





1994

# Parallel Satellite Orbit Prediction Using a Workstation Cluster

Stone, Leon C.



# PARALLEL SATELLITE ORBIT PREDICTION USING A WORKSTATION CLUSTER

Leon C. Stone
Shridhar B. Shukla
and
B. Neta *

Naval Postgraduate School
Code MA/Nd
Monterey, CA 93943
email: bneta@moon.math.nps.navy.mil

*Author to whom correspondence should be addressed.

**Abstract**

In this paper, the benefits of parallel computing using a workstation cluster are explored for satellite orbit prediction. Data and function decomposition techniques are used. Speedup and throughput are the performance metric studied.

The software employed for parallelization was the Parallel Virtual Machine (PVM) developed by the Oak Ridge National Laboratory. PVM enables a network of heterogeneous workstations to appear as a parallel multicomputer to the user programs.

A speedup of almost 6 was achieved when using 8 SUN workstations.

Keywords: Parallel computing, orbit prediction, domain decomposition, PVM.

# 1 Introduction

With the introduction of small, relatively inexpensive computers, a vast amount of computing resources are often left idle for a long period of time. A ship often has this characteristic. A ship's complement of computers is usually used for intermittent word processing or single dedicated computational tasks. With these computers networked together, a lot of unused CPU power is available. In order to tap into these unused assets, parallelization software tools have been developed such as PVM [1] or Linda [4]. These programs operate at the user level like an extra layer of operating system code.

In this paper we discuss the use of Parallel Virtual Machine (PVM) for parallelization. The program to be parallelized is the Naval Space Command's PPT2 satellite orbit prediction model. PVM is a software library, currently being refined, developed by the Oak Ridge National Laboratory (ORNL). It is a software system that enables a collection of heterogeneous computers to be used as a coherent and flexible concurrent computational system [1]. PVM was chosen because it is relatively easy to use, and is an emerging standard for software of its kind. It is currently available free of charge from ORNL and installation is relatively easy. PVM version 3.2 is used for this paper.

Parallelization could have been accomplished using a specific parallel multicomputer, such as the INTEL hepercube [2]. These systems tend to be large and expensive. While PVM may not accomplish the tasks as fast as, say, an INTEL iPSC/2 hypercube, (see Phipps et al [2]) the process execution times were satisfactory for the application tested. A speedup of almost 6 when using a cluster of 8 workstations was achieved.

In the next section, we discuss parallelization of PPT2 including a variety of domain decomposition schemes and give a preliminary results of our experiments on a small data set. In section 3, we discuss the results of our experiments with a larger data set and obtain the optimal number of input blocks to use along with speedup results.

# 2 Parallelization of PPT2

Currently the Naval Space Command tracks over 6000 Earth orbitting objects. With more and more countries entering space exploitation, and as the United States increases its emphasis on space communication, this data set of satellites will forseeably increase dramatically in

the future. These increases in the satellite catalog will increase the computational demands on the computer tasked with orbit prediction. If the NAVSPACECOM's orbital model's accuracy is increased or multiple calls to the orbit prediction algorithm are made for accuracy, or the number of objects tracked is increased, then the computational demands may be too much of a burden if the computer was a serial machine [2]. Given these computational loads, and the time dependency of the results, parallel processing of the catalog is a logical extension.

Given a program and its associated data set, there are two primary ways to process it in parallel. The program can be separated into individual sections (called control decomposition) with a processor dedicated to compute its respective part, much like a factory assembly line. The other method domain decomposition is to divide up the data set and send parts to many separate processors all running the same algorithm, but on different data. For PPT2, Phipps [2] showed that control decomposition is not efficient. We thus experiment with various ways of decomposing the satellite catalogue and distributing it to multiple nodes each propagating the orbit to several given times.

## 2.1 Decomposition Strategies

The basic algorithm for all of the decomposition methods used a master/slave strategy. For all the programs, there was one supervisor (master) node which decomposed the data set and distributed it to the worker (slave) nodes. Sending information requires the packing (by sender) and unpacking (by receiver) of data and buffer initialization. Each worker ran on a separate processor and sent its results to a gathering node which printed the results to a file and reported to the master when the process had completed for all satellites. Figure 1 graphically presents these relationships.

To get a general understanding of the decomposition requirements five decomposition strategies were developed. All the methods endeavored to minimize communication to computation ratio and to keep the worker processors busy as much as possible to increase speedup and efficiency. Each method is described below and denoted by ds1 to ds5 (for decomposition strategy).

ds1: Send/Request One at a Time

The supervisor initially sends one satellite to each individual worker node and waits for the workers to individually request another satellite. This method brought out the high PVM communications overhead which needed to be overcome for adequate speedup. Of course, in case a worker node is slow, this will ensure it will not get more data than it can process.

ds2: Send/No Request

The supervisor node for this routine sends one satellite at a time to each worker node in a round-robin fashion until the input file is distributed. This process reduces the communications overhead between the supervisor and workers, but it does not keep all the processors busy for a sufficiently long time, since the computation time is shorter than the time until the next data is received.

ds3: Send Block

For this scheme, the master divides the number, $S$, of input satellites by the number, $n$, of worker processors. The supervisor then sends a block of size $S/n$ to each worker. This is much more efficient than the previous two methods, but for $n$ greater than 8, the workers numbered eight and above were not getting data fast enough to notice effective processor computational overlap.

ds4: Send Half Block

Here, the master sends blocks of size $S/(2n)$ to each processor and sends another block of the same size again. The smaller blocks take less time to send.

ds5: Multiple Block

The above scheme, ds4, was modified to send a variety of block sizes. The master sends a block of data to each worker, then the worker extractred one satellite at a time from its input buffer and sent a block of results, equal in size to its input block, to the gathering processor. Sending blocks of data between processors vice one data element at a time, minimized the buffer manipulation which resulted in lower execution times.

# 3 Results of PPT2 with PVM

For preliminary experimentation, PVM was started on eighteen different workstations so measurements could be taken for one to sixteen working nodes. The workstations are SUN Sparc II and Sparc IPX having 40 MIIz processors and configured with 32 Mbytes of system memory. The workstations are connected by a 10 Mbytes Ethernet based network. The four schemes ds1 through ds4 were used with data sets of 600 and 1200 satellites. The programs were run ten times for each number of processors in order to get a good average time. The results for 1200 satellites are given in Figure 2. The figure shows a definite advantage in sending two input blocks of data (ds4) to each worker node over the other schemes.

The rest of our experiments are with decomposition strategy ds5 and a cluster of eight workstations. To determine the length of time required to run the parallel program, the execution time of each working node had to be determined. This execution time was broken down into three phases: setup, calculation and breakdown. During the setup phase, the worker waited for and received the next input block from the master. The calculation phase is the time it took PPT2 to execute the entire input block. The breakdown phase was simply the period in which the worker node packed and sent the results to the gathering node.

Using the variables defined in the following Table, Stone [3] has obtained expression for the setup time $t_s$, of the $i^{th}$ processor

$$t_s = i(C_f + C_{ps}S_b) + n_b(C_{upf} + C_{upps}S_b)\,. \tag{1}$$

The calculation time, $t_c$ is given by

$$t_c = T_{ppt2}S_b \tag{2}$$

and the breakdown time $t_b$ is

$$t_b = C_f + C_{ps}S_b\,. \tag{3}$$

Thus the total execution time of worker $i$ is

$$P_i = t_s + t_c + t_b\,. \tag{4}$$

| Variable | Definition | Value |
|---|---|---|
| $S$ | total number of satellites | 4800 |
| $t_o$ | node process initialization time | 5506.7 $\mu$s |
| $t_{gm}$ | time for gathering node to report to the supervisor the process is complete | 1300 $\mu$s |
| $n_b$ | number of blocks sent to each worker | 4 |
| $C_f$ | fixed communications time for buffer setup and network access for sending records | 6027.84 $\mu$s |
| $C_{ps}$ | communications time required to pack send one satellite record | 1264.52 $\mu$s |
| $C_{upf}$ | fixed communications time to unpack the input buffer | 132.98 $\mu$s |
| $C_{upps}$ | communications time to unpack one satellite record | 75.7 $\mu$s |
| $k$ | number of working processors used | 8 |
| $S_p$ | number of satellites sent to each worker $= S/k$ | 600 |
| $S_b$ | number of satellites per data block $= S_p/n_b$ | 150 |
| $T_{ppt2}$ | time for PPT2 to operate on one satellite record | 1850 $\mu$s |

The execution times for eight worker nodes, given four input blocks of data are shown in figure 3. The processor's phase times are described by two lines. The setup times are the lines on the processor number axis, and the execution and breakdown times are on the line one half space below the processor number. The blank space between the worker's breakdown phase and the next setup time is idle time. This idle time is clearly the result of the communication time required by the master to send blocks to all working nodes, taking longer than the execution time of PPT2 on each processor.

Given the fact PPT2 may need to be run several times for accuracy or tracking requirements, the calculation time must be scales by some factor $A$. This variable $A$ is the number of times PPT2 is executed on each satellite. The total execution time of worker $i$ is given by (Stone [3])

$$P_i = \begin{cases} t_s + At_c + t_b + (n_b - 1)(kt_b - t_u/n_b)\,, & A \leq \frac{(k-1)t_b + (C_{upf} + C_{upps}S_b)}{t_c} \\ t_s + n_b(At_c + t_b), & \text{otherwise} \end{cases} \tag{5}$$

The total execution time, $T_E$, of the parallel algorithm is

$$T_E = t_0 + P_K + t_{gm} \tag{6}$$

where $K$ is the number of workers used.

### 3.1 Comparison

In this section we used the above formula to compare the serial program to the parallel version using a data set of 4800 satellites and 8 workers (see Table 1 for the empirical values obtained from studying the performance of PVM on our SUN network). The total execution time of the serial program was taken to be simply $T_{ppt2}$ multiplied by the total number of satellites in the input file.

Figure 4 shows the final comparative results. The "theoretical" lines refer to using equation (6). The "actual" lines represent data obtained from running the serial program and ds5 (utilizing 8 workers) and a value of $A$ between one and ten. A block size of four was used for the parallel program. It is clear that the parallel program performed better than the serial program as the number of calls to PPT2 was increased. The theoretical and actual speedup are plotted in Figure 5. Note that theoretically a speedup of 6 (when using eight workers) may be possible. In all the runs, we were unable to achieve this theoretical result. One of the reasons is that it is virtually impossible to guarantee that the network is not used by others at the same time.

The comparative results using the actual catalog of 6795 satellites are plotted in Figures 6, 7. The actual results are closer to the theoretical ones in this case. A speedup of almost 6 using 8 processor was achieved.

# Conclusion

In this paper we demonstrated the effectiveness in reducing the overall execution time of updating the catalog of Earth orbitting objects by using a parallel algorithm. This algorithm was run using a parallelization software tool, PVM, on a loosely connected network of SUN workstations instead of a dedicated parallel multicomputer. A variety of data decomposition schemes were used. A speed up of almost 6 was achieved when using 8 workstations.

# References


1. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam, PVM 3 User's Guide and Reference Manual, Oak Ridge National Laboratory Technical Report ORNL/TM-12187, Oak Ridge, Tenn., 1993.

2. W.E. Phipps, B. Neta, and D.A. Danielson, Parallelization of the Naval Space Surveillance Satellite Motion Model, J. of Astronautical Sciences, **41** (1993), 207-216.

3. L.C. Stone, Parallel Processing of Navy Specific Applications Using a Workstation Cluster, M.S. Thesis, Naval Postgraduate School, Department of Electrical and Computer Engineering, Monterey, CA, December, 1993.

4. S. Ahuja, N. Carriero and D. Gelernter, Linda and Friends, IEEE Computer (August 1986).