[xsize]
        for x in range(0,xsize):
            y[i] += x[i]*x[i] * gamma_pt(n,m,i);
        gamma[n][m] = gsl_integrate(x[], y[]);

MPI_Reduce(gamma[][]);
```

- Integration has GSL dependency.
- Negligible in profile $\Rightarrow$ write my own integration routine and remove the dependency.

Making it Offloadable (2/3)

```
MPI_for n in primoridal_modes:
    MPI_for m in late_modes:
        y = double[xsize]
        for x in range(0,xsize):
            y[i] += x[i]*x[i] * gamma_pt(n,m,i);
        gamma[n][m] = my_integrate(x[], y[]);

MPI_Reduce(gamma[][]);
```

- ~~Integration has GSL dependency.~~
- Negligible in profile $\Rightarrow$ write my own integration routine and remove the dependency.

Making it Offloadable (3/3)

- Add offload pragma before main loop...

```
#pragma offload target(mic:0) \
    inout(gamma : length(N*M) ALLOC FREE) \
    in(primordial_modes, late_modes, mpi_vars)
MPI_for n in primordial_modes:
    MPI_for m in late_modes:
        y[0:xsize] = 0.0;
        for x in range(0,xsize):
            y[i] += x[i]*x[i] * gamma_pt(n,m,i);
        gamma[n][m] = my_integrate(x[], y[]);

// end offload region
MPI_Reduce(gamma[][]);
```

- Done? Nope. Just starting!

Tracking Down the Offloadables (1/3)

- **Doesn't compile!** – Missing symbols.
- Need to track down all the functions and global variables used in the main loop and declare them **offloadable**:

```
__attribute__((target(mic)))
double gamma_pt(int n, int m, int i);
```

- This part can be *fiddly*. Help:
  - Missing symbols will be found at compile time.
  - **ctags** with Vim or Emacs very useful for chasing down dependencies.
  - IDE could also have useful tools to help do this.

Tracking Down the Offloadables (2/3)

- Code now compiles, but the result is **garbage!**
- Declaring offloadable is only half the battle.
- Code has a lot of read-only global variables.

- Declaring variables offloadable just means that their symbols are **visible** on the MIC side.
- **Data isn't necessarily also there**.

Tracking Down the Offloadables (3/3)

- Need to track down the required global variables, and do an `#pragma offload_transfer` when their values are set.
- Allinea DDT offload debugger is useful for finding uninitialised variables offload-side.
- **Now done** :-).

Aside: Multi-dimensional Arrays

- Main loop reads several multi-dimensional arrays.
- These are implemented as *arrays-of-pointers*.
- Offload data transfers in LEO won't offload these properly.

- **Work-around**: transfer them **flat**, then rebuild / reinterpret dimensions on the 'other-side'.
- C one-liner to reinterpret flat array (basis_flat) as 2-dimensional (basis):

```
double (* restrict basis)[lsize_pad] = (double (*restrict)[
    lsize_pad]) basis_flat;
```

Xeon Phi Performance

- After offloading added threads via OpenMP of nm loops.
- This makes code OpenMP/MPI hybrid. Each MPI rank offloads to its own card and uses all the cores.
- With vectorisation enabled in main loop, test case:
  - 2× SandyBridge = 167s (2.7× original).
  - 1× Xeon Phi = 75s (6.0× original).
  - 1× Xeon Phi = 2.23× 2× SandyBridge.