S-B1-2

Local View Kernels: A New Programming Scheme for Plasma Simulation

Local View Kernels:プラズマシミュレーションの新たな記法

Hiroshi Nakashima
中島　浩

ACCMS, Kyoto University
Yoshida Hon-machi, Sakyo-ku, Kyoto 606-8501, Japan
京都大学 学術情報メディアセンター 〒606-8501　京都市左京区吉田本町

Plasma simulations with PIC need various implementation techniques to run them efficiently on modern supercomputers having up to many hundreds thousands of CPU cores and occasionally with accelerators such as GPGPU. Unfortunately, these techniques often architecture dependent and thus your codes must be revised, drastically sometimes, each time you have a new supercomputer. Our research on *local view kernels* aims at making you free from these tough and repetitive efforts providing a framework with which you may concentrate on a particle and a grid point describing *what*-part of your simulation as a set of local view kernels, while *how*-part representing the implementation techniques are automatically attached to the loops containing the kernels.

1. Introduction

In plasma simulation with PIC method, a huge number of charged particles interact with electromagnetic fields mapped onto a large number of grid points, governed by Maxwell's equations and the Lorentz force law. These hugeness and largeness of the simulation essentially require highly efficient implementations especially exploiting large-scale parallelism in modern supercomputers. Therefore, many researches have been conducted for efficient parallelization including our OhHelp [1] being the first scalable domain-decomposed parallelization with dynamic load balancing.

However, such a method is not always easily applicable to *your* simulation nor sufficiently improves its performance on a given supercomputer due to the following reasons. First, even with a well-designed library implementing the method, your code should be modified somewhat for the application. For example, though OhHelp provides highly sophisticated mechanism not only for load balancing but also inter-subdomain communications, you have to add ten or so library function (procedure) calls and reconfigure the fundamental time-evolutional loop in the form OhHelp requires.

Second, OhHelp takes care of parallel efficiency but does not concern about sequential performance. Therefore, it is perfectly up to *you* to make your code, e.g., cache-aware for efficient execution on your scalar MPP or cluster with further modification. Note that even if OhHelp and/or other libraries/ tools provide such a means for performance improvement, it is still up to you to modify your code so that the means works on the code together with OhHelped parallelization.

Third and finally, the efficiency and effectiveness of these methods often depends on the architecture of your supercomputer. This means that when you or your supercomputer center purchases a new system, your code might be modified to cope with the architectural change. This modification can be not only adding new techniques but also eliminating old ones as you should have done for vector-oriented code whose structure brings poor performance in executions on scalar parallel systems. Furthermore, a part for still-applicable techniques could have to be modified due to the change of architectural parameters such as cache size and memory/network bandwidth.

Our research aims at eliminating the necessity of these modifications of your code by making it described as the set of *local view kernels* acting on a small set of data elements, i.e., such as a particle and/or electromagnetic field vectors defined on a grid point and its neighbors. The kernels for *what*-type descriptions are then assembled together with *how*-type library calls and assembling methods such as cache-aware loop configurations.

2. Local View Kernels

Figure 1 shows a typical code structure of the main time-evolution loop of fundamental PIC simulators. The loop has calls of four kernel functions each of which has a loop (or a nest of loops) to scan all particles or all grid points on which electromagnetic field vectors are defined. That is, the first two scans particles to accelerate particles by Lorentz force (`particle_push()`) and then to scatter current caused by the movements of particles (`current_scatter()`), while the other two solve the

```
for (t=0; t<T; t++) {
  particle_push(...);
  current_scatter(...);
  field_solve_e(...);
  filed_solve_b(...);
}
```

Fig.1. Typical main loop of PIC code

progress of electric field vectors (`field_solve_e()`) and magnetic ones (`field_solve_b()`).

The loop above and four functions are simple but you have to make various modifications on it for efficient execution on your parallel supercomputer. For example, in order apply OhHelp to the code, you must duplicate four kernel function calls for the *secondary* subdomain and particles in it which a MPI process acts on for load balancing in addition to its *primary* ones, add an all-reduce communication to accumulate the current resulted from all particles in a subdomain, add two neighboring communications to exchange current and electromagnetic field vectors, and add a call of OhHelp load balancer to transfer particles crossing subdomain boundaries possibly with dynamic load rebalancing, to have the code shown in Fig. 2.

The code in Fig.2, however, is not very efficient in terms of its sequential performance, because it scans a huge one-dimensional array of particles thrice and then does it for a large three-dimensional array of electromagnetic field vectors thrice too resulting in a poor temporal locality in memory accesses. Therefore, you will have to modify the code further so that, for example, multiple scanning loops are *fused* and/or the spatial loops are *tiled*.

In order to avoid the modifications above which are often complicated and involve techniques themselves applicable to many PIC codes, we introduced *local view kernels* acting on each particle or each electromagnetic field vector as shown in Fig. 3. The code description is based on the domain-specific language *Physis*[3] developed for GPGPU-enabled coding for stencil computing but is extended to

```
for (t=0; t<T; t++) {
  particle_push(...);
  if(sec) particle_push(...);
  current_scatter(...);
  if(sec) current_scatter(...);
  if(sec) oh_allreduce_field(...);
  oh_exchange_borders(...);
  field_solve_e(...);
  filed_solve_b(...);
  oh_exchange_borders(...);
  oh_transbound(...);
}
```

Fig.2. Modified main loop

```
for (t=0; t<T; t++) {
  StencilMap(particle_push(...));
  StencilMap(current_scatter(...));
  StencilMap(field_solve_e(...));
  StencilMap(filed_solve_b(...));
}
stencil void particle_push(
  Part p, whole Vec b, whole Vec e){
  double a[3];
  lorentz(p,b,e,a);
  p[0].p[X]+=(p[0].v[X]+=a[X]);
  p[0].p[Y]+=(p[0].v[Y]+=a[Y]);
  p[0].p[Z]+=(p[0].v[Z]+=a[Z]);
}
```

Fig.3. Main loop with local view kernels

make it applicable to wider-range of simulation codes.

As shown in the figure, the Physis-based code has a structure similar to that shown in Fig. 1, but the kernel functions are for a particle or a vector. Therefore, the construction of loops scanning particles and vectors are up to our code translator which is also responsible not only of domain-decomposed MPI parallelization with OhHelp but also of thread-level parallelization with OpenMP, loop fusion and tiling for efficient memory access with for good temporal locality, a sophisticated inter-process communication for the overlapping computation and communication with improvement of spatial locality in neighboring communication, and so on.

For example, our preliminary investigation of the code automatically translated from that with local view kernels does not only show performance as good as hand-made OhHelped code but also exerts 40% better performance with threading and loop fusion. These implementation techniques are made applicable because the Physis-based description gives a free-hand to our translator in the construction of loops and the translator is domain-specific and thus have knowledge specific to PIC codes such as the hidden dependency between arrays.

Acknowledgments

The author would like to show his sincere appreciation to Dr. Naoya Maruyama and Dr. Tasuku Hiraishi for their contributions to our work.

References

[1] H. Nakashima, Y. Miyake, H. Usui and Y. Omura: *Proc. 23rd Intl. Conf. Supercomputing, Ossling, NY, 2009*, p. 90.

[2] H. Nakashima: *7th Intl. Cong. Industrial & Applied Math, Vancouver, 2011.*

[3] N. Maruyama, T. Nomura, K. Sato and S. Matsuoka: *Proc. SC11, Seattle, WA, 2011*.