Evaluating Portability of OpenMP for SNAP using Roofline analysis

Neil Mehta, Rahul Gayatri, Yasaman Ghadar, Christopher Knight, and Jack Deslippe

NERSC

27th August 2021
Adapting to Exascale

<table>
<thead>
<tr>
<th>System</th>
<th>Perlmutter</th>
<th>Aurora</th>
<th>Frontier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>AMD Milan</td>
<td>Intel Xeon SR</td>
<td>AMD EPYC</td>
</tr>
<tr>
<td>Device</td>
<td>NVIDIA A100</td>
<td>Intel Xe Ponte Vecchio</td>
<td>AMD Radeon Instinct</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test-bed</th>
<th>Cori</th>
<th>JLSE Iris</th>
<th>Tulip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>Intel Skylake</td>
<td>Intel Skylake</td>
<td>AMD EPYC</td>
</tr>
<tr>
<td>Device</td>
<td>NVIDIA A100</td>
<td>Intel Gen9</td>
<td>AMD MI60</td>
</tr>
<tr>
<td>Compiler</td>
<td>LLVM 13/NVHPC 21.3</td>
<td>oneapi (20201008)</td>
<td>rocm 3.6.0 (aomp 11.0)</td>
</tr>
</tbody>
</table>

- OpenMP 4.5 directives, as it requires less intensive code modifications and have compiler support by all major GPU vendors
Introduction to TestSNAP

- J determines bispectrum
- TestSNAP proxy app mimics computational load
- Test performance for J = 2, 8, and 14 (ECP FOM problem size for EXAALT MD project)
- Number of atoms: 2000 atoms
- Number of steps: 100

```c
for(int natom = 0; natom < num_atoms; ++natom)
{
    // build neighbor-list for all atoms
    build_neighborlist();

    // compute atom specific coefficients
    compute_U(); // Ulist[idx_max] and Ulisttot[idx_max]
    compute_Y(); // Ylist[idx_max]

    // for each (atom,neighbor) pair
    for(int nbor = 0; nbor < num_nbor; ++nbor)
    {
        compute_dU(); // dUlist[idx_max][3]
        compute_dE(); // dElst[3]
        update_forces()
    }
}```
Kernel optimizations for OpenMP (1/4)

- Arrays created using
  classes that include
  pointer to contiguous
  block of memory
- Case 1: baseline

Run times:
llvm   A100: 0.358 s
nvc++  A100: 0.321 s

```c
void add_uarraytot()
{
    #pragma omp target teams distribute parallel for
    for(int natom = 0; natom < num_atoms; ++natom)
        for(int nbor = 0; nbor < num_nbor; ++nbor)
            for(int j = 0; j < idxu_max; ++j)
                ulisttot(natom,j) += ulist(natom,nbor,j);
}
```
Kernel optimizations for OpenMP (2/4)

- Exploit the ability to collapse nested for loops
- Case 2: collapse

Run times:
llvm A100: 0.0559 s
nvc++ A100 : 0.0432 s
Kernel optimizations for OpenMP (3/4)

- Column major data access: atom loop as fastest moving index causes performance degradation
- Case 3: column major

Run times:
llvm A100: 0.0622 s
nvc++ A100 : 0.0516 s
Kernel optimizations for OpenMP (4/4)

- Make atom loop (fastest moving index) as inner most loop
- Case 4: reorder loop

Run times:
LLVM A100: 0.0241 s
nvc++ A100: 0.0141 s
TestSNAP profiling data

<table>
<thead>
<tr>
<th>Version</th>
<th>Intel Gen9</th>
<th>AMD MI60</th>
<th>NVIDIA A100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rank</td>
<td>Time (%)</td>
<td>Kernel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>compute_Y</td>
<td>57.01</td>
</tr>
<tr>
<td>1</td>
<td>65.65</td>
<td>compute_dU</td>
<td>31.53</td>
</tr>
<tr>
<td>2</td>
<td>19.15</td>
<td>compute_U</td>
<td>8.61</td>
</tr>
<tr>
<td>3</td>
<td>10.58</td>
<td>compute_dE</td>
<td>2.44</td>
</tr>
<tr>
<td>4</td>
<td>4.02</td>
<td>WriteBuffer</td>
<td>0.29</td>
</tr>
<tr>
<td>5</td>
<td>0.41</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version</th>
<th>Serial (Skylake)</th>
<th>OpenMP offload GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LLVM/11 ICX</td>
<td>Gen9 MI60 A100</td>
</tr>
<tr>
<td>Step time (s/step)</td>
<td>9.7671</td>
<td>9.8669</td>
</tr>
<tr>
<td>Grind time (ms/atm-stp)</td>
<td>4.8835</td>
<td>4.9334</td>
</tr>
<tr>
<td>compute_U (s)</td>
<td>0.6211</td>
<td>0.6221</td>
</tr>
<tr>
<td>compute_Y (s)</td>
<td>7.6839</td>
<td>7.6789</td>
</tr>
<tr>
<td>compute_dU (s)</td>
<td>1.2008</td>
<td>1.3363</td>
</tr>
<tr>
<td>compute_dE (s)</td>
<td>0.2604</td>
<td>0.2288</td>
</tr>
</tbody>
</table>
- Integrated GPU architecture
- 3 GPU slices, each having 3 sub-slices
- Each subslice has 8 execution unit (EU)
- Each EU has 7 thread
- Total of 504 threads per GPU
GPU architecture of AMD MI60

- Each GPU has 64 ‘Next-gen Compute Units’ or ‘NCUs’
- Analogous to NVIDIA warps, on AMD GPU, each ‘wavefront’ consists of 64 work items, i.e. threads
- MI60 capable of launching 4096 work items
108 SM per GPU
Total of 6912 FP32 cores per GPU
Understanding Roofline model

- Region to the left of ‘machine balance line’ represents memory bound
- Region to the right represents compute bound
- Kernels shown in blue and green are close to peak memory bandwidth and compute throughput, representing bounded performance
- Red kernels are neither compute or memory bound and show potential for greater optimization
- Ideally kernel should shift upwards and rightwards
• Kernels are memory bound on A100
• Compute_y kernel close to machine balance line
• Other three kernels are bound by HBM bandwidth
• Similar memory bound result for 3 out of 4 kernels as NVHPC
• Compute_Y however is compute_bound
• May be due to better memory transfer protocol but performance is lower
Hierarchical roofline from A100

- High reuse between DRAM and L2
- Poor cache utilization between L2 and L1
- Lower AI does not correspond to lower code runtime
- Code AI capped by L2 cache
Roofline from Intel Gen9

- Kernels are close to the DRAM bandwidth line, indicating memory bound
- Theoretically, roofline cannot cross bandwidth line
- Kernels crossing DRAM bandwidth indicates eDRAM memory usage
- Indicates performance capped by memory bandwidth and not compute capability
Kernels again close to the DRAM bandwidth line, indicating memory bound
Kernels are closer to compute bound regime
AI in the same order of magnitude
Summary

- OpenMP offload directives can be used successfully to write portable code across all three GPUs
- Compiler maturity plays important role in code performance
- Profiles indicate expected difference in the performance between all three GPUs
- Code optimization play a key role in profiling and vice versa
- Kernels on all three GPUs are memory bound
- Differences in how compilers and GPU hardware address memory transfer leading to either drop and increase in AI

TestSNAP code available at https://github.com/FitSNAP/TestSNAP/tree/OpenMP4.5
Acknowledgement

- We would like to thank Dr. Aidan Thompson for providing us the initial TestSNAP code, which was then highly modified for this work.
- We would like to thank Drs. Danny Perez, Noah Reddell, and Nicholas Malaya for enabling us access and providing compute resources on the DOE’s Cray Tulip machine.
- This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.
Joint Laboratory for System Evaluation (JLSE)
https://jlse.anl.gov

We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.
Thank You
Backup slides
Speed-ups due to optimizations

- All results normalized to baseline
- Column major data access: atom loop as fastest moving index causing performance degradation
Variadic versus non-variadic arrays

- A single class to create multidimensional arrays for every dimension and data type
- Class is templated over the number of dimensions
- Variadic template pack expansion is used to calculate the offset in each dimension

```
define ARRAY2D ArrayMD<int, 2>
define ARRAY3D ArrayMD<int, 3>
ARRAY2D y(N, N);
ARRAY2D x(N, N);
ARRAY3D m(N, N, N);
```

- An array class is templated but only per data type
- Requires duplication of templated class for each multi-dimensional array class
## Profiling data - NVIDIA V100

**Grid size:** 1000 x 1 x 1  
**Block size:** 128 x 1 x 1

<table>
<thead>
<tr>
<th>Metric</th>
<th>Baseline</th>
<th>With omp for</th>
<th>With omp simd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel time (s)</td>
<td>10.70</td>
<td>1.82</td>
<td>1.81</td>
</tr>
<tr>
<td>Total time (s)</td>
<td>11.85</td>
<td>2.99</td>
<td>2.96</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Variadic</th>
<th>Non-variadic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel time (s)</td>
<td>1.81</td>
<td>7.20</td>
</tr>
<tr>
<td>Total time (s)</td>
<td>2.96</td>
<td>8.75</td>
</tr>
</tbody>
</table>
### Profiling data - NVIDIA V100

<table>
<thead>
<tr>
<th>Metric</th>
<th>Baseline</th>
<th>With omp for</th>
<th>With omp simd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel time (s)</td>
<td>11.00</td>
<td>2.17</td>
<td>2.16</td>
</tr>
<tr>
<td>Total time (s)</td>
<td>13.10</td>
<td>4.15</td>
<td>4.16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Variadic</th>
<th>Non-variadic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel time (s)</td>
<td>2.16</td>
<td>3.08</td>
</tr>
<tr>
<td>Total time (s)</td>
<td>4.16</td>
<td>5.81</td>
</tr>
</tbody>
</table>
## Profiling data - Intel Gen9

<table>
<thead>
<tr>
<th>Metric</th>
<th>Baseline</th>
<th>With omp for</th>
<th>With omp simd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel time (s)</td>
<td>275.43</td>
<td>20.36</td>
<td>20.41</td>
</tr>
<tr>
<td>Total time (s)</td>
<td>275.94</td>
<td>20.87</td>
<td>20.91</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Variadic</th>
<th>Non-variadic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel time (s)</td>
<td>20.41</td>
<td>21.36</td>
</tr>
<tr>
<td>Total time (s)</td>
<td>20.91</td>
<td>22.37</td>
</tr>
</tbody>
</table>
Hierarchical roofline from Gen9

- Rooflines measured for DRAM, GTI, and L3 (SLM) cache level
- Reduction in AI due to lower FLOP count for each unit of memory moved across that memory level
- Larger difference between the kernel rooflines indicate good data reuse, i.e., good use of cache hierarchy