PAC Final Report (1)

2014-10-08

Our Topic

- Optimize this *scientific computing software*: **Gromacs**
  - **High-level:** *Simulator* for molecular dynamics problems
  - **Low-level:** With **SIMD, MPI parallel optimization**

Molecular Dynamics Simulation

- It’s a simulation of the *time evolution of an atomic system*.
- There’re 3 key points:

    1. There is a **initial state** of that system before we calculate, each molecular has its own *mass*, *acceleration* and *position*

2. The way to evolution is decided by the following equations:

    1. **Newton’s second law** (*mass* and *acceleration*)

    2. System’s **potential energy equation** (*position*)

    3. The most time-consuming part is calculating the **interaction forces**.

Algorithm Level Optimization

- Initially, the brute force algorithm should be O(N^2). (N is the number of particles.
- *Neighbor List* Method:
  - **Cutoff:** Particles out of the *neighbor sphere* with radius value **r**.
  - **O(N)**: Nearly **constant** number of particles inside the sphere.
  - **Update**: We need to update the neighbor list after each calculation time.

Basic Procedures

(1) Initialize the atoms' status and the parameters of controlling the simulation

(2) Start for:

(1) If there's an atom move too much(out of boundary), then do *move to much*, then *update the neighbor list(O(N^2))*.

(2) Use neighbor list to compute the forces. *Force calculation*

(3) *Update velocities*

(4) *Accumulate target statistics* of each time step.

Architecture Level Optimization

- We are focusing on **parallel computer architecture**:
  - *coarse-grained* architecture: PC clusters and computational grids
  - *fine-grained* architecture: special-purpose architectures, reconfigurable architectures, and GPUs

Architecture-dependent Methods

- From the view of *data decomposition*, we could categorize the methods into 3 types:

    1. **Atom-decomposition:** Each processor is assigned *N/P* atoms
        - Especially *shared memory architecture*.
    2. **Force-decomposition:** Divide the workloads of force calculating
        - Less memory cost but not easy to be load-balanced.
    3. **Spatial-decomposition:** Divide based on position and space
        - More suitable on *coarse-grained* architecture. (?)

- Which method should we choose?

What is Gromacs? (1)

- Features:
  - Gromacs likes a platform, rather than a single MD algorithm
  - It supports many parallel computing optimizations
- History:
  - 1995: *GROMACS: A message-passing parallel molecular dynamics implementation*
  - 2008: *GROMACS 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation*
  - 2013: *GROMACS 4.5: a high-throughput and highly parallel open source molecular simulation toolkit*

What’s Gromacs (2)

- Based on the previous historical description, Gromacs optimization methods are:
  - Derived from the *cluster architecture*
  - *Message-passing* model is the key feature.
  - Now supports more on the *fine-grained architectures*.

Gromacs on Xeon Phi

- We need to figure out these 3 questions:
  - The features of *Xeon Phi architecture*.
  - How to map the *optimization methods* upon Xeon Phi?
  - What are the differences among *Xeon Phi, GPU* and *cluster*.

Xeon Phi Architecture

- 3 key words:
  - **Multicores**: Around 60 computing units on the Xeon Phi computing card. Each could support 4 hardware threads at most.
  - **Vectorization**: Each core has a powerful VPU with 512bits length operands.
  - **Communication:** Xeon Phi card has its own OS and unique ip address, and there are 2 modes: *native* and *offload*.

Xeon Phi Parallel Programming

- Basically, we have 2 parallel programming models on Xeon Phi:
  - **SIMD:** Supported by VPUs
  - **MIMD**: Supported by hardware threads and multicores architecture.
- All these models could be implemented in **OpenMP.**

How to Optimize?

- All these methods will work on single mic card.
  - *Vectorization*: auto-vectorization, manually.
  - *Hyper-threads*: auto-parallelization, manually.
  - *Memory*: alignment, improve cache locality, use 2MB page, streaming store.
- Multiple MIC cards? How to cooperate with each other?

Xeon Phi Message Passing

- Xeon Phi supports Intel MPI library initially:
  - Each Xeon Phi card has its own OS and ip address, so we could view them as *independent machines*.
  - Xeon + Xeon Phi could become a *heterogeneous* system.

MPI Introduction

- **MPI** is *Message Passing Interfaces*. There are many vendors of this library.
- It could cooperate with C/C++, by including mpi.h and using the provided functions and types.
- The working model is simple:
  - All the processors could receive and send data.
  - One processor manipulates all the transfer issues.

MPI Program

- **Initialization**
  - **MPI_Init** starts MPI
  - **MPI_Finalize** exits MPI
  - procedures inside this block will be executed *locally*.
- **2 Questions to Ask**
  - How many processors are there? **MPI_Comm_size**
  - Who am I? **MPI_Comm_rank**

MPI Program (2)

```
#include "mpi.h"
#include <stdio.h>

int main( argc, argv )
int argc;
char **argv;
{
int rank, size;
MPI_Init( &argc, &argv );
MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
printf( "Hello world! I'm %d of %d\n",
        rank, size );
MPI_Finalize();
return 0;
}
```

MPI on Xeon Phi

- Two modes:
  - *Native*: send messages among all the processors and coprocessors
  - *Offload*: only use MPI on Xeon processors and use *offload* to communicate with Xeon Phi card.
- Need to be considered:
  - *Load balance*: Xeon and Xeon Phi fit different works.
  - *Communication Cost*: *Xeon*<->*Xeon* and *Xeon*<->*Xeon Phi*

Road Map

- First of all, be an expert on Xeon Phi parallel programming and MPI programming.
- Secondly, understand not only the advanced MD algorithm and its optimization, but also the project structure of Gromacs.
- Finally, write code and test.

Work Division

- There are 4 kinds of works to do:

  (1) *Gromacs Operator*: This man should know the code structure of Gromacs, and understand how to run the benchmark, what the benchmark is about, where's the bottleneck, and where to insert our revised code. Also, he should do the profiling work.

  (2) *Xeon Phi Programmer*: This man should understand how to write code on Xeon Phi and how to improve the performance.

  (3) *MPI Programmer*: This man should know how to write MPI code, and design a robust communication scheme among Xeon and Xeon Phi cards. Should also understand the way Gromacs used to transfer data and divide workload.

  (4) *Algorithm Designer*: This man should know most of the MD algorithms, their parallel optimization methods, and what the Gromacs is using.