Parallel Particle Framework Project Proposal

Martin Casado
Northern Arizona University
3515 N. Andes Dr.
Flagstaff, AZ 86004

Noel Keen
Lawrence Livermore National Laboratory
7000 East Ave. L-39
Livermore, CA 94550

October 17, 1999

Contents

List of Figures

List of Tables

Abstract

This document is a proposal to develop the Parallel Particle Framework (PPF). The Parallel Particle Framework will be an API to ease the parallelization of Monte Carlo physics packages simulating radiation particles. This document covers the problem statement, requirements, specifications, risk analysis, team standards, resource and schedule, design process and technical concepts to be used by the Parallel Particle Framework team.

Chapter 1

Problem Statement

1.1 Background

Physical modeling of real world phenomena often demands significant computer resources, such as speed and memory, that far exceed that offered by even the largest single processor or shared memory systems. The national laboratories including Lawrence Livermore, Sandia and Los Alamos, are using large distributed computer networks to handle these problems. Parallelization of an algorithm across a distributed network is not a mechanized process and the approach differs drastically between problems. Fortunately, some of these problems can be classed, and generic solutions applied to them. One class of these problems is radiation transport problems.

Radiation transport using the Monte Carlo method is well-known and is used extensively at Lawrence Livermore National Laboratory (LLNL). To accurately model a sophisticated three-dimensional problem including detailed geometry specification it is common to represent the problem with a *mesh*. The resolution can be controlled by the number of *zones* in the mesh which dictates the size of the problem, that is the amount of memory needed to store the mesh. As the mesh sizes become large it will not fit on even the largest single memory unit. (Note that even if it could, we would still want to run an even larger problem simply because we can – i.e. distributed memory...) Therefore, in order for the Monte Carlo particle packages to handle problem descriptions larger than conventional memory on a single shared memory node, the problem must be distributed across multiple nodes. A distributed network computer must then *domain decompose* the mesh such that it can be read into memory. Using a distributed network computer requires the application to communicate between nodes using message passing techniques such as with the MPI library. Learning the details of distributed computing requires time and experience not common to many developers of particle transport packages. In order to provide developers with an intuitive interface to efficiently handle domain decomposition and particle passing, we propose development of the Parallel Particle Framework.

1.2 Business Issues

Radiation particle packages are written largely by physicists without formal backgrounds in computer science. Learning the intricacies of distributed computing and applying it to a project increases the time of development and may compromise the correctness of that project. More specifically, the business issues surrounding the parallelization of radiation transport packages are as follows:

- Parallel training increases project times.
- Parallel computation is not germain to many package developers fields and can be viewed as a waste of resources.
- Efficiency and correctness of parallelization requires experience and knowledge of architectures that many radiation transport developers lack.
- Currently, each developer is approaching the parallelization problem differently. The absence of a common structure makes this class of code difficult to maintain.
- Current parallelization techniques used are not conducive to future changes, such as:
  - increased scalability
  - load balancing
  - optimization techniques
- Current parallelizations are not abstracted from the code to support adaption to new parallelization technology.

1.3 Product Statement

The Parallel Particle Framework (**PPF**) project involves designing and developing an API which will be integrated into physics modules within the KULL[1] framework. The API will provide a high level interface for the parallelization of particles over large meshes[2] for radiation transport physics packages developed within KULL. The PPF will decompose the problem mesh over large distributed super computers and handle all communication of particles moving between nodes. The API must be constructed to integrate into existing physics packages without changing the original programs' paradigms.

1.4 Projected Value

The goals of the PPF are to save time in developing particle Monte Carlo packages and to provide a standard method of parallelization. Benefits of a technological solution are:

- Reduction of development and training hours in order to reduce the time and cost of a project.
- Maximization of resources by enabling physicists to concentrate on work applicable to their areas of study.
- Ensured correctness in parallelization.
- Increased maintainability of code, due to standardization.
- Ensured robust, efficient parallelization over distributed systems, improving the end product.
- Preparation for future changes to distributed mechanisms by abstracting parallelization methods.

[1]KULL is a large physics modeling package currently in development at LLNL
[2]a mesh is a collection of geometric components which represent a physical problem

1.5 Business Environment

The parallel particle framework is intended to be integrated into radiation transport packages within the KULL framework. The KULL project environment requires the PPF to adhere to the following constraints:

- It must conform to the C++ standard.
- It must compile on Digital Unix and AIX using the KCC compiler.
- It will follow the KULL documentation standards.
- Builds should be easily integrated into the larger KULL make system.

Since the PPF is meant to be integrated not only into new projects but existing code, the interface must be conducive to the parallelization of serial code. That is to say, PPF must not require the project architecture to be built around it, but more so be easily integrated into existing architectures.

In general, applications which use the PPF must accept following assumptions:

- Particles do not interact with each other. They are absolutely independent.
- Particles contain the following base information:
  - position in the mesh(x,y,z)
  - direction of travel(u,v,w)
  - time(t) since being sourced
  - energy(E) carried
  - current zone in the mesh
- The mesh is rigid and particles move through the mesh.
- Particles are simulated and change mesh-based values such as energy deposited, material temperature, etc.) which are quantities of interest in the problem.

Chapter 2

Requirements

2.1 Goals

The goal of the PPF is to be a robust, efficient, and correct library to help in the parallelization of radiation particle transport packages. Specifically the goals of the PPF are to:

- Distribute workload of radiation particle transport problems semi-efficiently between nodes of large distributed supercomputers.
- Provide scalability of a problem across arbitrary nodes.
- Decompose the problem mesh across across multiple nodes transparently to the developer.
- Provide transparent particle passing between nodes/mesh sections.
- Eliminate the need for developers to learn the intricacies of distributed parallel programming.
- Provide a standard for particle physics module architectures.
- Reduce development and training time for radiation particle transport packages.
- Create a tracking system to aid in particle visualization.

2.2 Benefits

The benefits of creating a unifying parallelization library are as follows:

- Timeliness: The PPF will perform much of the work of parallelization so the code does not have to be re-written for each package.
- Efficiency: The PPF will ensure efficiency in message passing by limiting the amount of data that is passed, and only passing data when appropriate.
- Consistency: Packages using PPF can be contrasted because a unifying parallel framework provides consistent performance in the parallel regions.
- Accuracy: The PPF will ensure accuracy in parallelization so that particles are not lost, transported incorrectly or delayed.

2.3 Software Requirements

Since the parallel particle framework is a library to be used by developers of large physics problems where correctness and efficiency is crucial, the library must have the following attributes.

- Ease of Use: PPF must provide a simple interface that is intuitive to a developer with no experience in parallel computing. All code dealing with parallelization technology must be transparent to the developer.
- Efficiency: Large scale problems running billions of iterations on thousands of nodes can continuously accumulate small efficiency problems eventually leading to a significant increase to the running time of the problem. For this reason the PPF must be extremely efficient in message passing and performance.
- Stability: Problems run by radiation transport particle packages may take days to complete. For this reason it is imperative that PPF be stable by ensuring no runtime errors nor memory leaks. This is more important for PPF than other small scale applications because a faulty run can cost the developers days at a time. In the advent of a crash PPF should be able to resume operation at the last time step completed.

- Scalability: The platforms used for large scale radiation transport problems are continuously growing in size, therefore in order for PPF to be of use in the future it must be scalable to an arbitrary number of nodes efficiently.
- Longevity: PPF must ensure the integrity of all its requirements for the life of the package which implements it. The KULL framework is expected to be a 20 year code, the PPF should retain functionality for at least this time period.

2.4 Functional Requirements

The following two lists are details of the costumers needs and wants for the Parallel particle frameworks functions.

2.4.1 Needs

- The library must be able to handle split meshes by indexing matched faces between mesh partitions.
- During runtime the PPF must be able to make the following decisions correctly:
  - when to stop tracking particles on a partial mesh and send particles frozen on a split surface to another domain
  - how many particles to buffer before sending
  - how often to perform checks whether or not to send buffered particles
  - when to ask if another domain is ready to send particles
  - how to handle load balancing if possible
  - what to do if one domain ends up with too many particles to fit in memory
- Particles must be sent as efficiently as possible.
  - particles should not be packaged with extra information for sending

- data should only be sent to nodes which require it, no broadcasts

- Particle transfers between nodes must be accurate.

- The library's interface must be intuitive and not dependent on a prior knowledge in distributed computing.

2.4.2 Wants

- The library may handle large mesh sizes and adapt to changes in the mesh interface.

- The PPF may provide support for general domain decomposition and work sharing between nodes.
  - The mesh may be re-decomposed dynamically to handle load balancing.
  - The PPF may use a domain replication scheme to fairly distribute work loads.

- The PPF may include shared memory parallelization within each node using pthreads or OpenMP.

- The library may include extensions for particle visualizations.

- The library may decompose any given mesh using a graph cutting package such as Parmetis.

2.5 Non-Functional Requirements

- All code written for PPF must be compatible with both the egcs and KCC compilers

- Programming must conform to sound object oriented design, and make use of MPI2 if a reliable implementation is found for the IBM SP2 architecture.

- PPF code must have general compliance with POSIX standards.

- There must be readable and comprehensive documentation in all stages of the development process.

- The library must compile and run on AIX 4.3 and DEC Unix.
- The library must be easily integrated into the make system of existing projects.

2.6 Additional Constraints, Concerns or Requirements

- KULL is an export controlled technology and the PPF could fall under this category so open source may be impossible.

2.7 Appendix: API Functionality Overview

(For interface description and function specifics, refer to chapter 7)

Chapter 3

Risk Assessment

The following section provides an overview of potential risks in developing the PPF, risks will be divided into the following categories:

- Product Size Risk
- Business Impact Risks
- Process Risks
- Technology Risks
- Development Environment Risks

3.1 Product Size Risks

Risks
Addition of extraneous code not directly pertinent to a project.
Limited number of users may limit PPF's benefits.
The library may unnecessarily complicate the project using it.
Attempts to save a problems progress in the advent of failure may use large amounts of system resources including drive space.
The PPF may add a disproportionate amount of overhead to small problems.
MITIGATION
Development efforts should focus on limiting the code for PPF to only handle particle passing and mesh decomposition which is common to all projects.
During requirements capture, PPF's usefulness should be evaluated in light of the number of foreseeable clients.
Efforts should be made during development to make the library linkage and interface as simple as possible.
PPF should only do large data dumps to hard disk if explicitly sanctioned by the user.
It should be agreed by developers to use PPF only if the problems they run are large enough to warrant it.

3.2 Business Impact Risks

Risks
Delivery deadlines may be unreasonable.
Late deliveries may result in additional costs to Livermore.
Defects in the PPF may have negative consequences.
Governmental constraints may inhibit development of PPF.
Integration into larger systems may be problematic.
The PPF interface may be too difficult to use by developers.
The the developers needs may not remain consistent over time.
Senior management at Livermore may not approve of the PPF.
Documentation for developers may not adequately describe the PPF.
Mitigation
Careful planning in the design phase will help provide reasonable deadlines for deliverables.
Developers intending on using PPF will be updated as to its progress during development so they will be able to plan around the projects progress.
The PPF must be thoroughly tested to eliminate defects.
Full specification and design of project must be presented and approved of by senior LLNL management before development.
Packages which will include the PPF must be carefully studied during design to ensure that there will be no incompatibility issues.
Efforts must be directed towards keeping the PPF interface simple.
The PPF should provide a basic functionality that will be useful for the life of the KULL project.
As stated before, the proposal of the project must pass senior management approval.
Each stage of the design and the development process must be fully documented.

3.2.1 Process Risks

Risks
There is no process standard for development.
The lack of a write-up on the process may result in design and development in an ad-hoc fashion.
Communication barriers between Livermore and Northern Arizona University can result in discrepancies in the process design.
There has not been any full projects similar to PPF to base the process on.
Both members of the project are relatively inexperienced in organized process for design and development.
Time constraints will limit testing and code review.
The nature of the PPF limits test case design and use cases.
Mitigation
The PPF should follow a commonly used process that has proven to be effective on other projects.
An effort should be made to document all aspects of the process to aid in a structured development stage.
Planned weekly meetings and regular email exchanges will help ensure both parties are on the same page.
Attention of the process selection based on the nature of the project will provide a basis for design in the absence of experience.
Reviewing process decisions with professors/managers will help correct mistakes made from inexperience.
The project scope must be limited in light of time constraints to guarantee a functional and stable product.
Interface issues can be resolved by approaching developers who intend to use the PPF library.

3.2.2 Technology Risks

Risks
Parallelization techniques may become outdated within a few years.
Load balancing is difficult and ensuring optimality is virtually impossible.
The PPF will be interfacing with BLUE's hardware which is experimental and unproven in stability.
Misuse of the PPF will compromise its performance.
Simplifying the interface may compromise the library's functionality.
The nature of the project requires facing problems which have not been dealt with or mitigated in the past.
There is an uncertainty that the requirements requested are doable.
Mitigation
The parallelization mechanism should be abstracted from the functionality of the PPF so changes to a new technology will not be difficult.
It should be understood by the developers who use PPF that the load balancing may not be optimal.
Frequent comparison between builds on a proven system such as the TERA cluster will help isolate errors caused by anomalous behavior by BLUE. These problems can be brought to the attention of Livermore Computing and/or IBM.
Careful and comprehensive documentation will help prevent product misuse.
The priorities within the project must be analyzed prior to design. If one decision compromises another the one with higher priority is chosen.
If a perfect algorithm cannot be developed to handle all cases, a heuristic will have to do, and all shortcomings of the algorithm used must be documented.
If the requirements prove to be to difficult a review of the requirements must be brought before the clients and an acceptable compromise must be reached.

3.2.3 Development Environment Risks

Risks
The project team does not have access to a process management tool.
The project team has little access to design and analysis tools.
Popular design and analysis tools are not appropriate for development of the PPF.
Compatibility between the KCC and egcs compilers may be problematic.
Cross platform compatibility may prove problematic.
The project members have no experience debugging threads on the sp2 or Digital Unix platform.
Thread debuggers for all platforms may not be available.
Mitigation
Process management can be accomplished by use of programs such as Microsoft Project run on a separate machine than that used for the project.
The PPF project does not warrant any special tools for design or development, design should be rather straightforward as the library has relatively few use cases.
Again the nature of the PPF project does not require the use of extravagant design and development tools.
All code for the PPF must follow the C++ standard.
Continuous builds on all target platforms will ensure compatibility at every step of the development process.
Project members must research debugging methods for threads on the sp2 and Digital Unix platform.
In the case that an adequate debugger is not found, the threads can be debugged separately from the running code using a traditional debugger.

Chapter 4

Team Standards

4.1 Roles

4.1.1 Leader

Martin Casado

4.1.2 Communicator

Martin Casado

4.1.3 Recorder

Martin Casado

4.1.4 Project Design

Martin Casado, Noel Keen

4.1.5 Programmers

Martin Casado, Noel Keen

4.2 Meetings

4.2.1 Weekly Time for Outside Meetings

Weekly meetings will be held Wednesdays between 11:30am and 12:30pm over the phone, unless conflicting with Noel's schedule at LLNL. If a meeting is missed it will be held at a later time or by discussion via email.

4.2.2 Standard agenda

The following items will be discussed in each meetings if applicable.

- Review of last meeting
- Review of previous weeks goals for current week
- Discussion of current status of the project
- Discussion of how we succeeded or fell short of goals set at previous meeting
- Discussion of problems in development and design
- Set goals for the next week
- Conclude

4.2.3 Decision Strategy

All decisions will be made upon agreement by both Martin Casado, and Noel Keen. If opinions result in a deadlock Dr. Pat Miller's opinion on the matter will serve as the final decision.

4.2.4 Minutes

Minutes will be kept for each meeting by Martin Casado and distributed via email to Noel Keen and Stu Wecker by the end of the day Wednesday. In the advent that the meeting did not take place at its scheduled time on Wednesday, minutes will be distributed via email at the end of the day in which the meeting took place.

4.3 Documentation

4.3.1 Typesetting Language

All documents produced during the parallel particle framework project will be done using the $\LaTeX\ 2\varepsilon$.

4.3.2 Coordination

All documentation will be written by Martin Casado and submitted to Noel Keen for editing and revision. If time constraints restrict Noel from going over the paper, Martin will submit the papers directly.

4.3.3 Version Control

Documentation revision numbers will be maintained in the comments at the top of each $\LaTeX$ file. All changes should be added to the comments, as well who made the changes and the date. Official copies of all documents will be kept by the document coordinator.

4.3.4 Format

All documents will adhere to $\LaTeX$ file templates and document classes which will be created and revised as the project progresses. Each document will use the standard $\LaTeX$ styles, and set the following options

- 12pt font
- oddside margin 2.54 centimeters
- evenside margin 2.54 centimeter
- Roman style font

4.3.5 Review Process

Documents will be written by Martin Casado and submitted to Noel Keen for review. All drafts must be finished by a pre-determined deadline and submitted for review before the final paper is written.

4.4 Development Standards

4.4.1 Version Control

All code versioning will be done through Perforce on the LC machines and CVS in Linux.

4.4.2 Commenting and Documentation

All comments and documentation of development will follow KULL documentation standards defined by the KULL team.

4.4.3 Compilers and Platforms

All program development will be on the UNIX operating system and support the following platforms.

- IBM AIX 4.3
- Digital Unix
- Linux

All code developed will compatible with the KCC and ecgs compilers.

4.5 Self Evaluation Methods

4.5.1 Weekly Evaluations

Each week, Noel Keen will submit a quick email memo evaluating Martin's performance on the project. Each evaluation will be printed and saved in the project notebook, and submitted to Stu Wecker upon request. Martin will as well, submit a personal evaluation into the project folder weekly.

4.5.2 Evaluation Contents

Each evaluation will discuss positive aspects of Martin's performance as well as negative, and provide a final statement on whether the performance was low, average or well done.

4.6 Standards for Behavior/Cooperation

4.6.1 Design Changes

All design changes must be approved by Noel Keen. Both Noel Keen and Martin Casado can propose design changes.

Chapter 5

The Design/Development Process

5.1 Methodology

We propose to use the iterative design process for construction of the Parallel Particle framework. Preliminary steps for development are as follows:

- Create workspace with working PolyMesh code.
- Develop a fake physics package to randomly move particles around the mesh.
- Write a driver file to model a use of the PPF
- Create the skeleton of the PPF. All methods should be defined, however to begin with they will have no functionality.
- Incrementally add functionality to the methods.

5.2 Deliverables

During the course of the PPF project, deliverables demonstrating the teams progress will be submitted to Livermore for review. The deliverables will include:

- PPF interface description documentation
- PPF OO architecture write-up
- PPF skeleton mock-up
- PPF demonstrating basic functionality on a false physics package
- Test results for runs of PPF on:
  - large particle sets, few nodes
  - large particle sets, many nodes
  - large particle sets, serial run
  - small particles sets, few nodes
  - small particles sets, many nodes
  - small particle sets, serial run
- Final product

Chapter 6

Resources

6.1 Tools and Environment

The following is a list of software and hardware resources available to the PPF project team:

6.1.1 Compilers

The compilers which will be used for PPF development are as follows:

- egcs 2.95.1
- KCC 3.4d

Compilers may be updated during development.

6.1.2 Platforms

All targeted platforms are large-scale distributed supercomputers at Livermore. If time is available during development, PPF will be ported to Linux as well.

Need:

- TERA Cluster
  - 24 DEC AlphaServer 4100 Model 5/533 SMP systems

  - Each system has 4 CPU's and 1 GB memory
  - Processor speed of 533 MHZ
  - Currently available with 23 nodes
  - All nodes run Digital UNIX 4.0

- ASCI Blue-Pacific

Want:

- Linux

6.1.3 Version Control

- Both Blue and the Tera cluster share an NFS drive. One both of these platforms **Perforce** will be the revision control system.

- If time constraints permit the development of PPF on Linux, CVS will be used for revision control.

6.1.4 Debuggers

- TotalView

- gdb on Linux?

Chapter 7

Technical Concept

7.1 Discussion of Views

This section describes the PPF with respect to its function, data flow, and interaction with the current environment.

7.1.1 Functional View

Functionally, the PPF is a library which is compiled or linked with the physics package using it. All methods available to the user from the PPF are public methods in the PPF_Interface class. The PPF is intended to mask the programmer from the intricacies of parallel programming as well as distributed domains. Figure 7.1 diagrams the users perspective when using PPF with a physics package.

7.1.2 Data Flow

The major component of data within a package using the PPF is the particle. The particle is passed between the physics code and the PPF throughout program execution. The figure 7.2 illustrates the particles interactions with the package and the PPF during runtime:

7.1.3 Physical View

Packages using the PPF will be run as MPI programs. At the start of execution the program binary will be distributed to all participating nodes. During

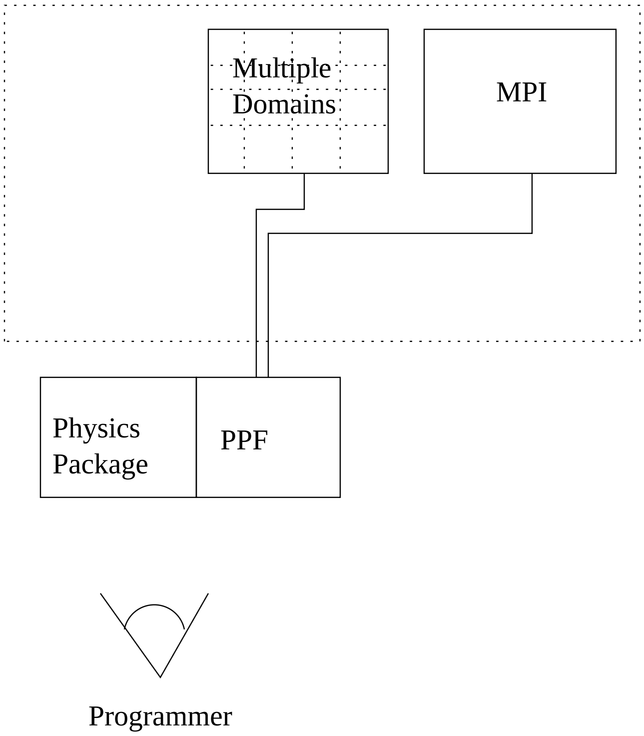

Figure 7.1: Programmers view of physics package with PPF. PPF hides issues of multiple domains and all MPI programming.

program execution, the PPF will use MPI as a medium of communication between nodes. All buffering issues will be handled by the PPF.

7.2 Preliminary Design

7.2.1 Block Diagram

An abstracted view of high-level object interactions can be seen in figure 7.3.

7.2.2 Interface Option 1

The following preliminary design option gives the PPF more control over the particles. The added control enables the PPF to force the programmers to correctly implement domain decomposition of their code and cuts down on the amount of code the developer using the PPF must write. The drawbacks of implementing the PPF with more control is that it makes it more difficult to integrate the PPF into existing code and packages using the PPF will contain large amounts of proprietary code that is PPF specific.

- **PPF.submitParticles** pass the PPF a pointer to the list of particles
- **PPF.reset** set the current position in the particle list to the first particle
- **PPF.nextParticle** return a pointer to current particle in the list, advance current particle to the next particle in the list
- **PPF.moveTo** move current particle to x position, y position in mesh (Note: message passing must be considered in this method)
- **PPF.getParticle** return the current particle
- **PPF.addParticle** add a single particle to the current position
- **PPF.distributeParticleList** add the particles within a linked list to the current list of particles by randomly distributing them throughout the current list.
- **PPF.deleteParticle** delete current particle, or particle at a given index

7.2.3 Interface Option 2

A minimalist approach to the design of the PPF give the developers more freedom to apply PPF to various problems and code structures. However, the PPF will not be able to guaranty load balancing and development under these loose guidelines may be detrimental to the integrity of the architecture of the package. A minimal interface for the PPF package must include the following methods.

- **PPF.moveTo** Move a particle to a different node

- **PPF.sendParticles** Send all of the particles frozen on a nodes edge to the neighboring node.
- **PPF.recieveParticles** Receive particle list being sent from a different node.

This model assumes that all other concerns of particles positions are handled by the package developer.

7.2.4 Appendix I: Example Driver File Pseudo-Code

The following code snippet is an example of how the PPF may be used in a project. Particle packages as not limited to a single design however, a method's logical position within the codes execution structure must remain the same.

```
(1) Compute One Time Step {
(2)   generateSources()
(3)   Loop over Particles{
(4)       Advance Particle {
(5)          Do Work
             if Frozen {
(6)             PPF_Submit(particle)
(7)             Erase particle from current node             }
(8)          PPF_SendFrozenParticles()
(9)          PPF_RecvFrozenParticles()
          }
       }
(10) PPF_DoneAdvancing()
(11) if(PPF_NumRemainginPsarticles == 0){
(12)     we are done with this time step
     }
(13) PPF_Repartition()
     Replicate Mesh()
```

7.2.5 Appendix II: Driver Description

1. Each time step is an independent parallel problem. All particles must be run to completion in each time step before continuing.

2. This line generates the source particles which are coming from outside, sources within the mesh, or are "census particles" which are particles left over from the last time step. Source particles also include any previous "combining" of particles.

3. The important item here is that the particles do not need to be computed in any particular order (Note: there is a potential for cache optimization here).

4. Move the particle within the mesh

5. Each particle's work can be computationally intensive, this is where the runtime code spends most of its time.

6. The application code using PPF decides which particles have hit the mesh boundaries and at what times, these particles are then submitted to the PPF.

7. Application removes particle from list of particles still advancing.

8. Ask PPF to send all particles to the correct nodes.

9. Ask PPF to receive particles sent from other nodes

10. Let PPF know that the the advancing particles loop is finished.

11. If there are particles that are jumping back and forth between meshes, keep iterating until they have all terminated, otherwise the time step is finished.

12. If PPF handles general domain decomposition it must be done between time steps, here we can re-partition and replicate.

7.3 Feasibility

The issues that have not been resolved are as follows:

- If a node's memory is completely used by the particles within that mesh partition, how can PPF stop particles being sent to that node?

- If particles are stalled within the MPI_Send of the sending nodes then we risk buffer overflows on the sending sides system buffer.
- If we require programmers to check whether a node's memory is saturated we compromise ease of use and PPF becomes more difficult to integrate into existing code.
- If we manage to stop sends to that node there are a number of timing issues we must deal with.

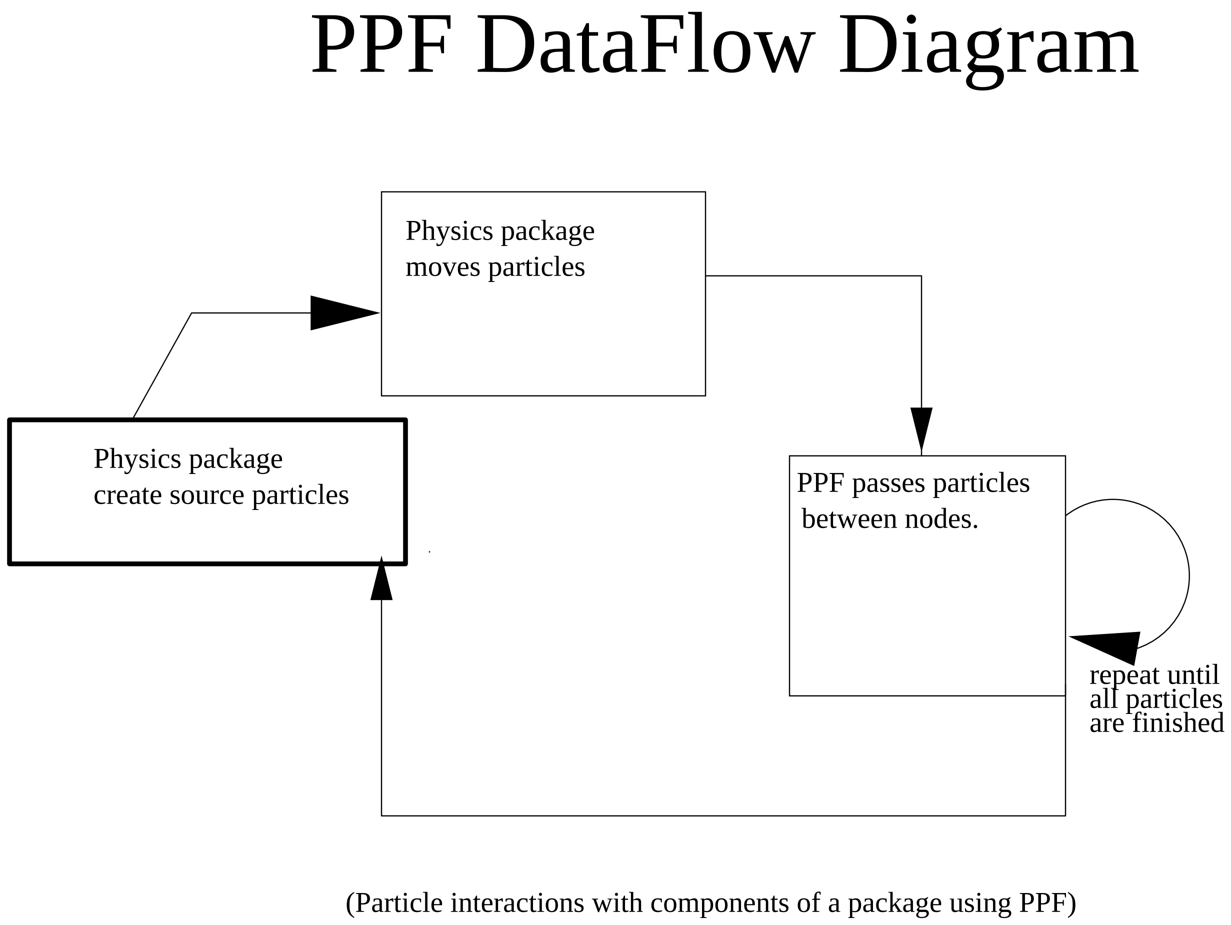

Figure 7.2: Flow of particle interactions in a package using PPF

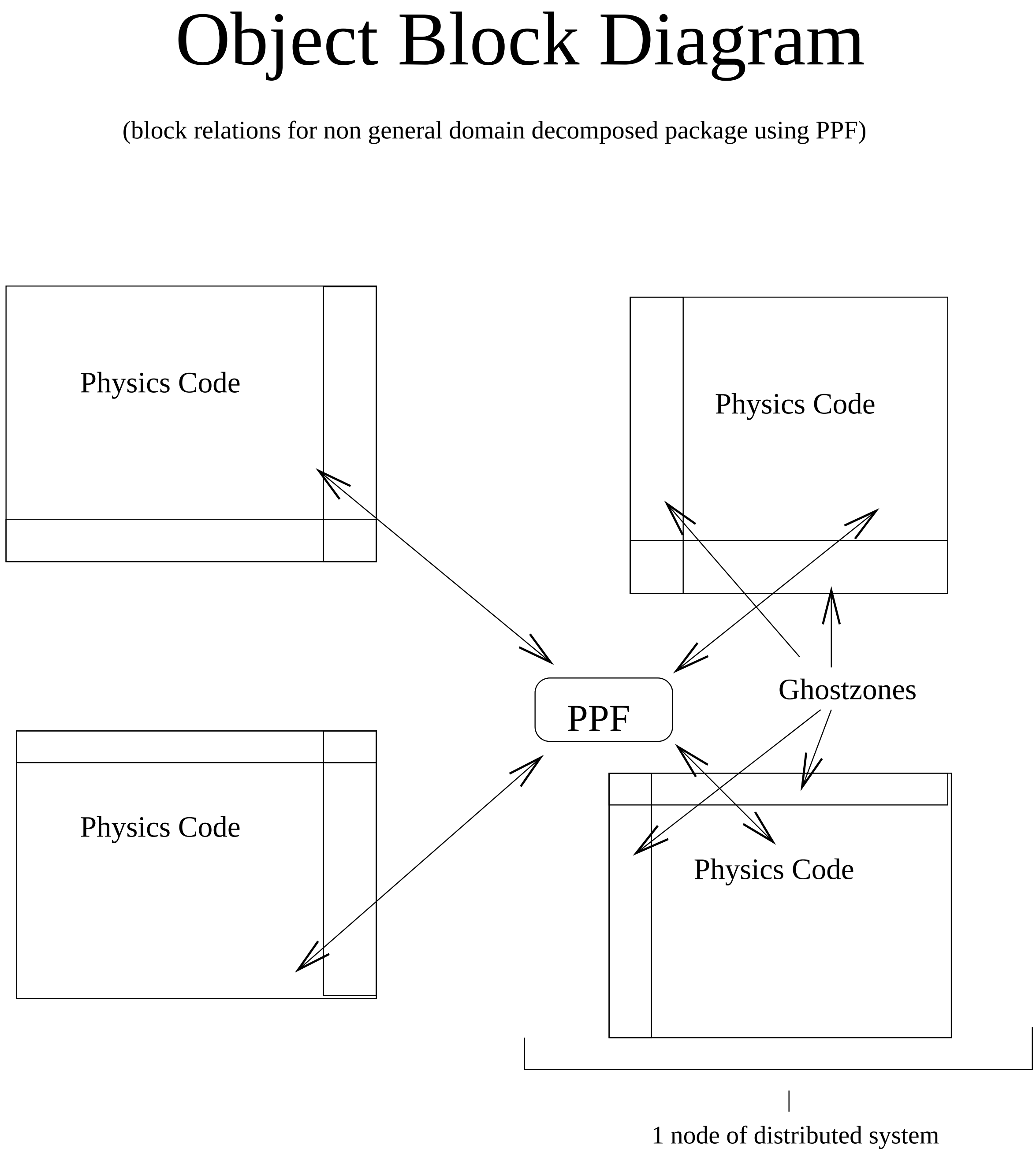

Figure 7.3: Block diagram of component interactions of a package PPF in a using PPF in parallel environment.

Chapter 8

Architecture

We have decided to begin by implementing the second option mentioned in the design section. This method allows the package programmer more flexibility since the PPF is only responsible for send and receives of lists of particles. Secondly, this method allows the PPF to be more easily integrated into existing code because it demands less PPF specific sections. In the advent that the PPF should need more control, it can be implemented on top of the current method which is a logical stepping stone.

8.1 Class Descriptions and Interactions

The PPF will be designed using a minimum of five classes distinguished by responsibility.

- **PPF_Interface** Facade class [1] whose responsibilities are as follows:
  - knows which subsystems' classes are responsible for a request
  - delegates client requests to appropriate subsystem objects
- **PPF_Statistics** Keeps track of statistics for each run including:
  - number of particles sent from current node
  - number of particles received from each neighbor

[1] Design Patterns, Gamma, Helm, Johnson, Vlissides. Addison Wesly, 1995. Pg 185-193

  - average time for send and receives
  - average time between send and receives
  - time it took for the run from initialization to conclusion

- **PPF_Timer** Simple utility timer accurate to milliseconds

- **PPF_Logic** Handles all logic for particle passing, including:
  - which domains to pass particles too
  - which domains to receive particles from
  - if not enough particles are in a list don't send it (unless the send is forced)
  - if the current node is out of memory, write particles to disk

- **PPF_Parallel_Wrapper** Wrapper class to abstract all parallel calls to MPI

Figure 8.1 shows a general block diagram of class interactions in the PPF.

8.2 Data Structures

The architectural approach we choose in creating the PPF does not require an extensive use of data structures. Rather, the majority of the code will involve the technicality of sending particles between nodes and keeping track of statistics. However, the PPF will include the following data structures:

- Particles will be stored and passed between functions in linked lists or vectors from the Standard Template Library.

- Each node will contain a lookup array defining the neighbors. An enumerated type will represent each of the nodes sides and the neighbor's ID will be placed in the index represented by the value of the enumerated type of that side.

This list is by no means inclusive, for example disk writes of particles in saturated nodes may require the use of buffers and lookup tables.

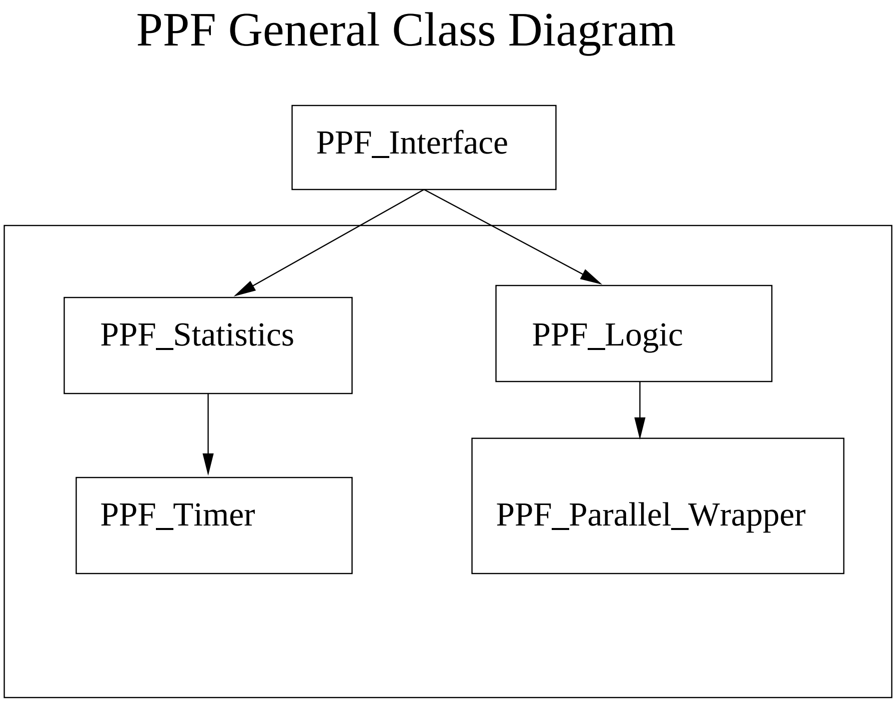

Figure 8.1: Flow of particle interactions in a package using PPF

Chapter 9

Schedule

9.1 Project Table

The following project table is a planned schedule for spring semester, 2000.

Work Tasks	**Planned Start**	**Planned Complete**
1. Finish Proposal		
Specs	finished	finished
Research	finished	finished
Schedule	wk1, d2	wk1, d4
Architecture	wk1,d2	wk1, d5
2. Validate with Sponsor:		
Meet with Sponsor	wk2,d3	wk2,d3
Validate Web-site	wk2,d1	wk2,d3
3. Do Detailed Software Design		
Data	wk2,d4	wk2 d6
Processes	wk3,d1	wk3,d3
Objects	wk3,d3	wk3,d5
4. Coding and Implementation		
Interface Class and Headers	wk4,d1	wk4,d3
PPF_Logic Prototype	wk4,d3	wk5,d3
Parallel_Wrapper	wk5,d1	wk6,d1
PPF_Logic with Wrapper	wk6,d1	wk7,d3
Statistics and Timer	wk7,d3	wk8,d1
5. Finish Code		
Testing	wk8,d1	wk10,d5
Revision	wk8,d1	wk10,d5
Review	wk8,d1	wk10,d5
6. Finish documentation		
Code Documentation	wk11,d1	wk11,d5
Project Usage Documentation	wk11,d1	wk11,d5

9.2 Gantt Chart

Work Tasks	Wk1	Wk2	Wk3	Wk4	Wk5	Wk6
1. Finish Proposal						
Specs						
Research						
Schedule	xxxx					
Architecture	_xxxxx					
2. Validate with Sponsor:						
Meet with Sponsor		___x				
Validate Website		xxx				
3. Do Detailed Software Design						
Data		___xx				
Processes			xxx			
Objects			___xx			
4. Coding and Implementation						
Interface Class and Headers				xxx		
PPF_Logic Prototype				__xxx	xxx	
Parallel_Wrapper					xxxxxx	x
PPF_Logic with Wrapper						xxxxx

Work Tasks	Wk7	Wk8	Wk9	wk10	Wk11	
PPF_Logic with Wrapper	xxx					
Statistics and Timer	__xx	x				
5. Finish Code						
Testing		xxxxx	xxxxx	xxxxx		
Revision		xxxxx	xxxxx	xxxxx		
Review		xxxxx	xxxxx	xxxxx		
6. Finish documentation						
Code Documentation					xxxxx	
Project Usage Documentation					xxxxx	

Chapter 10

Testing

10.1 Philosophy of Testing

Modules of the PPF will be tested at every step of the development process. Our design will begin with skeleton code for the entire PPF. As each module or subcomponent of the system is enhanced, the same black and white box tests and integration tests will be done to ensure that the project has maintained its integrity. During early stages of development, a oversimplified pseudo physics package will be used to test the correctness of the PPF code. As the project progresses actual physics problems with known results will be used as a basis for integrated black box testing.

10.2 Testing by Subcomponent

- **PPF_Interface** Testing for the interface will happen mainly during integration testing. The purpose of the PPF_Interface is to call the correct subcomponent when an interface method is called. After each of the subcomponents is tested for correctness, each of the interface methods will be tested to ensure the correct subcomponent is called in all preconceived scenarios.

- **PPF_Statistics** The PPF_Statistics class will be tested as an individual component and when integrated with the timer. White box testing will consists of creating debug files of all computations during trial runs from a driver program. A debug file dump will allow us to trace through the computations and verify correctness. Each statistical computation will be tested on data sets with known answers to ensure correctness. Illegal number usage (such as negatives) must be caught and filtered by the software and hence these numbers and other bounds conditions will be tested. Black box testing will consist of the PPF_Statistics package integrated with PPF_Timer. During these tests it will be assumed that PPF_Timer has already been tested. Tests will check final results after exhaustive runs from a driver file.

- **PPF_Logic** PPF_Logic may be the most difficult component to test. It must be tested individually and integrated with the PPF_Parrallel_Wrapper class. Testing should include the following:

  1. Ensure that domains and neighbor array lookup tables are loaded correctly. Testing should include loading a mesh and physically plotting out the neighbors to ensure correctness.
  2. Ensure that particles are being sent to the correct domain. Each particle tested must be identifiable by a unique ID so as to be traceable. Each test should be from a contrived driver file, where each particle destination is known before hand. This is one of the most critical portions of the PPF and correctness must be ensured to the best ability of the developers.
  3. Ensure that the PPF_Logic only sends particles when enough particles are present to justify the message passing overhead. This limit must be tested and may be a variable parameter within the code. White box testing of particle numbers in each particle send should be adequate for this.
  4. During times when a node is saturated particles must be written to disk to free up RAM. This can be tested by creating readable ASCII files for each particle written and read to disk. After a run from a driver file, each file should be checked to ensure that all particles which were written to disk were read and moved correctly. Obviously if the node crashes during a test, the PPF_Logic did not handle the freeing of memory correctly.

- **PPF_Parallel_Wrapper** Send and receive pairs must be tested for correctness. Currently the PPF is planned to be run only on homogeneous distributed systems, however if it is ported to a heterogeneous system, like a Linux beowolf cluster, byte ordering may be a problem. During test runs, each send and receive should dump a debug file detailing the exact byte data sent and received to ensure the data is not being corrupted. This must be rigorously tested on all architectures supported by the PPF.