Many-core Implementation of Numerical 3D Isotropic Acoustic Wave Equation and Analysis of its Performance Portability

Oil & Gas HPC Conference
Rice University
March 16, 2017

Reid Atcheson
Accelerator Software Engineer

Experts in numerical software and High Performance Computing
Coding for Portability

With acoustic wave equation as example, I show that:

- Performance portability possible between Xeon and Xeon Phi
- Tuning strategies from Xeon Phi transfer to Xeon
I will proceed in following order:

1. Base Kernel
2. Vectorization with OpenMP 4.0+
3. Blocking for cache efficiency
4. OpenMP Tasks for Hyperthreading
5. MCDRAM considerations
// Loop over gridpoints
for(index_t i=ORDER/2;i<nx-ORDER/2;i++){
    for(index_t j=ORDER/2;j<ny-ORDER/2;j++){
        for(index_t k=ORDER/2;k<nz-ORDER/2;k++){
            // Compute stencil
            for(index_t m=-ORDER/2;m<ORDER/2;m++){
                out[id(i,j,k)]+=cx[m+ORDER/2]*in[id(i+m,j,k)];
                out[id(i,j,k)]+=cy[m+ORDER/2]*in[id(i,j+m,k)];
                out[id(i,j,k)]+=cz[m+ORDER/2]*in[id(i,j,k+m)];
            }
        }
    }
}

The base kernel I will tune.
Vectorization

- With cores occupied, need to make sure vector units are busy
- OpenMP also helps us here
- But will need to change data access for unit stride
for (index_t k = ORDER/2; k < nz - ORDER/2; k++) {
    for (index_t m = -ORDER/2; m <= ORDER/2; m++) {
        // Compute stencil
        
    }
}

Exchanging loops for unit stride access. Indicate aligned pointers.

for (index_t m = -ORDER/2; m < ORDER/2; m++) {
    // Initialize pointers o,ix,iy,iz
    #pragma omp simd aligned (o, ix, iy: ALIGNMENT)
    for (index_t k = 0; k < nz - ORDER; k++) {
        // Compute Stencil
        
    }
}
Cache Efficiency

- The loop interchange is performance-portable
- Unit-stride access *almost always* ideal.
- But now need to better utilize cache.
Portable Cache Efficiency

```c
#pragma omp parallel for
for(index_t i=ORDER/2;i<nx-ORDER/2;i++){
    for(index_t j=ORDER/2;j<ny-ORDER/2;j++){
        for(index_t k=ORDER/2;k<nz-ORDER/2;k++){
            // Acoustic wave equation code here
        }
    }
}
```

Change from point-oriented loop to parameterized block-oriented loop. Add parameterized collapse to ensure sufficient parallelism.

```c
#pragma omp parallel for collapse(COLLPASE)
for(index_t i=ORDER/2;i<nx-ORDER/2;i+=BLKI){
    for(index_t j=ORDER/2;j<ny-ORDER/2;j+=BLKJ){
        for(index_t k=ORDER/2;k<nz-ORDER/2;k+=BLKK){
            for(index_t _i=i;_i<i+BLKI;_i++){
                for(index_t _j=j;_j<j+BLKJ;_j++){
                    for(index_t _k=k;_k<k+BLKK;_k++){
                        // Acoustic wave equation code here
                    }
                }
            }
        }
    }
}
```
Cache Efficiency

- Cache efficiency important on Xeon and Xeon Phi
- Main difference: blocking parameters change
- Hyperthreading also benefits from locality
- OpenMP tasks enable portable hyperthreading
#pragma omp parallel for collapse(COLLAPSE) 
for(index_t i=ORDER/2;i<nx-ORDER/2;i+=BLKI){
for(index_t j=ORDER/2;j<ny-ORDER/2;j+=BLKJ){
for(index_t k=ORDER/2;k<nz-ORDER/2;k+=BLKK){
    //Loop over sub-block...
}}}

Indicate that sub-blocks may execute asynchronously. Add thread pinning to ensure "for loop" threads go to cores and sub-blocks go to hyperthreads.

#pragma omp parallel for \
collapse(COLLAPSE) proc_bind(spread) schedule(SCHEDULE) 
for(index_t i=ORDER/2;i<nx-ORDER/2;i+=BLKI){
for(index_t j=ORDER/2;j<ny-ORDER/2;j+=BLKJ){
for(index_t k=ORDER/2;k<nz-ORDER/2;k+=BLKK){
    #pragma omp task
    {
        //Loop over sub-block...
    }}
}}
High Bandwidth Memory

- Hyperthreading on KNL usually helps
- If it does not, just turn off the OpenMP task
- Next I discuss MCDRAM: high bandwidth memory
Performance: High Bandwidth Memory

Two main cases for MCDRAM:

1. Cache Mode
   - No code change necessary
   - MCDRAM operates as LLC

2. Flat Mode
   - Use libmemkind to control allocations
   - Allocate MCDRAM staging arrays
   - Process grid as series of chunks
The following OpenMP features enabled this portable code

1. Parallel for loops
   - `proc_bind` directive
   - `collapse` directive
   - `schedule` directive

2. Vectorization
   - `align` directive

3. Tasks
The following parameters are tuned to target architecture

1. BLKI
2. BLKJ
3. BLKK
4. SCHEDULE
5. COLLAPSE
6. ALIGNMENT
7. Number of Hyperthreads (1X, 2X, 4X)
For 16th order stencil I found following results:

- **Hand-tuning parameters:**
  - KNL: 5 billion gridpoints/s
  - 2X Haswell: 4 billion gridpoints/s

- **Adding extra blocking layer and brute-force selecting parameters**
  - KNL: 8 billion gridpoints/s
  - 2X Haswell: 4 billion gridpoints/s

Note: 4 billion gridpoints/s is roofline peak for 2X haswell
Discussion

- Tuning strategies are generalizable
- This keeps codebase under control
- But does not sacrifice performance
- I only scratched surface here!
Discussion

- Free Knights Landing training April 25, 26 here at Rice university.
- [http://www.nag.com/content/nag-and-intel-2-day-training-course](http://www.nag.com/content/nag-and-intel-2-day-training-course)