Component-based Parallel Programming for Peta-scale Particle Simulations

Cao Xiaolin, Mo Zeyao and Zhang Aiqing
Institute of Applied Physics and Computational Mathematics, No. 2, East Fenghao Road, Beijing, China

Keywords: Parallel Programming, Parallel Integrator Component, JASMIN Infrastructure, Particle Simulation.

Abstract: A major parallel programming challenge in scientific computing is to hide parallel computing details of data distribution and communication. Component-based approaches are often used in practice to encapsulate these computer science details and shield them from domain experts. In this paper, we present our component-based parallel programming approach for large-scale particle simulations. Our approach encapsulates parallel computing details in parallel integrator components on top of a patch-based data structure in JASMIN infrastructure. It enables domain programmers to "think parallel, write sequential". They only need to assemble necessary components and write serial numerical kernels on a patch invoked by components. Using this approach, two real application programs have been developed to support the peta-scale simulations with billons of particles on tens of thousands of processor cores.

1 INTRODUCTION

The complexity of application systems and that of supercomputer architectures are providing a great challenge for parallel programming in the field of scientific computing. Parallel software infra-structures are the new tendency toward solving such challenges (Post and Votta, 2005). These infrastructures are intrinsically different to traditional libraries because they provide data structures and parallel programming interfaces to shield the details of parallel computing from the users. Based on software infrastructures, a user can easily develop parallel programs for complex computers.

J Adaptive Structured Meshes applications Infrastructure (JASMIN) is a parallel software infrastructure oriented to simplify the development of parallel software for multi-physics peta-scale simulations on multi-block or adaptive structured meshes (Mo and Zhang, 2010). Patch-based data structures, efficient communication algorithms, robust load balancing strategies, scalable parallel algorithms, object-oriented parallel programming models are designed and integrated. Tens of codes have been developed using JASMIN and have scaled up to tens of thousands of processors.

Particle simulations are a kind of typical applications supported by JASMIN. For these applications, particles can randomly distribute across the cells of a uniform rectangular mesh. These applications are usually used for the large scale computing of molecular dynamics, electromagnetism and so on. Usually, these simulations require careful tradeoff among data structures, communication algorithms (Brown et al., 2011), load balancing strategies (Chorley et al., 2009). Several particle application programs have been developed on JASMIN to support the peta-scale simulations tens of thousands of processors are used.

Component-based programming has been applied to address the requirements of large scale applications from sciences and engineering with high performance computing requirements (Francisco and Cenez, 2011). Component-based software engineering is to enable interoperability among modules that have been developed independently by different groups (Jalender et al., 2012). It treats applications as assemblies of software components that interact with each other only through well-defined interfaces within a software infrastructure.

In this paper, we emphasize component-based parallel programming for these particle application programs. Parallel integrator components are presented to shield the details of parallel data distribution, data communication and dynamic load balancing. They invoke numerical subroutines

Xiaolin C., Zeyao M. and Aiqing Z..
Component-based Parallel Programming for Peta-scale Particle Simulations.
DOI: 10.5220/0004587603340339
In *Proceedings of the 8th International Joint Conference on Software Technologies* (ICSOFT-EA-2013), pages 334-339
ISBN: 978-989-8565-68-6

written by users for numerical computing on a single patch. A particle application program can be assembled by using these components. The software complexity has been significantly reduced.

The organization of the paper is the following. We first briefly describe the software architecture and data structure of JASMIN infrastructure. Then, we detail the design and implementation of typical parallel integration components. We especially focus on the numerical integration component involving data communication and numerical computing. In section 4, a molecular dynamics parallel program has been developed by assembling these components. The program achieves a parallel efficiency above 60% on 36000 processor cores.

2 OVERVIEW OF JASMIN

2.1 Software Architecture

Figure 1 depicts the three layers architecture of JASMIN. The bottom layer mainly consists of modules for high performance computing for SAMR meshes. These modules encapsulate memory management, restart, data structures, data communication and load balancing strategies. The middle layer of JASMIN contains the modules for the numerical algorithms shared by many applications including computational geometry, fast solvers, mathematical operations on matrix and vectors, time integration schemes, toolkits, and so on. The top layer is a virtual layer consisting of C++ interfaces for parallel programming. On the top of this layer, users can write serial numerical subroutines for physical models, parameters, discrete stencils, special algorithms, and so on; these subroutines constitute the application program.

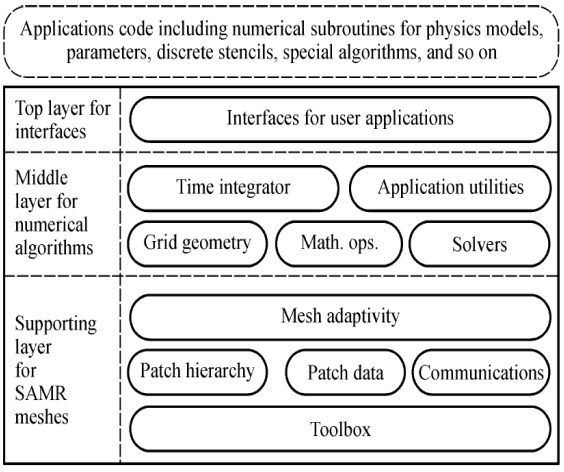

Figure 1: Software architecture of JASMIN.

2.2 Patch-based Data Structures

Figure 2 depicts a typical patch-based data structure (Mo and Zhang, 2009). Figure 2(a) shows a two-dimensional structured mesh consisting of 20x20 cells on one patch level. It is decomposed into seven patches and each patch is defined on a logical index region named “Box”. In each patch level, patches are distributed among processors according to their computation loads. In Figure 2(b), these patches are ordered and distributed between two processors. The left four patches belong to one processor and the right three green patches belong to the other processor. In Figure 2(c), the sixth patch is shown to illustrate the neighbour relationships. It is the neighbourhood of the other four patches and it shares contact with the physical boundaries.

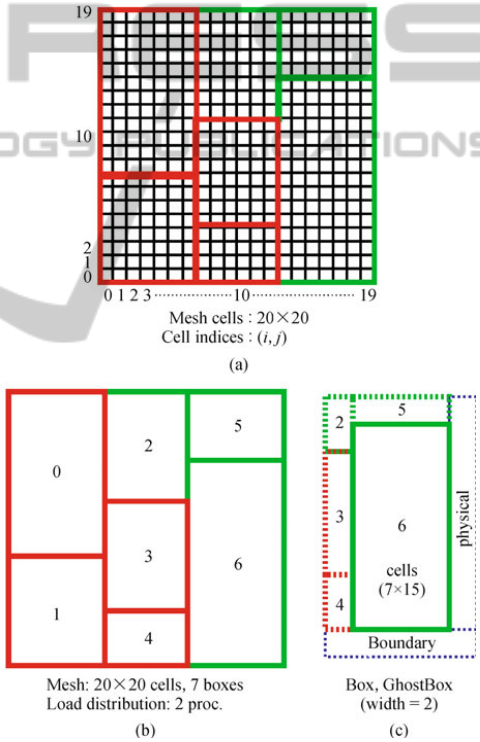

Figure 2: A mesh includes seven patches.

In JASMIN, patch is a fundamental container for all physical variables living on a logically rectangular mesh region and that all such data is accessible via the patch. A patch level is used to manage the structured mesh. For the patch level, a user can freely define the physical variables. On each patch, physical variables exist in the form of an array of patch-data. Patch-data is defined on the region with a ghost box. To store the data transferred from neighbour patches, each patch extends its box to a ghost box within a specific width. When memory

allocation is requested, the memory of each patch of data for all related variables is allocated on each patch.

3 PARALLEL INTEGRATOR COMPONENTS

3.1 Design Idea

On the patch-based data structure, the parallel algorithm designed on BSP model is organized as a series of parallel computing patterns involving data communication and/or numerical computations on patches. These typical patterns cover various data dependencies which occur in different phases of a numerical simulation such as variable initialization, time stepping, numerical computing, memory management, patch-data copy, parallel sweeping, particle motion, etc.

A parallel integrator component (PIC) was designed to encapsulate each parallel computing pattern. The component encapsulates the details of parallel data distribution and data communication among MPI processors. Furthermore, it organizes concurrent numerical computing among MPI processors or OpenMP threads. The component invokes user function to perform problem-specific numerical computing.

3.2 Implementation

The Strategy pattern is the primary object-oriented design tool employed in JASMIN to encapsulate a family of PIC components by making their constituent parts interchangeable through common interfaces. It was used to implement a family of PIC components. The *StandardComponentPatchStrategy* abstract base class defines an interface between the PIC components and problem-specific integrator object.

For example, the *NumericalPIC* component encapsulates a parallel computing pattern involving data communication phase and numerical computing phase. In data communication phase, it transfers data among processors for exchanging boundary data among patches. In numerical computing phase, it performs numerical computing on each processor for updating numerical solution on each patch. For the *NumericalPIC*, two abstract interfaces were defined in the abstract base class.

- *registerPatchData()* for registering physical variables needed to fill ghost cells before numerical computing.
- *computingOnPatch()* for implementing serial and numerical subroutine on one patch.

The NumericalPIC component possesses two private functions for data communication. Function *createScheduleOnLevel()* constructs schedule on new patch level for physical variables registered by user. This schedule includes memory copy in the same processor or message passing across processors. When a patch level was created or changed, the function was automatically called. Function *fillDataAmongPatches()* manages data transfer guided by the communication schedule for exchanging boundary data among patches.

The *NumericalPIC* component supplies a public function *computingOnLevel()*, which performs following two steps.

- In data communication phase, function *fillDataAmongPatches()* is automatically invoked for exchanging boundary data among patches.
- In numerical computing phase，each processor loops over all local patches and performs numerical computing on each patch by calling user function *computingOnPatch()*. For OpenMP threads parallelization, each thread deals with one or several patches.

3.3 Typical Components

Table 1 lists seven PIC components in JASMIN for typical parallel computing patterns covering various data dependencies in single level application(Mo, 2009). The name of PIC components was listed in the first column. The second column shows these functions involving parallel data distribution and data communication without user intervention. These numerical computing functions are shown in the third column.

For the four components involving numerical computing, user interfaces are defined in the *StandardComponentPatchStrategy* base class described as following:

- *initializePatchData()* for InitializePIC.
- *computingOnPatch()* for *NumericalPIC*.
- *getPatchDt()* for *DtPIC*.
- *getLoadOnPatch()* for *DlbPIC*.

In the user interface, parameter patch is very important for containing all physical variables. All data of physical variables are accessible via the patch. All dependent data coming from neighbour patches have been filled in ghost cells of patch after data communication. Therefore, user can write serial function to deal with physical variables on one patch.

Table 1: Seven typical parallel integrator components.

name	Parallel operations	Numerical computing
Initialize	Allocate memory; initialize from restart files	set initial value of physical variables
Numerical	Fill ghost cells	Update value of physical variables
Dt	reduce min value of dt	Compute dt
Particle-Comm	Migrate or fill particles	None
Copy	Copy physical variables	None
Memory	Allocate or deallocate memory	None
Dlb	Dynamics load balance	Compute load of cells

4 REAL APPLICATION

Particle simulations are typical applications supported by JASMIN infrastructure. Two particle application programs have been developed, which include classical molecular dynamics (MD) program, laser plasma intersection program and so on. Take MD program as an example of how we apply these PIC components for parallel programming in section 4.1-4.3. In section 4.4, laser plasma intersection program is briefly introduced.

4.1 Parallel Algorithm

The MD application calculates the time dependent behavior of a molecular system. The new position and the velocity of each particle are computed by numerical integration of Newton's laws of motion equations. every time step. The forces between particles are obtained by the summation of interactions between two particles at the cut-off radius. The interaction between two particles is obtained by potential function. Here we use the EAM (Embedded Atom Method) potential function (Brown et al., 2011). Parallel MD method based domain decomposition is described in algorithm 1 involving 7 steps. It mainly computes the forces and the positions of each particle while it transfers data for filling ghost cells and migrating particles.

```
Algorithm 1: parallel MD method
1) initial load balancing
2) set particles in each cell
  for all time steps do
3)  fill particles in ghost cells
4)  compute forces of particles
5)  update positions of particles
6)  migrate particles among cells
7)  dynamic load balancing
 end do;
```

4.2 Component-based Programming

For parallel programming on JASMIN, some PIC components have been assembled for implementing the corresponding step in algorithm 1.

```
Algorithm 2: assemble components
1) d_dlb->assign() //Dlb
2) d_init->initialize()//Initialize
  for all time steps do
3) d_pcomm->fill() //ParticleComm
4) d_force->computing() //Numerical
5) d_position->computing//Numerical
6) d_pcomm->migrate()//ParticleComm
7) d_dlb->adjust()  //Dlb
  end do
```

In step 1 and 7, a *DlbPIC* component named *d_dlb* was used for assigning initial load and performing dynamic load balancing. In step 1, it assigns evenly patches across processors. In step 7, it migrate patches across processors. The *DlbPIC* component includes many load balancing methods such as the space filling curves coupled with the multilevel average weights methods, greedy methods, geometrical bisection methods, etc. It invokes user function *loadOnPatch()* for defining the loads of each cell in a patch. Using this information, the *DlbPIC* can automatically distribute and adjust loads across processors. Figure 3 and Figure 4 depicts the load distribution and adjustment on 8 processors. Figure 3 shows the initial distribution of patches with non-uniform distribution of particles. Figure 4 depicts the redistribution of patches after the motion of particles. Patches with the same color are assigned the same processor. The patch with over-weight load is divided into several small patches.

In step 2, a *InitializePIC* named *d_init* was used for performing initial distribution of particles based on physical model. It calls user function *initializeOnPatch()* for setting the initial positions of particles and all attributes such as velocity, mass etc on a patch at time zero. These particles can distribute across the cells of a uniform rectangular mesh based on metal crystal structure.

In step 3 and step 6, a *ParticleCommPIC* named *d_pcomm* is used for halo-swapping and particle

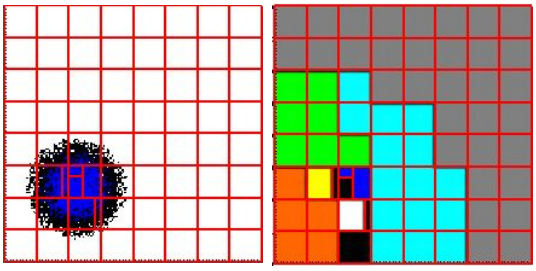

Figure 3: Assign initial load with non-uniform distribution of particles on 8 processors.

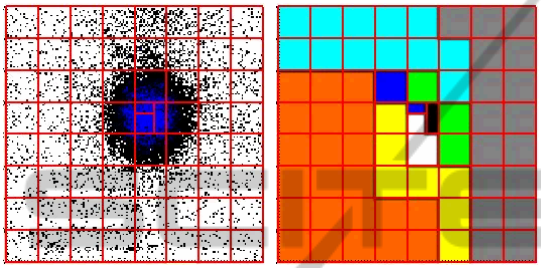

Figure 4: Dynamic load balancing after the motion of particles on 8 processors.

migration. Before computing forces of particles, the information about particles in cells at patch boundaries needs to be communicated. The component uses its own public function *fill()* for exchanging boundary particles among patches. After updating positions of particles, some particles might move from one cell to another. The component uses its own public function *migrate()* for migrating particles across processors. The two functions are performed without user intervention.

In step 4, a *NumericalPIC* named *d_force* is used for computing forces of particles. In step 5, a *NumericalPIC* named *d_position* is used for updating positions of particles. The two components both call user function *computingOnPatch()*. In the function, users implement two numerical subroutines written with F77 language. One is *computingForce()* , the other is *updatingPosition()*. In the first subroutine, the forces are computed between particles in the same or neighbour cells. In the second subroutine, the positions of particles are computed by integrating Newton's equation of motion. The two functions loop through all cells in one patch.

4.3 Peta-scale Simulations of MD

We developed a MD code with EAM potential on JASMIN infrastructure, which is named with md3d_EAM. A dynamics response of metal with nano-metre hole model has been simulated on 36000 cores of TianHe-1A supercomputer (Yang et al., 2011). The total number of particles in the simulation is 5.12×10^8 with a void density of 0.5%. Table 2 shows the parallel performance of strong scalability experiment with fixed the number of particles in 10 time steps. It has achieved parallel efficiency above 60% on 36000 cores.

This real whole simulation costs about 4 hours for 30000 steps. In figure 5, we can see from the simulation that voids collapsed by the emission of dislocation loops and hot spot initiated through the collapse of voids.

Table 2: Parallel performance with fixed problem size.

Cores	Time(s)	Efficiency
6000	15.92	100.0%
12000	9.32	85.4%
18000	5.88	90.2%
24000	5.35	74.3%
36000	4.37	60.7%

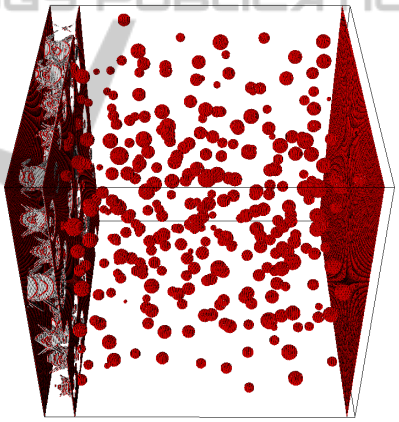

Figure 5: The shock structure and dynamic response of copper with hundreds of nano-metre hole.

4.4 Parallel Program of Laser Plasma Intersection

LARED-P is a three-dimensional program for the simulation of laser plasma intersections using the method of Particle-In-Cell. Electrons and ions are distributed in the cells of a uniform rectangular mesh. The Maxwell electromagnetic equations coupled with particle movement equations are solved. Particles intersect with the electromagnetic fields.

The LARED-P program has been developed by assembling these components described in Table 1. The program achieves a parallel efficiency above 45% for a typical real model using 20 billion particles on 36000 processor cores. For this

simulation, the efficient load balancing strategies are essential for successful executions.

Figure 6 shows the distribution of particles in a snapshot and the related volume rendering of laser intensity. The distribution of the particles is showed with yellow colour. Laser energy is mainly distributed in the internal cavity of cone target. It demonstrated clearly the existence of plasma significantly affects the light propagation and the generation of relativistic electrons.

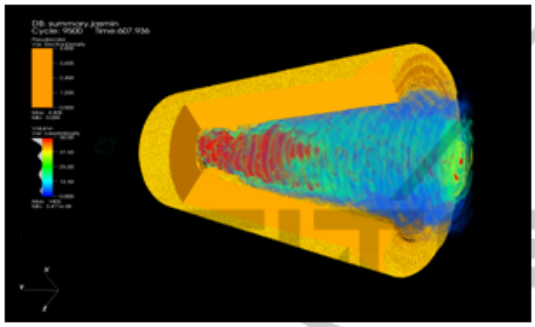

Figure 6: Focusing transport of laser in plasma taper.

5 CONCLUSIONS

Component-based software engineering is very useful to address the requirements of large scale applications in scientific computing. Components are logical means of encapsulating parallel details from computer science domain for use by those in real application domain. Parallel integrator components in JASMIN infrastructure are presented to shield the details of parallel data distribution, data communication and dynamic load balancing. It has been proved that particle application programs can be easily implemented by assembling these components. Users mainly write application-specific numerical subroutines invoked by these components.

The complexity of parallel programming continues to increase as multi-model, multi-physics, multi-disciplinary simulations are becoming widespread. These challenges make it clear that high performance scientific computing community needs advanced software engineering techniques which facilitate managing such complexity while maintaining scalability and parallel performance. Parallel software infrastructures along with these advanced techniques will allow domain programmers to “think parallel, write sequential”.

ACKNOWLEDGEMENTS

This work was under the auspices of the National Natural Science Foundation of China (Grant Nos. 61033009), the National Basic Key Research Special Fund (2011CB309702) and the National High Technology Research and Development Program of China (863 Program) (2012AA01A309). Thanks for the many contributions from members of the high performance computing centre in IAPCM.

REFERENCES

Brown W. M., Wang P., Plimpton S. J., 2011. Implementing molecular dynamics on hybrid high performance computers - short range forces. *Comp Phys Comm.* 182: 898-911.

Chorley M. J., Waker D. W., Guest M .F. Hybrid message-passing and shared-memory programming in a molecular dynamics application on multicore cluster. *International Journal of high Performance Computing Applications*. 23(3): 196-211, 2009.

Francisco H. C., Cenez A. R., 2011. Component-based refactoring of parallel numerical simulation programs: a case study on component-based parallel programming. In *SBAC-PAD '11, 23rd International Symposium on Computer Architecture and High Performance Computing*. IEEE Computer Society.

Jalender B., Govardhan A., Premchand P., 2012. Designing code level reusable software components. *Int. J. Software Engineering & Applications*. 3(1): 219-229.

Mo Z. Y., Zhang A. Q., 2010. JASMIN: A parallel software infrastructure for scientific computing. *Front. Comput. Sci. China*. 4(4): 480-488.

Mo Z. Y., Zhang A. Q., 2009. User’s guide for JASMIN, Technical Report . *https://www.iapcm.ac.cn/jasmine*.

Pei W. B., Zhu S.P., 2009. Scientific computing in Laser Fusion. *Physics* (in Chinese), 38(8): 559-568.

Post D. E., Votta L. G., 2005. Computational science demands a new paradigm. *Physics Today*, 58(1): 35-41.

Yang X. J., Liao X. K., Lu. K., 2011. The TianHe-1A supercomputer: Its hardware and software. *J. of Computer Science and Technology*. 26(3): 344-351.