

# A Parallel Software Development and Optimization Model for Scientific Application

Yeliang Zhang, Salim Hariri, Manish Parashar, Casey A. Meakin, Adam Burrows,  
Jingmei Yang, Madhuri Dwivedula  
High Performance Distributed Computing (HPDC) Laboratory  
Department of Electrical and Computer Engineering, University of Arizona  
Tucson, Arizona – 85721

## Abstract:

*Parallelizing personal codes developed by single users or small research group could be a difficult task. Careful analysis of the application in a systematic way is important to achieve efficient parallel algorithms. In this paper, we present a design model, the Parallel Software Design Model (PSDM) and show how it can be applied to parallelize and optimize a Supernova Neutrino Transport Problem. Our experimental results showed that by following approach we were able to achieve better parallel algorithm design and enhance application performance.*<sup>1</sup>

## 1. Introduction

Recent research in parallel software engineering has concentrated on developing parallel programming environments and parallel programming languages to make parallel programming easier and more appealing. [16, 17, 18, 19, 20]. The development activities addressed by these tools include problem partitioning, scheduling or mapping, automatic or assisted code generation, program debugging, and performance evaluation and fine-tuning. Few have focused on the development of parallel

software at the design stage. The design of a parallel application can be represented in several ways; formally, algebraically, graphically, etc. The more intuitive and unambiguous the model, the better the transfer of ideas from design to implementation. Graphical design models allow a software designer to represent the problem solving in a visible form suitable for analysis and evaluation. Since the most important criterion in evaluating a parallel application is how well it performs compared to the sequential implementation, it is crucial for a design model to be able to model timing flow at the design stage. A parallel application that performs poorly is said to have a performance bug [21]. The identification and elimination of program bugs, including performance bug, should be carried out at the earliest stage possible since the cost for removing a bug becomes much more expensive at later stages of program development. The traditional ad-hoc approach of implementation followed by fine-tuning to improve the performance of a parallel application is synonymous to purposely introducing a bug into the program at the design stage and debugging in at later stages. Most parallel programming environments provide performance tools that can be used to optimize the application after the design and implementation stages [16]. Such a “fix-it-later” approach is uncalled for from a software engineering point of

---

<sup>1</sup> Support for this work is provided in part by the Scientific Discovery through Advanced Computing (SciDAC) program of the DOE, grant number DE-FC02-01ER41184

view. Consequently, an important feature of a parallel design technique is its ability to model timing flow and find out the modular bottleneck. With this ability, the designer is able to analyze the performance of a parallel application prior to implementation. And if possible, the application could be redesigned to improve its performance and/or by utilizing less resources. Existing parallel software design models such as Conway's fork-join construct [7], Yau's modified petri-net [22] and distributed system design approach [23], Chen's event-driven approach [24], and De Marco's dataflow diagrams [25], all lack the capability to precisely model timing flow in a parallel algorithm.

In this paper, a design model is presented to optimize the parallel application with three steps: 1) Program Characterizing and Quantification 2) Parallelization Algorithm Design, and 3) Implementation and Optimization.

The organization of the paper is as follows. In section 2, we present a parallel software design model for the development of efficient parallel applications. We illustrate the use of the parallel design model to improve the performance of personal code in a case study in section 3. Section 4 concludes the whole paper.

## 2. A Parallel Software Design Model (PSDM)

In our design model, a processor is represented by a rectangle with timing flow on the vertical axis. The height of the rectangle denotes its total execution time. The width of the rectangle carries no information. Message flow is aligned between communicating processors to distinguish between communication time and computation or idle times. It is assumed that the communication

overhead associated with each message transfer is included in the communication time. The total execution time of the application is the elapsed time between the initiations of the first processor to the termination of the last processor. The model allows the designer to visually inspect the computation, idle, and communication times during the design stage and optimize them if possible.

The design model goes through three steps:

*Step 1: Program Characterizing and Quantification.* Given the initial problem/application that needs to be parallelized, the first step would be to use analytical or profiling tools to estimate the time complexity of each computational module.

*Step 2: Parallelization Algorithm Design.* Function and data partitioning can be used to design the appropriate parallel algorithm. This step analyzes the application data flow and the dependency between the application modules.

*Step 3: Parallel Design Implementation.* After we find the bottleneck of the application and analyze the data dependency of the modules, we can implement the parallel algorithm design identified in Step 2.

## 3. Case Study: Supernova Neutrino Transport Problem (SESAME code)

SESAME code is developed by the department of Astronomy and Steward Observatory, University of Arizona. It is a one-dimensional Lagrangean radiation-hydro code with general equation of state, transport and gravity. The transport method is multi-group, employs the Feautrier technique, uses the tangent-ray approach to resolve angles, is implicit in time, and is second-order accurate in space. The comoving transport equation in SESAME is solved

based on Eastman & Pinto algorithm [14]. The zeroth- and first-moment equations are iterated with the Boltzmann/transport equation using Accelerated Lambda Iteration (ALI) to speed convergence of the radiative transfer solution. This method is akin to the standard variable Eddington factor (VEF) approach, but no ad hoc flux limiters or artificial closures are necessary. The Feautrier scheme can transition to the diffusion limit seamlessly and accurately and the tangent-ray method automatically adapts with the Lagrangean hydrodynamics grid as it moves. In constructing the tangent rays, SESAME cast them from every outer zone to every inner zone. Hence, if there are 300 radial zones, the outer zone has 299 angular groups in each quadrant of the unit circle. Because of the spherical nature of the core-collapse problem and the need to accurately reproduce the angular distribution of the radiation field that transitions from the opaque (inner) to the transparent (outer) regions, such fine angular resolution is

useful, though computationally demanding, as radiation becomes more and more forward-peaked.

Hydrodynamics part couples Newtonian and Lagrangean predictor/corrector hydrodynamics scheme that use artificial viscosity for shock resolution.

We apply the PSDM steps that were discussed in section 2 to SESAME code.

### Step 1: Quantification

This step uses time measurement techniques to estimate the total execution time on various modules. From the analysis of the SESAME algorithm, we can divide the whole application into four major modules, *transe*, *transa*, *transm*, and hydrodynamics. Within transport modules, we can sub-divide them into two major parts, ray integration and index calculation and redistribution. We use measurement techniques and together with PGF profiling tool [26] to estimate the execution time.

Transe			Transa			Transm			Hydro	Others
32.8699%			33.7999%			33.1266%				
Ray integration	Index cal. Redis.	others	Ray integration	Index cal. Redis.	others	Ray integration	Index cal. Redis.	others		
69.58%	26.85%	3.57%	67.67%	26.11%	6.22%	69.05%	26.64%	4.31%		

Table 1: Execution Ratio for major modules

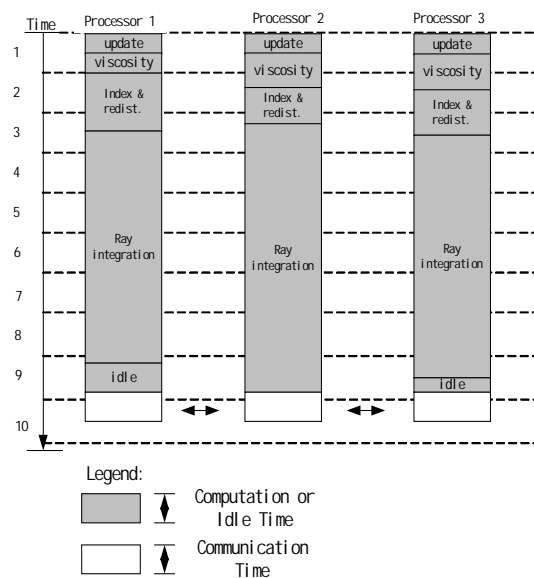
### Step 2: Analysis of parallelizable modules and Function/Data Partitioning

From table 1, we can see that the code sequentially calls *transe*, *transa*, and *transm* to calculate different neutrino species. By carefully examine the code, we found that these three subroutines are

independent except that *transe* initializes angular integration weights and other quantities that are needed when it is first called. We remove this dependency by letting every subroutine to call this initiation subroutine, and then apply function parallelism to *transe*, *transa* and *transm* functions.

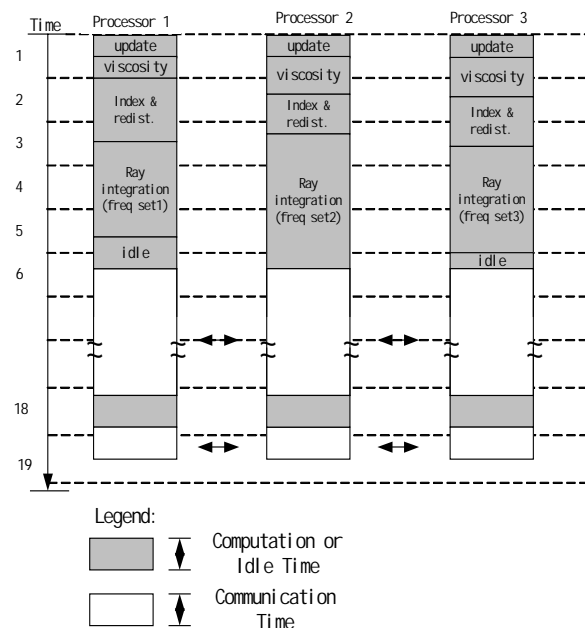
Each function (*transe*, *trnasa*, and *transm*) calculates ray-by-ray integration for all the frequencies. The ray calculation of each frequency is independent and therefore can be parallelized using data parallel model. We can partition the data set into chunks to reduce the communication time. In this scenario, we need to exchange more dataset to ensure that all the processors have the right copy of data for its transportation calculation. There are 3 dataset: *vn*, *un*, *surfint*, each of them has 2,400,000 double precision real numbers. Each processor needs to exchange all the dataset generated in ray integration.

Figure 1 shows the PSDM representation of parallel algorithm that involves calling the three functions in parallel while Figure 2 shows the PSDM representation of the parallel algorithm that partitions the frequency computations among the processors as discussed in Step 2.



**Figure1:** PSDM program representation parallel implementation of the three functions.

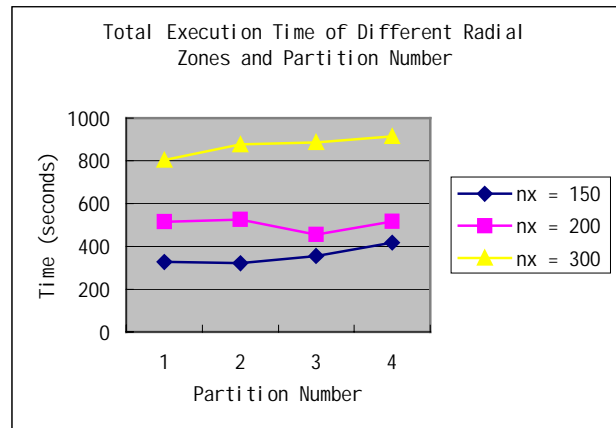
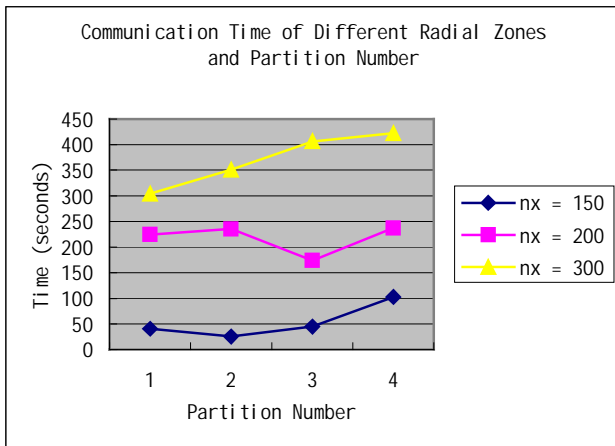
From PSDM analysis, we can see that we have two different ways to parallelize the SESAME code. The best performance will be achieved if we can use functional parallelism for the neutrino transport solution and data parallelism for the frequency integration. Table 2 shows the communication time and the total execution time for different data partitions and by using 6 CPUs with 300 time steps. Here *nx* is the number of radial zones solved by SESAME code. More radial zones represent more precise calculation and more computation load. Figure 3 is the graphic result of applying functional and data parallelism on SESAME code based on our PSDM analysis.



**Figur2:** PSDM program representation for implementing the frequency computations in parallel. (Assume each neutrino species use 3 processor to solve comoving equation, this figure is for the *transe* function)

	Parallel Partition#	1	2	3	4	5	Sequential (W/O parallelization)
<b>nx = 150</b>	Communication Time	2104.20	40.96701	25.48401	45.06698	102.5327	479
	Total Time	3358.01	326.903	321.7334	354.6542	418.3583	
<b>nx = 200</b>	Communication Time	2176.71	224.5511	235.4571	173.9196	237.3848	801
	Total Time	3695.86	515.8465	526.0094	455.5631	516.3366	
<b>nx = 300</b>	Communication Time	2361.13	304.0094	351.07	406.3966	421.8033	1271
	Total Time	4408.40	804.1778	875.6822	886.9063	914.6419	

**Table 2: Communication time and total execution time for different partition number and computation load**



**Figure 3: Communication Time and Total Execution Time of Different Partition Number and Computation Load (6 CPUs and 300 time steps)**

We can see from Table 2 that by using PSDM to choose the best partition number for the datasets, we can achieve 33%, 43% and 37% speedup when  $nx$  is 150, 200, and 300 respectively. We also observe that different computation load corresponds to different partition number to provide best performance from Figure 3. For example, when  $nx = 150$ , partition number 3 will give the least execution time which means at that point, the

communication and computation time are overlapping in the most efficient way.

Interestingly, we found that in our parallelization version, if we don't use any data partition technique, the communication time will consume most of the execution time. This is because extremely large arrays  $vn$ ,  $un$ ,  $surfint$  that have 2,400,000 elements respectively need to be distributed among the processors using *MPI\_Allgatherv*. When we are using

data partition, we surpass this problem by using *MPI\_Isend* and *MPI\_Irecv* instead of *MPI\_Allgatherv*. *MPI\_Allgatherv* has much more overhead than *MPI\_Isend* and *MPI\_Irecv* according to Pallas MPI Benchmarks [15] and with data partition, we can even reduce the overhead on *MPI\_Isend* and *MPI\_Irecv* by overlapping communication time and computation time.

#### 4. Conclusions and Future Work

In this paper, we introduced the Parallel Software Design Model (PSDM), that involves using measurement and profiling techniques to quantify the performance of the main modules, design an efficient parallel algorithm using the PSDM representations, and then implementing the designed algorithm..

We have applied the PSDM methodology to design an efficient parallel algorithm for a large sequential code (SEASME) used in astronomy. The implementation results proves that our approach can assist in the development of efficient parallel algorithms. We are currently applying the methodology to parallelize legacy codes using in genomic research.

#### References

[1] Kok K Kee and Salim Hariri, "PSDN: An Efficient Parallel Software Design Notation"  
[2] C.P.Wadsworth, "Getting Parallelism into Perspective", 2nd International Seminar on design and application of parallel digital processors, 1991  
[3] Robert Cypher and Eric Len, "The Semantics of Blocking and NonBlocking Send and Receive Primitives", 8th

International Parallel Processing Symposium, 1994

[4] Arun K Somani and Allen M Sansano, "Minimizing Overhead in Parallel Algorithms through overlapping Communication/Computation", Institute for Computer Applications in Science and Engineering, 1997

[5] Madhuri Dwivedula, Salim Hariri and Manish Parashar, "A software design model for parallel applications on heterogeneous systems", International Parallel and Distributed Processing Symposium: IPDPS 2002 Workshops

[6] C.D.Pham and C.Albrecht, "Optimizing Message Aggregation for Parallel Simulation on High Performance Clusters", 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 1992

[7] Françoise Baude, Denis Caromel, Nathalie Furmento and David Sagnol, "Optimizing Metacomputing with Communication-Computation overlap", High Performance distributed Computing Conference, 2000

[8] Gagan Agarwal, "Interprocedural Communication Optimizations for Message Passing Architectures", 7th Symposium on Massively Parallel Computation, 1999

[9] Ahmad Afsahi and Nikitas J Dimopolous, "Hiding Communication Latency in Reconfigurable Message-Passing Environments", 10th Symposium on Parallel and Distributed Processing, 1999

[10] Rajat P. Garg and Ilya Sharapov, "Techniques for optimizing applications"

[11] Robert a. Fiedler, "Optimization and scaling of shared-memory and message-passing implementations of the ZEUS hydrodynamics algorithm", Proceedings of SC97: High Performance Networking and Computing

- [12] Adam Burrows, Timothy Young, Philip Pinto, Ron Eastman and Todd A. Thompson, "A new algorithm for supernova neutrino transport and some applications", *The Astrophysical Journal*, 539: 865-887, 2000 August 20
- [13] T. Thompson, A. Burrows, P. Pinto, "Shock Breakout in Core-Collapse Supernovae and its Neutrino Signature", accepted to *The Astrophysical Journal*, April 2003
- [14] Eastman, R. & Pinto, P. 1993, *The Astrophysical Journal*, April 2003
- [15]<http://www.pallas.com/e/products/pmb/index.htm>
- [16] Topham, N., Ibbett, R., and Bemmerl, T., "Programming Environments for Parallel Computing", IFIP North-Holland, 1992
- [17] Saletore, V. A., Jacob, J., and Padala, M., "Parallel Computations on the CHARM Heterogeneous Workstation Cluster", Department of Computer Science, Oregon State University, Corvallis, Oregon 97331, in review for HPDC-3, 1994.
- [18] Sunderam, V. S., "PVM: A Framework for Parallel and Distributed Computing", *Concurrency: Practice and Experience*, Vol 2, No. 4, pp. 315-339, Dec. 1990.
- [19] Lusk, E., and Butler, R., "Portable Parallel Programming with p4", *Proceedings of the Workshop on Cluster Computing*, Tallahassee, FL, Dec. 1992
- [20] Terrano, A. E., Dunn, S. M., and Peters, J.E., "Using an Arcitectural Knowledge Base to Generate Code for Parallel Computers", *Communications of the ACM*, Vol. 32, No. 9, pp. 1065-1072, September 1989.
- [21] Carriero, N. and Gelernter, D., "How to Write Parallel Programs", The MIT Press, 1990
- [22] Yau, S. S. and Caglayan, M. U., "Distributed Software System Design representation using modified Petri nets", *IEEE Trans. Software Eng.*, Vol. SE-9, pp. 733-745, Nov 1983.
- [23] Yau, S. S. And Shatz, S. M. "An Approach to Distributed Computing System Software Design" *IEEE Trans. Software Eng.* Vol. SE-7, pp. 427-436, July 1981
- [24] Chen, B-S and The T. Y., "Formal Specification and Verification of Distributed Systems", *IEEE Trans. Software Eng.*, Vol. SE-9, NO. 6, Nov. 1983
- [25] De Marco, T., "Structured Analysis and System Specification", Prentice-Hall, 1979
- [26]<http://www.pgroup.com/tools/pgprof.htm>