I. MPS method and our in-house code: P-Flow

Moving Particle Semi-implicit (MPS) method

MPS method is a sort of particle methods used for computation fluid dynamics. It is originally developed for simulating fluid dynamics such as fragmentation of incompressible fluid. Target fluid or objects are divided into particles and each particle interacts with neighbor-particles. Search of neighbor-particles is a main bottleneck of MPS method. We’re researching and developing in-house program.

Complexity of data structure and memory access pattern

Our in-house MPS code; P-Flow adopts complex data structure for scalability and extensibility. It is written in Fortran and utilizes multiple de-rived types.

For physical values (SoA)

II. Search for neighbour-particle

MPS performs search for neighbor-particle multiple times in each time step (e.g. density calculation). Each particle drifts as time-step progress and neighbor-particle changes.

Search for neighbour-particle with bucket

The following 4-nested loop step is used.
1. Choose a target bucket
2. Pickup a target particle (red) in the bucket
3. Traverse 3*3*3 adjacent buckets (in no particular order)
4. Search particles in a bucket
4-1. Calculate distance and weight between the target particle
4-2. Accumulate weighted physical value to a target particle (in no particular order)

✓ Maximum number of particles in a bucket is 33

Memory access pattern and Characteristics

✓ Indefinite loop: number of particles in a bucket is uncertain
✓ Vectorization: Each target particle accesses different bucket and particle
✓ Cache: Not easy to utilize cache since adjacent particles changes time by time
✓ Parallelism: Thousands of in-flight data requests can hide memory access latency

MPS simulation: A collapse of water column

III. Porting and optimization strategy

In order to make maintenance easy and keep the structure of original code, OpenACC and OpenMP are used for GPU and CPU, respectively. OpenACC works well on the complex data structure.

P100: Bucket per thread block, particles per CUDA thread

Each target bucket (Loop1) is assigned to thread block. Each particle in the same bucket (Loop2) is assigned to CUDA thread. Loop 3 and 4 are processed sequentially by each CUDA thread.

Memory access of threads in bucket becomes the same since each particle in the same bucket accesses the same particle in loop 3 and 4. But not coalesced and low bandwidth utilization. Utilization of CUDA thread is low since the # of particle in bucket often becomes less than 32.

KNL and Skylake: Bucket per logical thread

Each target bucket (Loop1) is assigned to logical thread. Loop 2*4 are processed sequentially by each thread. OpenMP’s “schedule (dynamic)” clause is used for load balancing since the # of particles in bucket is different.

Memory access of threads is different and cache utilization is very low. Vectorization is not done as well. Different algorithm (e.g. Verlet list in molecular dynamics) is required to solve these problems. KNL and Skylake achieved higher performance when Hyper-Threading is enabled.

IV. Performance on P100, KNL and Skylake

<table>
<thead>
<tr>
<th>Single prec.</th>
<th>Clock (Bost)</th>
<th>Number of Threads</th>
<th>Mem BW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-KNL-7210</td>
<td>5.3</td>
<td>1.3 (1.5)</td>
<td>64 * 4HT = 256</td>
</tr>
<tr>
<td>Two-Xeon Gold 6150</td>
<td>N/A</td>
<td>2.7 (3.7)</td>
<td>18 * 2HT * 2CPUs = 72</td>
</tr>
<tr>
<td>Two-P100 (PCIe)</td>
<td>9.3</td>
<td>1.1 (1.3)</td>
<td>3,584 * 2GPUs = 7,168</td>
</tr>
<tr>
<td>Two-P100 (NVLink)</td>
<td>10.6</td>
<td>1.3 (1.4)</td>
<td>3,584 * 2GPUs = 7,168</td>
</tr>
</tbody>
</table>

Data sets
- Collapse of water column (40[cm] × 40[cm] × 8[cm])
- # of particles: 224,910, # of buckets: 70,707 × 14
- Average time of 200 time step of particle density computation

Compilers and configurations
- P100: PGI Compiler 17.7 w/ “-ta=nvidia:cc60,cuda8.0,fastmath”
- KNL7210 (Fiat+Quadrant, 4HT) and Xeon Gold 6150 (2HT): Intel Compiler 2017 w/ “-qopenmp -O3 -xCOMMON-AVX512 -fp-model-fast2”

Performance is different when host processor is different

Single KNL-7210 Xeon Gold 6150 Gold 6150+P100 PCIe POWER8+P100 ES-2695v4-P100 NVLink

Projected number of two-KNL (104.2[ms]) for single-KNL

P100 is x4.5 faster than Skylake