Parallelization and Optimization of Laser-Plasma-Interaction Simulation Based on Kepler Cluster

Haipeng Wu¹ Minhua Wen¹ Simon See² James Lin¹,³

¹ Shanghai Jiao Tong University
² NVIDIA
³ Tokyo Institute of Technology

HPC CHNIA 2016, Xi’an China
October 28, 2016
Outline

✓ Introduction

✓ Goal Proposal and Contribution

✓ Parallelization and Optimization
  ▪ GPU Parallelization
  ▪ Optimizations on GPU

✓ Empirical Evaluation

✓ Conclusion and Future Work
The theoretical and experimental research on laser-matter interaction is developing very quickly.

- Due to the progress in generating intense ultra-short laser pulse

The code we present is a newly developed, electromagnetic, relativistic Particle-in-Cell laser-plasma-interaction simulation code.
PIC (Particle-In-Cell) Method

- The PIC method has been widely used for simulation of laser-plasma interaction and other physics simulations.

- The quality of the results achieved with this method depending on involving a large number of particles.

- Due to the possible data hazards, irregular data accesses, achieving great parallel and architectural efficiency is an extremely challenging task.
Goal, Proposal and Contribution

- **Goal:** Parallel and Optimize the Laser-Plasma-Interaction code on GPU cluster.

- **Proposal:**
  - GPU Parallelization: merged function, thread dispatching and data structure transformation
  - Optimizations on GPU: Dynamic Duplication Algorithm, mix-precision computing and a parameterized particle sorting algorithm.
  - Optimizations for MPI: GPUDirect RDMA

- **Contribution:**
  - Developing a series of methods to speed up the initial GPU version
  - Evaluating using GPUDirect RDMA in GPU cluster
Outline

- Introduction
- Goal Proposal and Contribution
- Parallelization and Optimization
  - GPU Parallelization
  - Optimizations on GPU
- Empirical Evaluation
- Conclusion and Future Work
GPU Parallelization

- Data structure transformation

- Merged Function

- Thread dispatching strategy
  - We assign a GPU thread to each particle location in the global particle array
Optimizations on GPU

- Duplication Method

The writing scheme when writing to different grid cells

The writing scheme when conflicts happens within the same cell
What’s “Dynamic”

- Based on the fact that particles are not well distributed on the grid.

**Algorithm 3: DynamicDuplication**

1. Calculate the average number of particles each cell has, named A;
2. Calculate the max number of particles in a cell, named B;
3. For the cells that contains particles fewer than the average number A,
4. Make `Base_duplication_num` duplications of that cell, for those contains more particles than the average number A, Make `Base_duplication_num \times B / A` duplications of the cells.
Mixed Precision Method

- Compare & Set atomic implementation is slow for the double precision method.

- Do the data precision transformation work before and after the particle related data computation.

- Speed-up of 1.2x without apparent loss of accuracy.
Particle Sorting

\[
\begin{array}{cccccccccc}
1 & 1 & 2 & 4 & 5 & 5 & 5 & 5 & 7 & 7 \\
\end{array}
\]

Particle_array

\[
\begin{array}{cccccccccc}
1 & 1 & 2 & \text{Empty} & 5 & 12 & 5 & 5 & 7 & 1 \\
\end{array}
\]

After one step

\[
\begin{array}{cccccccccc}
1 & 1 & 1 & 2 & 5 & 5 & 5 & 7 & 12 \\
\end{array}
\]

After sorting

Tune this parameter

\[
\text{empty\_ratio} = \frac{\text{number\_of\_empty\_locations}}{\text{number\_of\_all\_locations}}
\]

- Trade off between the benefit of sorting and the overhead it brings.
CUDA-Aware MPI

- MVAPICH2
- OpenMPI
- CRAY
- SGI MPI
...

```c
//MPI rank 0
cudaMemcpy(s_buf_h,s_buf_d,size,cudaMemcpyDeviceToHost);
MPI_Send(s_buf_h,size,MPI_CHAR,1,100,MPI_COMM_WORLD);

//MPI rank 1
MPI_Recv(r_buf_h,size,MPI_CHAR,0,100,MPI_COMM_WORLD, &status);
cudaMemcpy(r_buf_d,r_buf_h,size,cudaMemcpyHostToDevice);
```

```c
//MPI rank 0
MPI_Send(s_buf_d,size,MPI_CHAR,1,100,MPI_COMM_WORLD);

//MPI rank n-1
MPI_Recv(r_buf_d,size,MPI_CHAR,0,100,MPI_COMM_WORLD, &status);
```
CUDA-Aware MPI + RDMA

- CUDA-Aware MPI can transparently use the GPU Direct.
- Buffers can be directly sent from the GPU memory to a network adapter without staging through host memory.
Outline

✓ Introduction
✓ Goal Proposal and Contribution
✓ Parallelization and Optimization
  ▪ GPU Parallelization
  ▪ Optimizations on GPU
✓ Empirical Evaluation
✓ Conclusion and Future Work
# Test Bed

## Experimental platform

<table>
<thead>
<tr>
<th>Component</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel(R) Xeon(R) CPU E5-2670 @ 2.60GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>62GB DDR3</td>
</tr>
<tr>
<td>GPU</td>
<td>NVIDIA Kepler K20</td>
</tr>
<tr>
<td>Compiler</td>
<td>NVCC 7.0 + Open MPI 1.10 + GCC 4.9.1</td>
</tr>
</tbody>
</table>
Empirical Evaluation

Baseline: initial GPU version
Combined: 3.5x
RDMA

Non-RDMA: MPI communication time + Gather Data + Scatter Data
Conclusion

- A way to implement the whole PIC Laser-Plasma-Interaction Simulation code onto GPU.

- Novel methods to accelerate the initial GPU version

- Evaluate the use of GPUDirect RDMA technique
Future Work

- Utilize GPU shared memory to store the Grid related data combined with a different GPU thread dispatching strategy.