Large-scale Simulations of Peridynamics on Sunway TaihuLight Supercomputer

Authors: Xinyuan Li, Huang Ye, Jian Zhang

Reporter: Xinyuan Li Computer Network and Information Center, Chinese Academy of Science lixy@sccas.cn

Outline

- Introduction
- Optimizations
    - Memory Access
    - Vectorization
    - Communication
- Performance Evaluation
- Conclusion
- Future work

Introduction

- Peridynamics models
- Sunway TaihuLight
- Challenges to implement PD applications on Sunway TaihuLight

Peridynamics Models

- Peridynamics (PD) is a non-local mechanics theory proposed by Stewart Silling in 2000. The models are built based on the idea of non-local behaviors and describe the mechanical behaviors of solids by solving spatial integral equations.

- Because of the superiority on simulating the discontinuous problems, in recent years, the PD methods have been widely used in material science, human health, electromechanics , and disaster prediction , etc.

Peridynamics Models

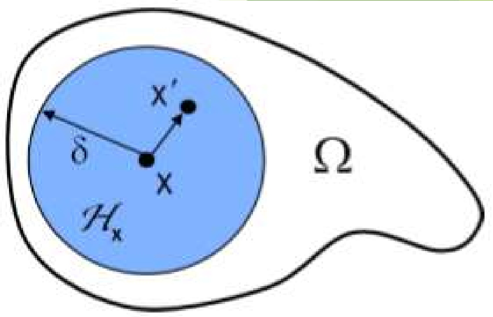
- The simulation processes update the state of points by solving the mechanical equilibrium equation.

- The strong form of the equation is as follows

$$\rho(\boldsymbol{x})\ddot{u}(\boldsymbol{x},t) = \int_{Hx} \left(\underline{T}[\boldsymbol{x},t](\boldsymbol{q}-\boldsymbol{x}) - \underline{T}[\boldsymbol{q},t](\boldsymbol{x}-\boldsymbol{q})\right)dV_q + b(\boldsymbol{x},t)$$

- Its discrete equation is as follows

$$\rho(\boldsymbol{x})\ddot{u}(\boldsymbol{x},t) = \sum_{Hx} (\underline{T}[\boldsymbol{x},t](\boldsymbol{q}-\boldsymbol{x}) - \underline{T}[\boldsymbol{q},t](\boldsymbol{x}-\boldsymbol{q}))dV_q + b(\boldsymbol{x},t)$$

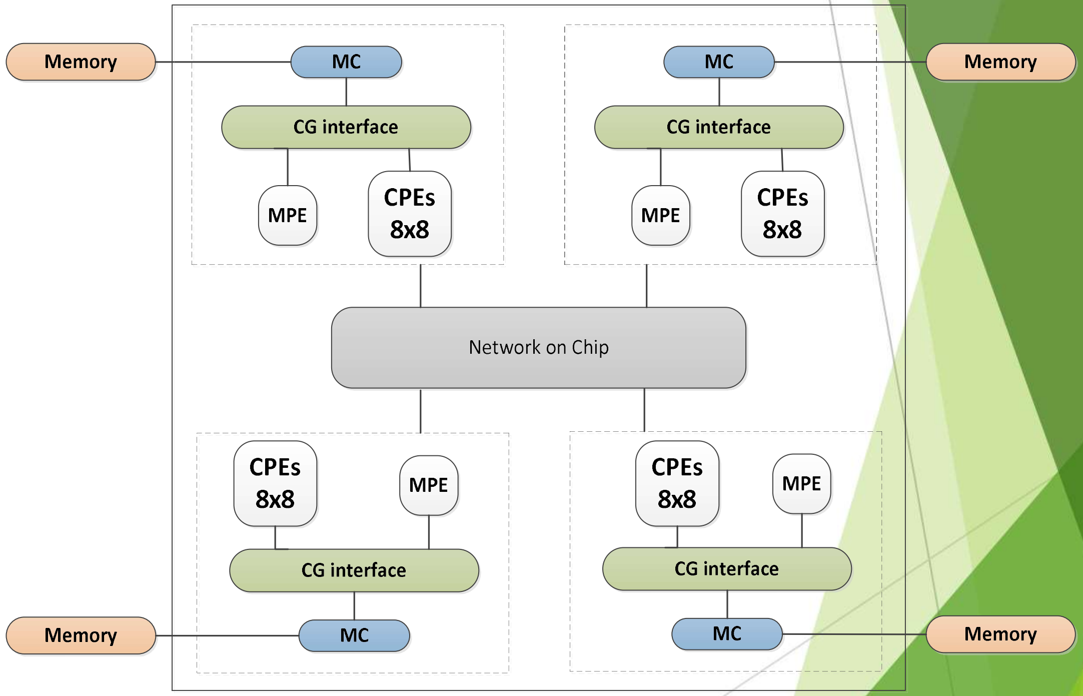


- In order to describe the crack initiation, propagation until the failure, the concept of local damage of the material point is introduced.

$$D(x,t) = \frac{\sum_{Hx}(1-\varphi(x,q,t))dV_q}{\sum_{Hx} dV_q}, \text{ where } \varphi(x,q,t) = \begin{cases} 1\ (s \le s_0) \\ 0\ (s > s_0) \end{cases}$$

Sunway TaihuLight

- Sunway TaihuLight consists of 40,960 SW26010 processors.
- A processor consists of four core groups (CGs), each including one Management Processing Element (MPE) and 64 Computing Processor Elements (CPEs).
- DMA is used by CPEs to exchange data with MPE in the same core group
- There are two instruction pipelines in each CPE, which enables overlapping between memory access instructions and computation instructions



SW26010 processor

Challenges to implement PD applications on Sunway TaihuLight

- DMA requires the data block more than 128 bytes to make data transaction between CPEs and MPE efficient, so the data organization should be adjusted.
- Bandwidth between the CPE and MPE is relatively low compare to the compute ability, which will make the simulation become memory-bound.
- Data dependencies and high-latency instructions in bond-based part affect the throughput of instruction pipelines.
- The cost of communication between processes may be obvious when facing large-scale simulations

Optimizations

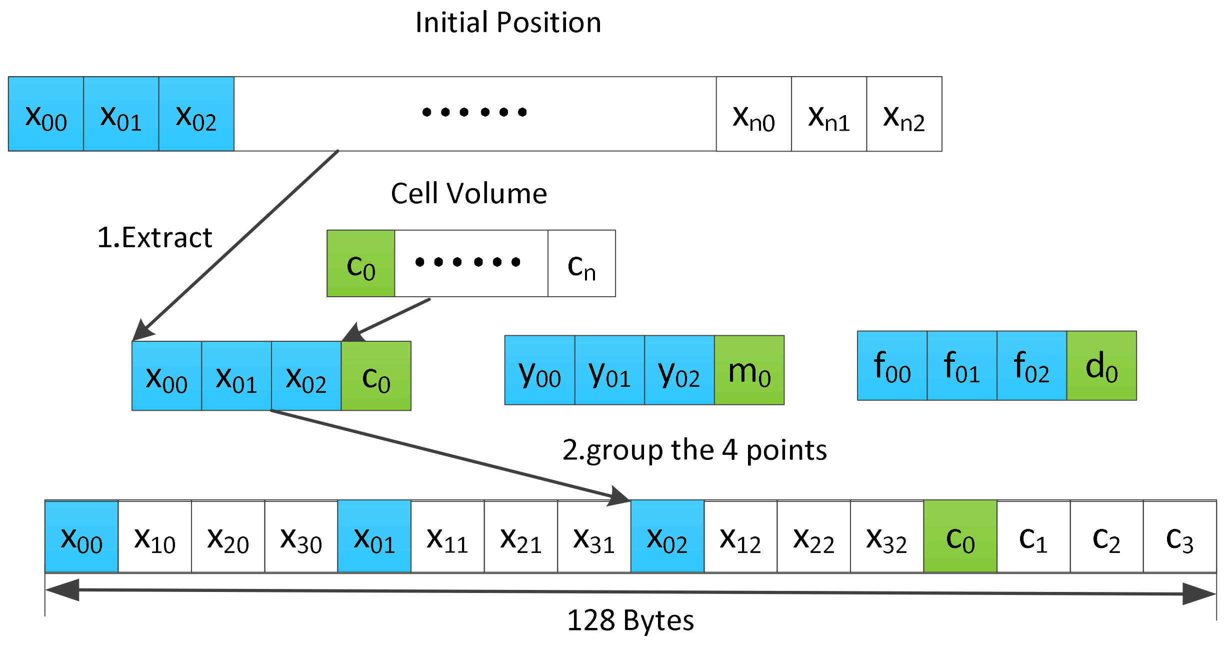
- Memory Access
- Vectorization
- Communication

Memory Access

- Data grouping for DMA
- SPM-based cache

Data grouping for DMA

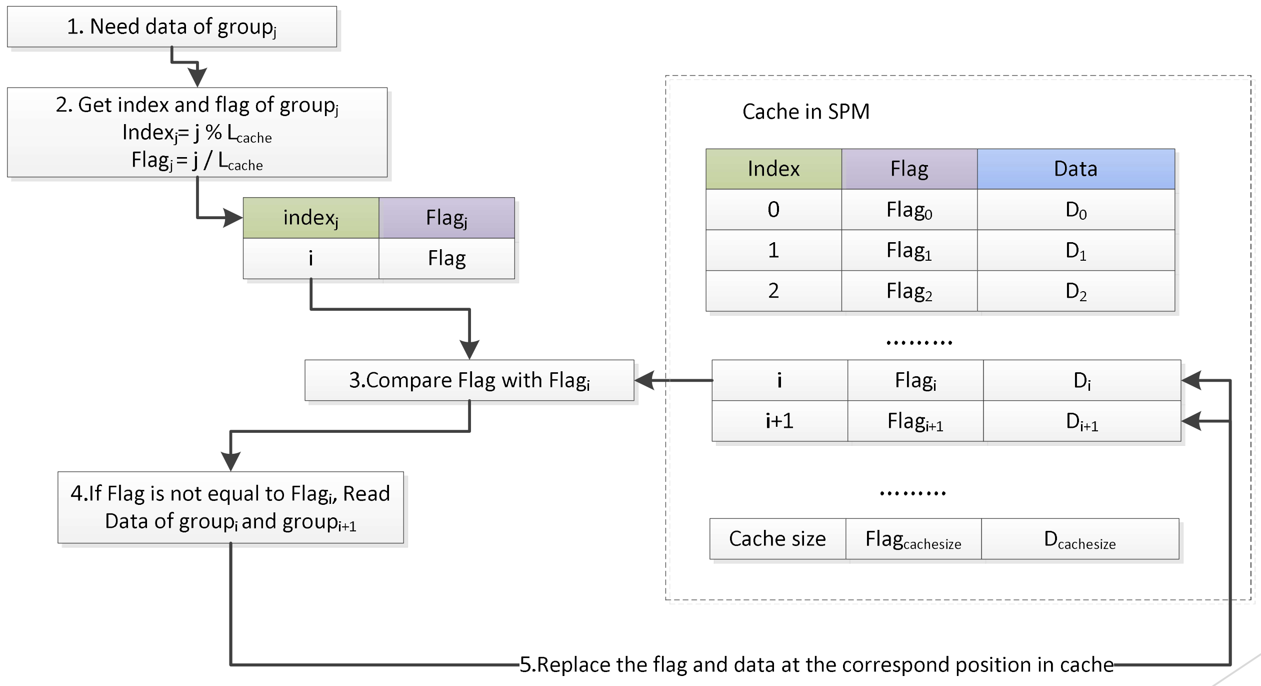
- In the PD simulation, each point consists of six items: ***x***, ***y***, ***f***, *m*, *c*, and *d*. These data are grouped based on the data dependencies of algorithms. For examples x and c are always needed together.



Initial Position
x_{00} x_{01} x_{02} ······· x_{n0} x_{n1} x_{n2}
Cell Volume
c_0 ······· c_n
1.Extract
x_{00} x_{01} x_{02} c_0
y_{00} y_{01} y_{02} m_0
f_{00} f_{01} f_{02} d_0
2.group the 4 points
x_{00} x_{10} x_{20} x_{30} x_{01} x_{11} x_{21} x_{31} x_{02} x_{12} x_{22} x_{32} c_0 c_1 c_2 c_3
128 Bytes

SPM-based cache strategy

- Each CPE has a 64 KB scratchpad memory(SPM).
- In the PD simulation, during the calculation between bonds, each point is accessed multiple times by CPEs. Performance can be improved if CPEs can read most of the required data from the SPM.

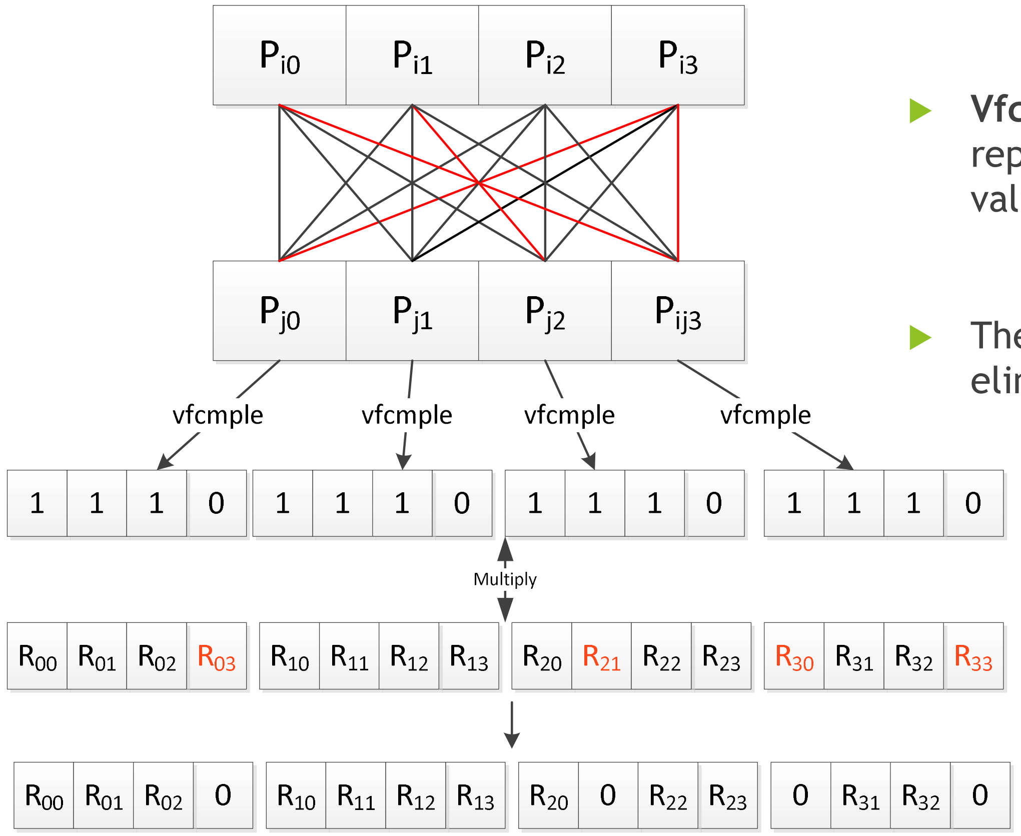


1. Need data of group$_j$
2. Get index and flag of group$_j$ Index$_j$= j % L$_{cache}$ Flag$_j$ = j / L$_{cache}$
index$_j$ | Flag$_j$
i | Flag
3.Compare Flag with Flag$_i$
4.If Flag is not equal to Flag$_i$, Read Data of group$_i$ and group$_{i+1}$
5.Replace the flag and data at the correspond position in cache
Cache in SPM
Index | Flag | Data
0 | Flag$_0$ | D$_0$
1 | Flag$_1$ | D$_1$
2 | Flag$_2$ | D$_2$
....
i | Flag$_i$ | D$_i$
i+1 | Flag$_{i+1}$ | D$_{i+1}$
....
Cache size | Flag$_{cachesize}$ | D$_{cachesize}$

Vectorization

- Error-fixed vectorization for kernel functions
- Optimized instruction scheduling
- Vectorized bond damage flag operations

Error-fixed vectorization for kernel functions



P_{i0} P_{i1} P_{i2} P_{i3}
P_{j0} P_{j1} P_{j2} P_{ij3}
vfcmple
vfcmple
vfcmple
vfcmple
1 1 1 0
1 1 1 0
1 1 1 0
1 1 1 0
Multiply
R_{00} R_{01} R_{02} R_{03}
R_{10} R_{11} R_{12} R_{13}
R_{20} R_{21} R_{22} R_{23}
R_{30} R_{31} R_{32} R_{33}
R_{00} R_{01} R_{02} 0
R_{10} R_{11} R_{12} R_{13}
R_{20} 0 R_{22} R_{23}
0 R_{31} R_{32} 0

- There are invalid bonds which will be calculated between two groups.
- **Vfcmple** is used to get the flag represents whether the interaction is valid.
- The affect of invalid bonds can be eliminated by multiplying the flag.

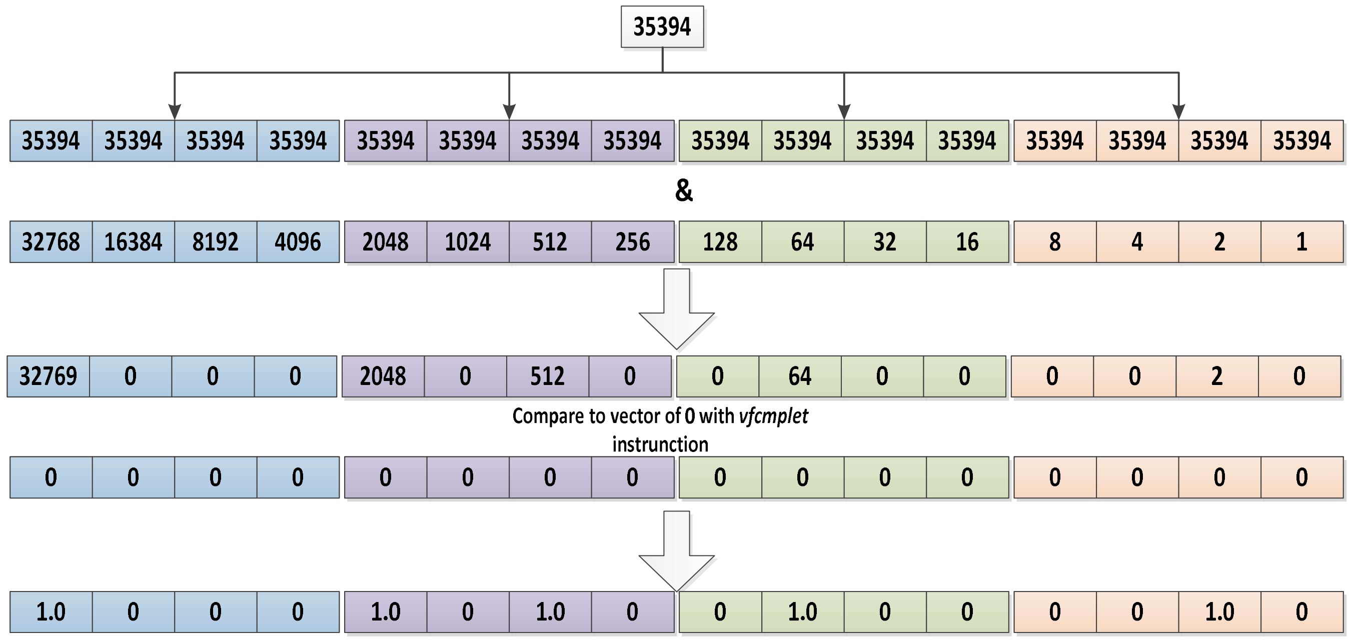
Optimized instruction scheduling

- To fully utilize the instruction pipelines, we need optimize the scheduling manually.

- The throughput can be improved from 3 aspects
  - Reduce the data dependencies between instructions by inserting independent instructions between two dependent instructions.
    - Unroll the loop because the calculations of the bond is independent except the reduction step.
    - Reorder the instruction sequence can further reduce the dependencies
  - Overlapping the memory access instructions with floating-point instructions
    - The computation instructions are much more than the memory access instructions
  - Reduce the high-latency instructions
    - Refine the calculation algorithms, e.g. replacing the division with multiplying the reciprocal of the divisor
    - Replace the high-latency instructions (i.e. sqrt, div, rsqrt) with software-implement versions

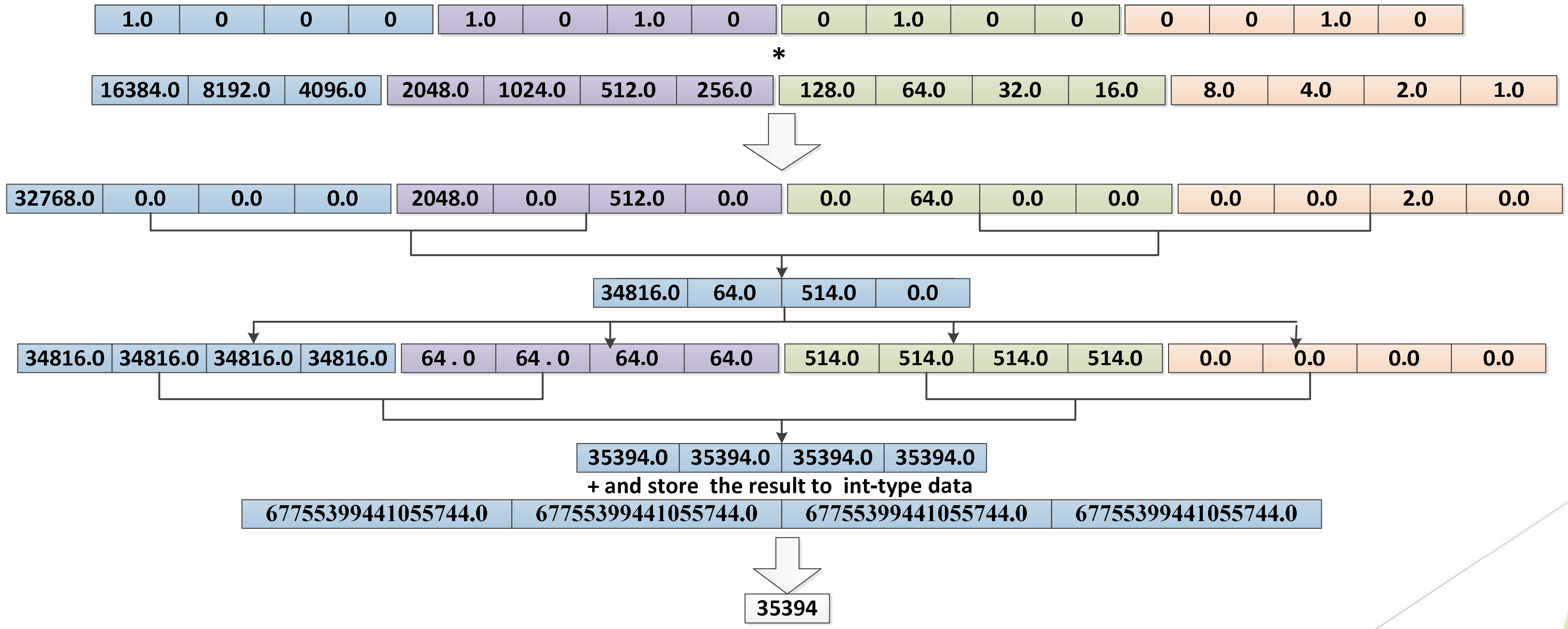
Vectorized bond damage flag operations

- Vectorization of decompression of bond damage flags
  - The binary code of 35394 is 1000101001000010



Vectorized bond damage flag operations

- Vectorization of compression of bond damage flags

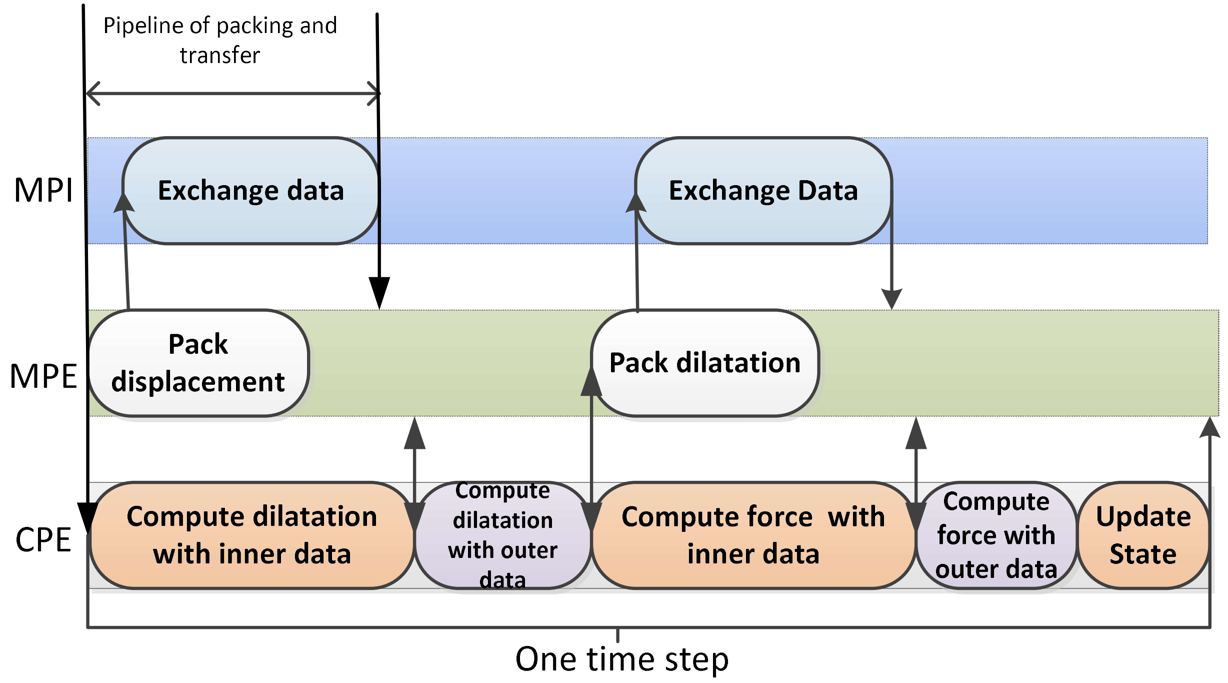


Overlapping strategies

- Process-level overlapping strategy
- Double buffer-based overlapping strategy

Process-level overlapping strategy

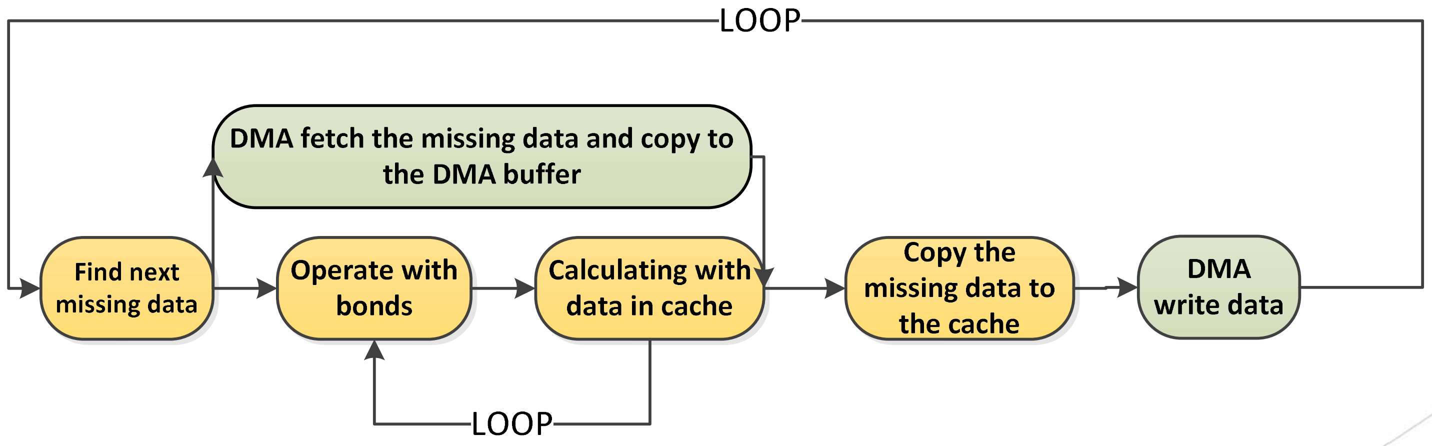
- Overlapping happens between:
  - Data exchange and Data packing
  - Tasks on MPEs and tasks on CPEs



Pipeline of packing and transfer
MPI
Exchange data
Exchange Data
MPE
Pack displacement
Pack dilatation
CPE
Compute dilatation with inner data
Compute dilatation with outer data
Compute force with inner data
Compute force with outer data
Update State
One time step

Double buffer-based overlapping strategy

- In order to achieve the overlapping between calculation and DMA, besides the SPM-based cache on CPE, we set an additional buffer (namely DMA buffer) to store DMA data for next calculation.
- Cache and DMA buffer form double buffers on CPE.



LOOP
DMA fetch the missing data and copy to the DMA buffer
Find next missing data
Operate with bonds
Calculating with data in cache
Copy the missing data to the cache
DMA write data
LOOP

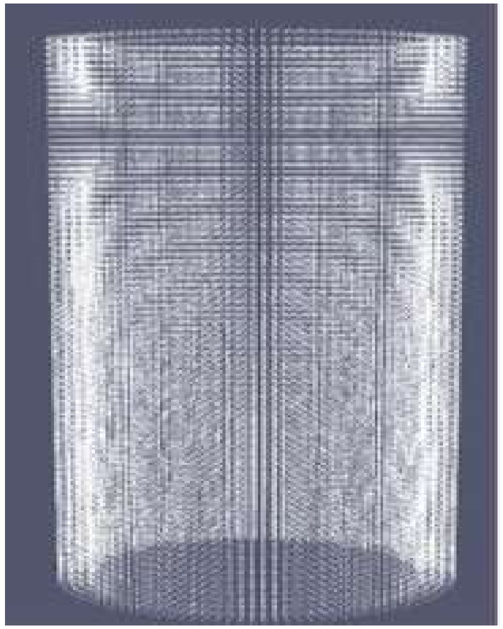
Performance Evaluation

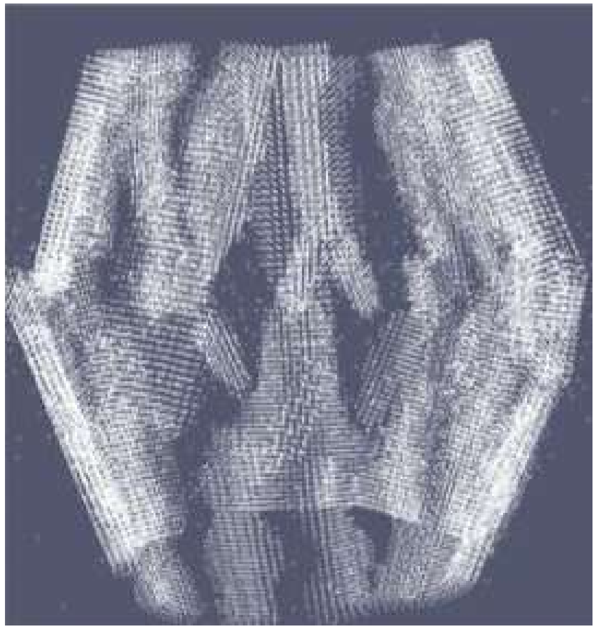
Experiment Setup

- The test cases are taken from the examples of Peridigm, which simulates the fragment process of a cylinder.
- All test cases are generated by a generator provided by Peridigm.

Material	Elastic
Density	7800.0 kg/m^3
Bulk Modulus	130*e^9 Pa
Shear Modulus	78*e^9 Pa
Critical elongation	0.02
Horizon	0.00417462 m
Timestep	0.26407 us
Start time - End time	0.0 us – 250 us

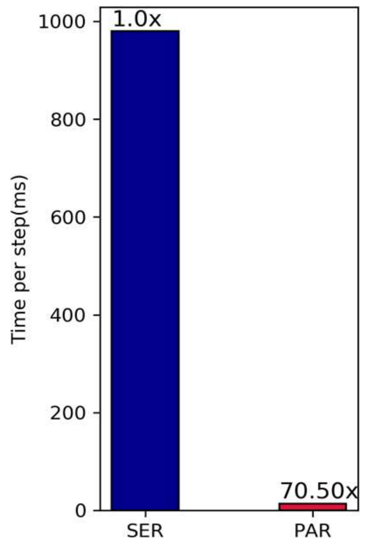
Experiment Setup

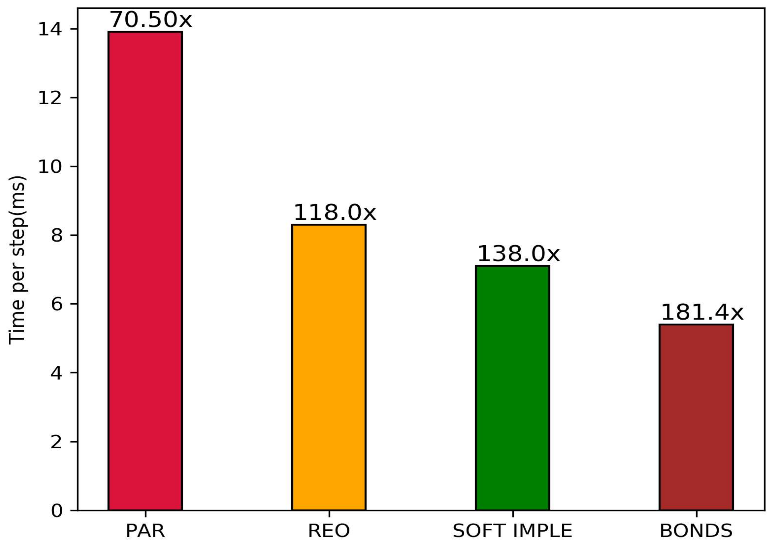




Single Core Group Evaluation

- A speedup of 181.4 times is achieved compared to serial version run on MPE when run with the example with 36160 points.





SER: serial version run on 1 MPE
PAR: parallel version with memory access and overlapping optimizations
REO: PAR + instruction reordering
SOFT IMPLE: REO + software implement instructions
BONDS: Adopt all optimizations

Single Core Group Evaluation

- During the compression of bond damage flags, the data dependencies are more serious than the decompression, which causes that the acceleration for compression is not as good as decompression.

Time taken by each part in a timestep of PAR version(msec)

Function	Total	bonds	kernel	DMA	other
(#5)Dilatation	5.636	0.931	4.049	0.502	0.487
(#6)Force	8.097	0.529	6.828	0.611	0.355
Update	0.332	\	0.012	0.329	0.003

Time taken by each part in a timestep of BONDS version (msec)

Function	total	bonds	kernel	DMA	other
(#5)Dilatation	2.434	0.489	1.361	0.502	0.487
(#6)Force	2.874	0.160	2.069	0.611	0.355
Update	0.332	\	0.012	0.329	0.003

Single Core Group Evaluation

Performance of four examples

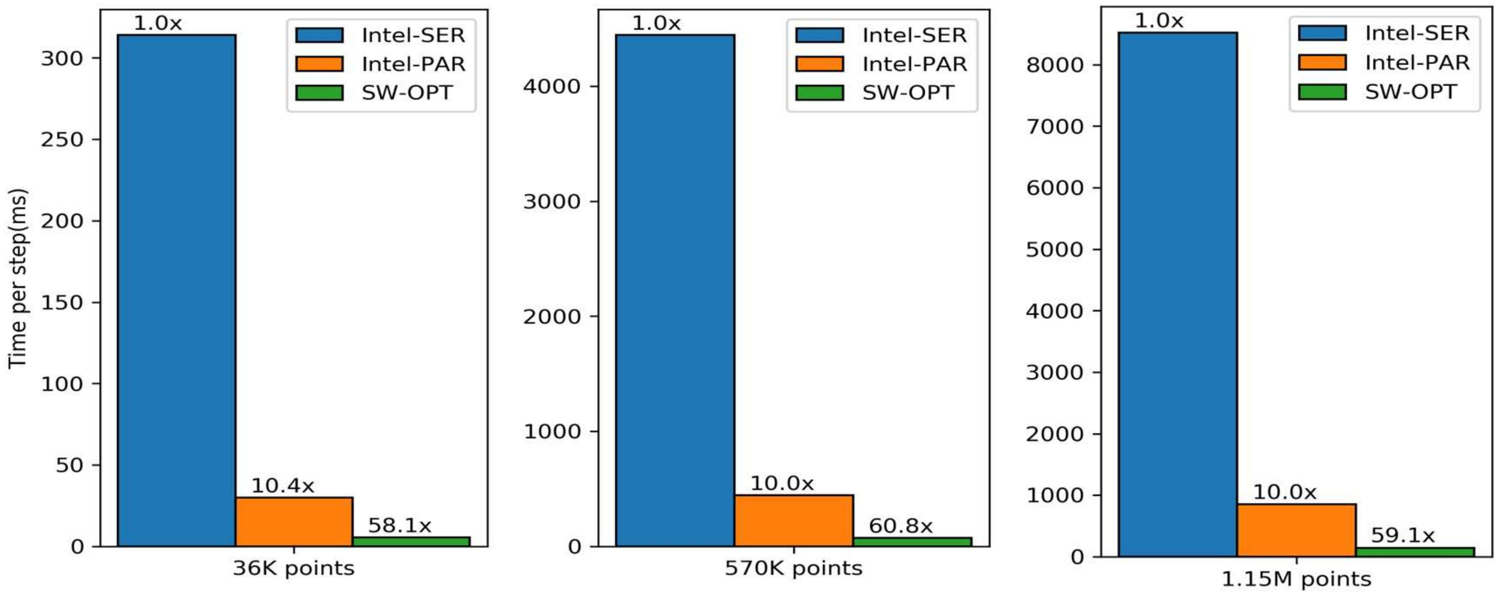
Points number	Bonds/ point	Cache hit ratio (%)	Time/step (msec)	Performance ratio (%)
36160	145	90.31	5.7	18.73
87000	367	96.22	30.06	21.62
141000	332	95.01	44.46	21.43
144640	147	90.47	21.61	20.17

Single Core Group Evaluation

- We compare our application with Peridigm.
- Considering the differences in computing power, we choose a computing environment with similar power consumption for the application.
  - Intel Xeon E5-2680 V3: 120W
  - A core group of SW26010: 94W

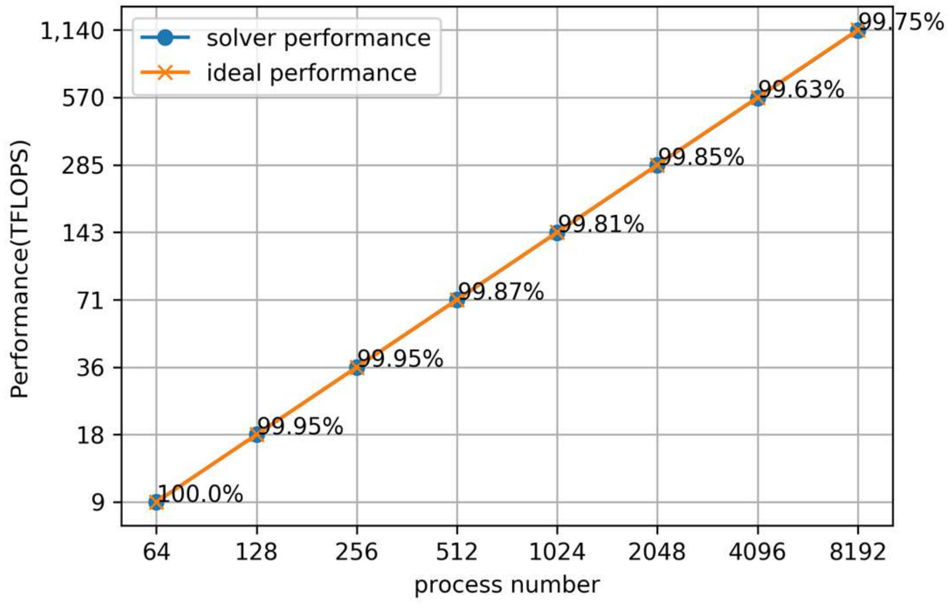
Evaluation	Platform	Scale	Frequency	Memory
Intel-SER	Xeon E5-2680 V3	1 process	2.5GHz	8G
Intel-PAR	Xeon E5-2680 V3	1 CPU	2.5GHz	8G
SW-OPT	SW26010	1 group	1.45GHz	8G

Single Core Group Evaluation



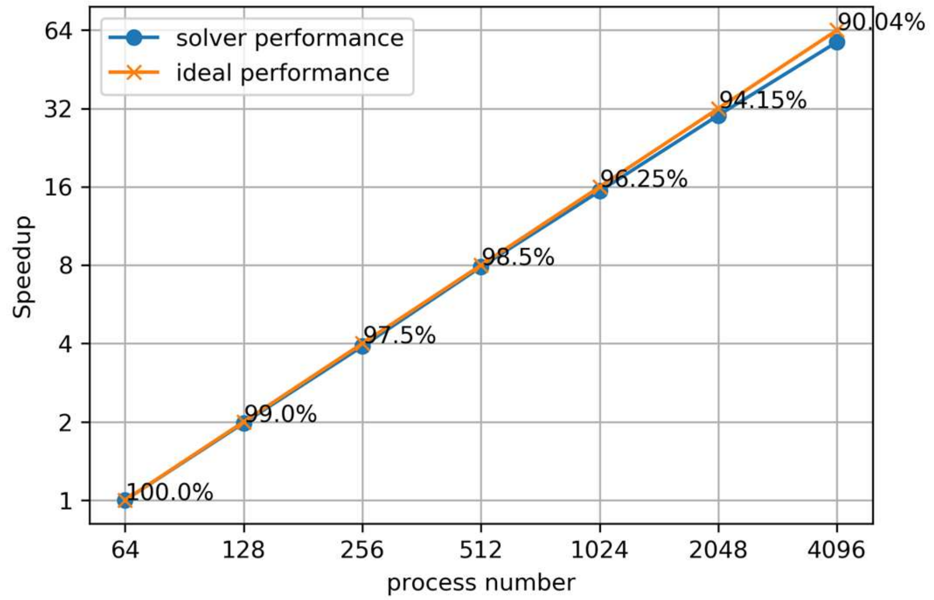
Scalability Evaluation

- For the weak scaling test:
  - The size of points assigned to each process stays constant (i.e., 36160) and the number of processes is from 64 to 8192.
- The results show that the parallel efficiency is almost ideal.



Scalability Evaluation

- For the strong scaling test:
    - we fix the problem size to 148,111,360 points and execute the example using different number of processes (64-4096).
- The parallel efficiency is over 90% when the number of processes scales 64 times.
- The parallel efficiency decreases because of the cache misses happen at the early stage of the simulation. The smaller the problem size per process is, the higher the proportion of time it takes to read data through DMA at the beginning is.



Conclusion

- Our optimization techniques greatly improve the efficiency of the large-scale PD simulation and provide an efficient application on the Sunway TaihuLight.
- Our work can offer insight into similar applications on other heterogeneous manycore platforms.

Future work

- Do larger-scale PD simulations
- Transplant Peridigm to Sunway TaihuLight
- Implement efficient PD simulation software on the GPU cluster

Thanks!

Looking forward to your questions!