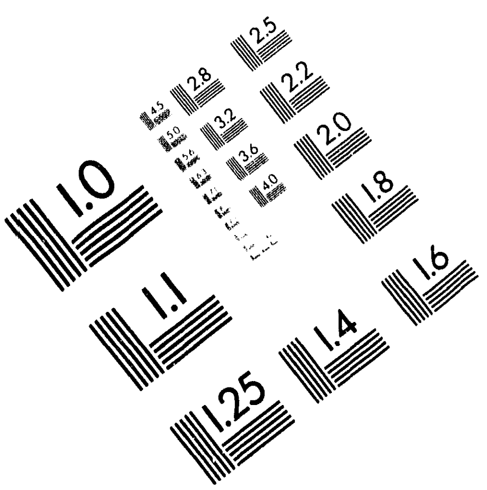
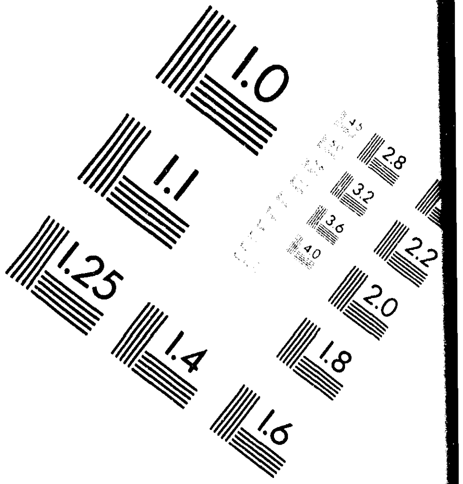
AIIM

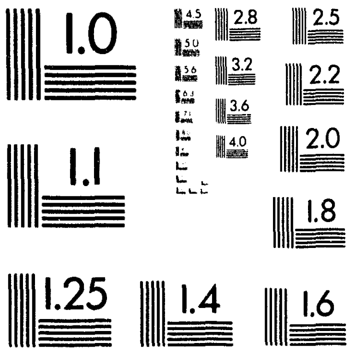
Association for Information and Image Management

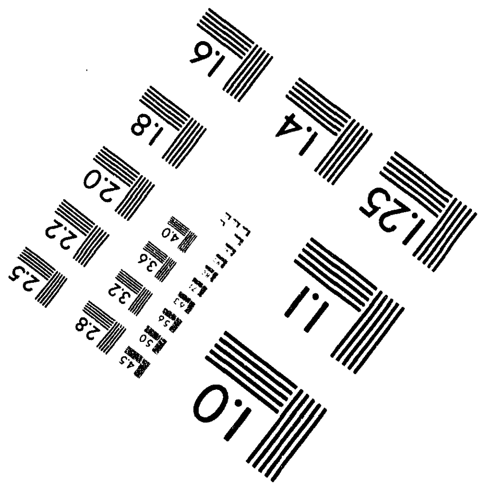
1100 Wayne Avenue, Suite 1100
Silver Spring, Maryland 20910

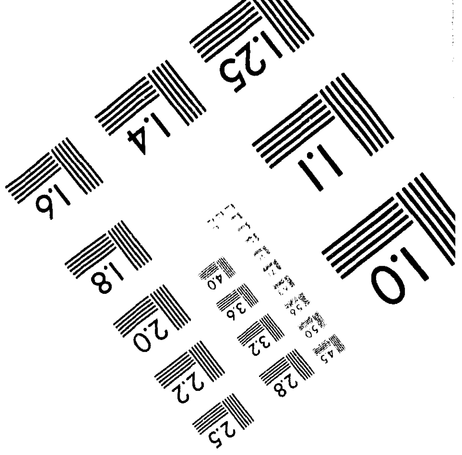
301/587-8202











MANUFACTURED TO AIIM STANDARDS
BY APPLIED IMAGE, INC.

ANL/DIS/CP-81788
Conf-940784--1

PARALLEL METHODS FOR THE FLIGHT SIMULATION MODEL

Advanced Computer Applications Center
Argonne National Laboratory
9700 South Cass Avenue DIS/900, Argonne, IL 60439-4832

Wei Zhong Xiong
Craig Swietlik

Tel: 708-252-3749
e-mail: wei@athens.eid.anl.gov

Abstract

The Advanced Computer Applications Center (ACAC) at Argonne National Laboratory has been involved in evaluating advanced parallel architecture computers and the applicability of these machines to computer simulation models. The advanced systems investigated include parallel machines with shared memory and distributed architectures consisting of an eight processor Alliant FX/8, a twenty four processor Sequent Symmetry, Cray XMP, IBM RISC 6000 model 550, and the Intel Touchstone eight processor Gamma and 512 processor Delta machines.

Since parallelizing a truly efficient application program for the parallel machine is a difficult task, the implementation for these machines in a realistic setting has been largely overlooked. Over several years, the ACAC has developed considerable expertise in optimizing and parallelizing application models on a collection of advanced multiprocessor systems. One of aspect of such an application model is the Flight Simulation Model, which used a set of differential equations to describe the flight characteristics of a launched missile by means of a trajectory.

The Flight Simulation Model was written in the FORTRAN language with approximately 29,000 lines of source code. Depending on the number of trajectories, the computation can require several hours to full day of CPU time on DEC/VAX 8650 system. Therefore, there is an impetus to reduce the execution time and utilize the advanced parallel architecture computing environment available.

ACAC researchers developed a parallel method that allows the Flight Simulation Model to be able to run in parallel on the multiprocessor system. For the benchmark data tested, the parallel Flight Simulation Model implemented on the Alliant FX/8 has achieved nearly linear speedup.

In this paper, we describe a parallel method for the Flight Simulation Model. We believe the method presented in this paper provides a general concept for the design of parallel applications. This concept, in most cases, can be adapted to many other sequential application programs.

1. INTRODUCTION

The ever increasing demand for computing power has driven the development of parallel computing technology. Computer simulation models have led many to believe that parallel computing technology is the key to realizing performance requirements [1]. The Advanced Computer Application Center at Argonne National Laboratory has developed a parallel program for a military simulation model. This simulation model is designed to be a Battle/Management/Control tool. It gives users the capability of modeling the major functions of a strategic defense system simulating the flight missile trajectories.

Simulating missile trajectories can involve an extensive amount of computation. The basis of such computation is a set of fourth-order ordinary differential equations describing the flight characteristics of a launched missile. The program is designed to be used in an interactive mode with the requirement of real-time performance. The sequential model can not meet these requirements. The parallel version of this simulation model provide better performance because it runs on a multiprocessor system, in which many processors can work on one problem or different problems simultaneously. However, there are issues that arise in developing such programs. The first issue is program dependencies. These dependencies must be removed from the sequential program in order to affect the parallel structure. The second issue is problem decomposition. We must decompose the problem into a number of subproblems. If there is not enough work to be done by the number of processors available, then a parallel program will show constrained speedup. This phenomenon is known as the Amdahl Effect [2]. The third issue is that the program may be I/O bound. The sequential models read and write from a number of files. In a parallel program, these I/O bound processes can place unnecessary constraints upon concurrent execution.

This paper documents a solution from our investigation on these issues and presents a parallel method for improving the state of the art of software applications for multiprocessor systems. We demonstrate the methodology for developing a parallel Flight Simulation Model.

2. THE FLGHT SIMULATION MODEL

The Flight Simulation Model is a program based on a DEC/VAX platform written in the FORTRAN language. To simulate flight missile trajectories, the program model initializes the user-defined missile parameters and trajectory parameters, then applies a series of processing functions. The missile parameters are used for flight calculations in the model. The missile flights can have up to three thrusted stages with each stage requiring certain parameters. The trajectory parameters include altitude, launch and target positions. The simulation model can then feed this information to the flight algorithms for computing the trajectory. The core of the flight algorithms is set of the Runge-Kutta method of ordinary differential equation that evaluate the three-dimensional rocket motion equations accounting the gravity $\vec{g}$, thrust $\vec{T}$, and drag $\vec{d}$ forces. The equations are as following [3]:

$$\frac{d\vec{r}}{dt} = \vec{v} \qquad \text{(Velocity)}$$

$$\frac{d\vec{v}}{dt} = \vec{T} + \vec{g} + \vec{d} \qquad \text{(Acceleration)}$$

where $$\vec{T} = T\vec{u} \qquad \text{(Thrust Acceleration)}$$

$$\vec{g} = -g(h)\frac{\vec{r}}{|\vec{r}|}$$ (Gravity Acceleration)

$$\vec{d} = -\frac{1}{2}\frac{q(h)\,A_{drag}C_D(v_m)}{m(t)}|\vec{v}_{rel}|\vec{v}_{rel}$$ (Drag)

$\vec{r}$ = Position Vector

q(h) = air density model

h = max($|\vec{r}| - R_e$, 0) = altitude

R_e = 6378.0 = earth radius

m(t) = $m_0 - m_t$ = mass function

m_0 = initial mass

A_{drag} = drag area of missile

v_m = Mach integer

$C_D(i)$ = Drag coefficients, i=1,2,3,...

$\vec{v}_{rel} = \vec{v} - \vec{v}_0$

$\vec{u}$ = Thrust direction unit vector, specified by flight control programs

The physical forces acting on the flight missile at instance point P is illustrated in Figure 1.

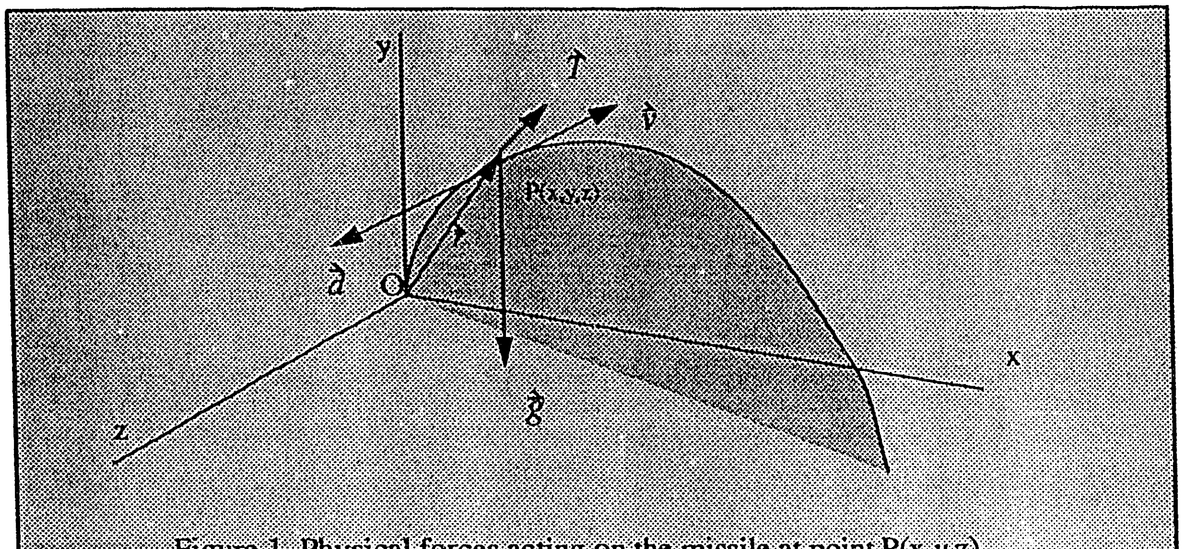


Figure 1 Physical forces acting on the missile at point P(x,y,z)

The simulation processing consists of a nested loop of missile types and threat rockets. Pseudo-code for a simplified processing flow is presented below [4]:

```
MAIN()
 Do for each trajectory
 if missile type change then
  initialize missile parameters and trajectory parameters;
 endif;
 call Flight Algorithm();
 enddo;
 ......
End

Subroutine Fight Algorithm()
.....
 call Runge-Kutta Routine();
 ......
 return;
End;

Subroutine Runge-Kutta Routine(y,dydx,n,x,h,yout,derivs)
integer n
real h,x,dydx(n),y(n),yout(n)
external derivs
```

/*Given values for the variables y(1:n) and their derivative dydx(1:n) known at x, use the forth-order Runge-Kutta method to advance the solution over an interval h and return the incriminated variables as yout(1:n). The user supplies the routine derivs(x,y,dydx), which compute the thrust, drag, and gravity forces, and returns derivatives dydx at x.*/

```
integer i
real h6,hh,xh,dym(50),dyt(50),yt(50)
hh=h*05
h6=h/6.0
xh=x+hh
do 11 i=1,n                    /*First step*/
    yt(i)=y(i)+hh*dydx(i)
11 enddo
call derivs(xh,yt,dyt)         /*Second step*/
do 12 i=1,n
    yt(i)=y(i)+hh*dyt(i)
12 enddo
call derivs(xh,yt,dym)         /*Third step*/
do 13 i=1,n
    yt(i)=y(i)+h*dym(i)
    dym(i)=dyt(i)+dym(i)
13 enddo
call derivs(x+h,yt,dyt)        /*Fourth step*/
do 14 i=1,n                    /*Accumulate increments with proper weight*/
    yout(i)=y(i)+h6*(dydx(i)+dyt(i)+2.0*dym(i))
14 enddo
return
end
```

```
Subroutine derivs(x,yt,dyt)
real yt(7),dyt(7),x
/* Rocket Motion Differential Equation */
Compute Gravity()        /*program segment for compute the gravity*/
Compute Drag()           /*program segment for compute the drag force*/
Compute Thrust()         /*program segment for compute the thrust force*/
return (dydx at x)
end
```

The initialization of missile and trajectory parameters accounted for as little as 5% of the execution time on sequential machines. The remaining 95% of time was spent executing the flight algorithm where the Runge-Kutta ordinary differential equations coupled with the rocket motion equations modeled the flight. As a result, the model is compute intensive and requires extensive CPU time. To address the performance problem, the task was established to convert this model from a sequential system to the parallel environment. This environment was the Alliant FX/8 vector multiprocessor system.

The first step of this task was to select a portion of the model to execute in parallel. The next step was to define a process structure and redefine the data structures for parallel execution. After implementation of the parallel Flight Simulation Model, tests were conducted to verify correct operation.

3. ALLIANT FX/8 SYSTEM

The parallel Flight Simulation Model were evaluated on Alliant FX/8 system. The Alliant FX/8 is a shared memory multiprocessor system with 8 computational elements (CEs) and 8 interactive processors (IPs). Each CE is a processor compatible with the MC68000 architecture. The processor contains floating point, concurrency, and vector instructions. A CE has vector processing capabilities and multiprocessor support. The 8 CEs of the system are crossbar connected to the shared, direct mapped cache. The cache is connected to the shared memory via a bus. The IP is a processor compatible with MC68000 architecture. The processor connects to disk, tape, terminal, printer, and communication devices through a multibus. The memory system contains 8 eight-megabyte memory unit for a configuration of 64 megabytes [5].

The operating system, Concentrix, is a multiprocessor Unix based on Berkeley 4.2 BSD. The system controls operations on all CEs, IPs, and memory units. Multiprocessing is realized by concurrency instructions. The operating system supports parallel FORTRAN and C compilers.

4. THE PARALLEL FLIGHT SIMULATION MODEL

Since the computation of trajectories is inherently highly parallel, because of the independent equations used to calculate the trajectory, the only dependencies among the processes were the initialization for each missile type. Therefore, the removal of the data dependent relations between the initialization process and computation process was a crucial issue for the design of the parallel version of the Flight Simulation Model.

The sequential execution profile also indicated during the Runge-Kutta computation, that the 3-dimensional equations of motion accounting for the three forces, gravity, thrust and drag were consuming the most CPU time. The model sequentially computes the value for thrust, gravity and drag. We identified that these computations can be parallelized due to their nature.

From above analysis, it was decided to take two approaches to parallelize the entire model. The first approach focues on the parallel numerical solution for Runge-Kutta method that would reduce total evaluation times. Second approach focus on the parallel simulation that would reduce total simulation times.

The Runge-Kutta method can be viewed as a task graph that is illustrated in figure 3. The perfromance profile indicated approximately 3 million functions of evaluation calls were invoked for each trajectory computation and each of them computed three physical forces sequentially.

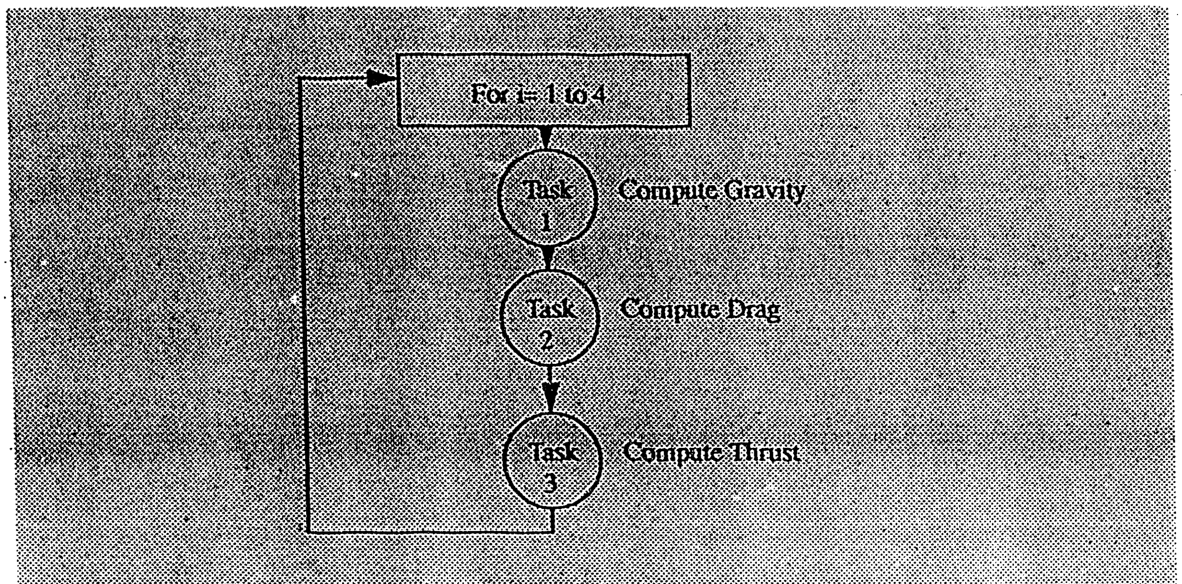


Figure 3 Task graph for Runge-Kutta Method

We converted this function derivation processing into parallel form so that three physical forces can be executed in parallel. The parallel task graph is shown in Figure 4.

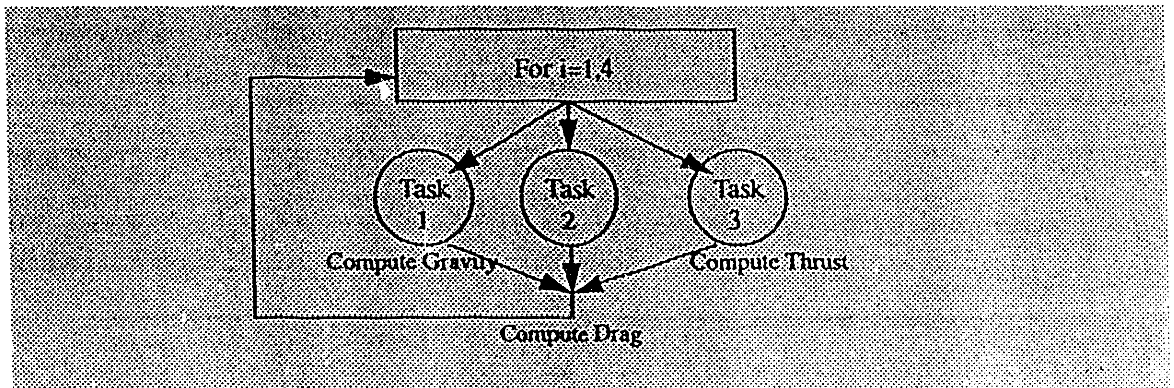


Figure 4 Parallel Structure for Runge-Kutta Method

The entire simulation processing can be viewed as a main program that contains a for-loop. This for-loop iterates from 1 to the final simulation trajectory. The flight missile launch side angle and target conditions is calculated at each loop iteration as fetching the information from the stored tables. Then each trajectory is computed independently through out the loop iteration. A flowchart of the simplified processing is shown in figure 5.

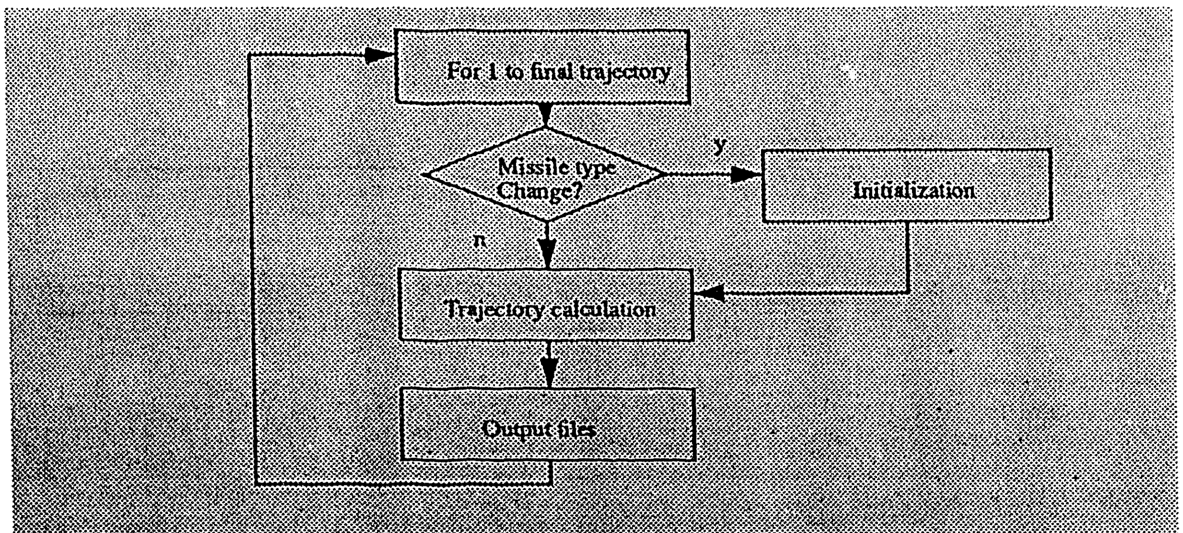


Figure 5 Simplified Sequential Processing Structure

We converted the above for-loop structure to a parallel routine by rewriting the missile initialization phase which reads the missile and trajectory parameters into memory and assigns a index key to each record. The index key for each record would become a key address for each trajectory computation processing for initializing the missile and trajectory parameters. In this way, the data dependencies are completed removed from the trajectory computation loop, so that trajectory computation can be distributed across multiprocessors and completed in parallel. Figure 6 illustrates the parallel structure for simulation processing.

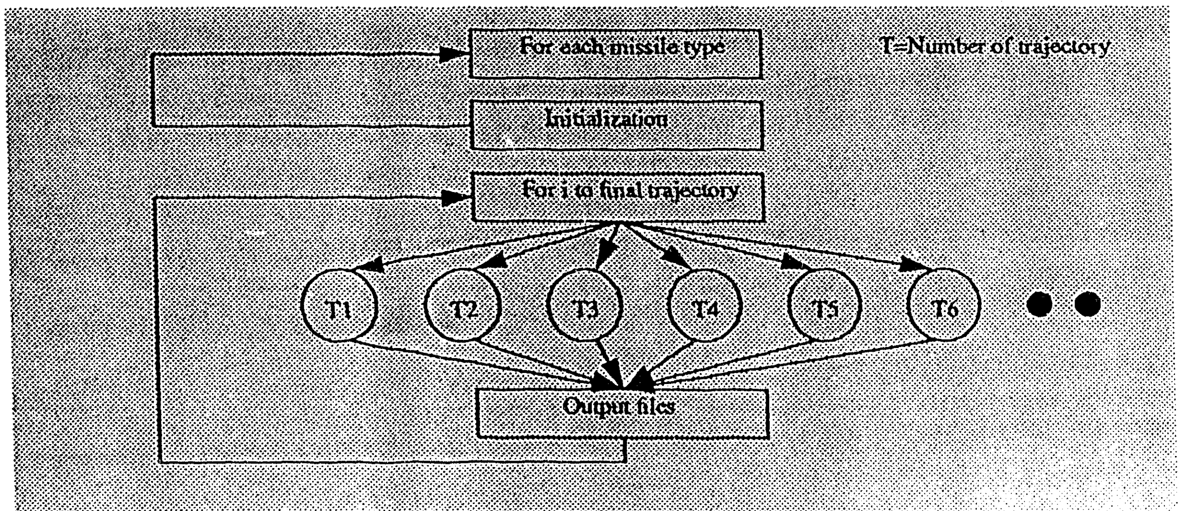


Figure 6 Parallel Simulation Structure

Figure 4 and 6 are combined in Figure 7 to show the entire parallel structure of the Flight Simulation Model. Since the initialization phase of processing is about 5 percent of the execution time, the remaining 95 percent of execution time that is devoted to flight simulation can be executed in parallel. We are using Amdahl's equation $speedup = \frac{1}{(1-f)+\frac{f}{r}}$ and basing the calculation on 8 physical processors, we obtain the $speedup = \frac{1}{(1-0.95)+\frac{0.95}{8}} = 6.7$.

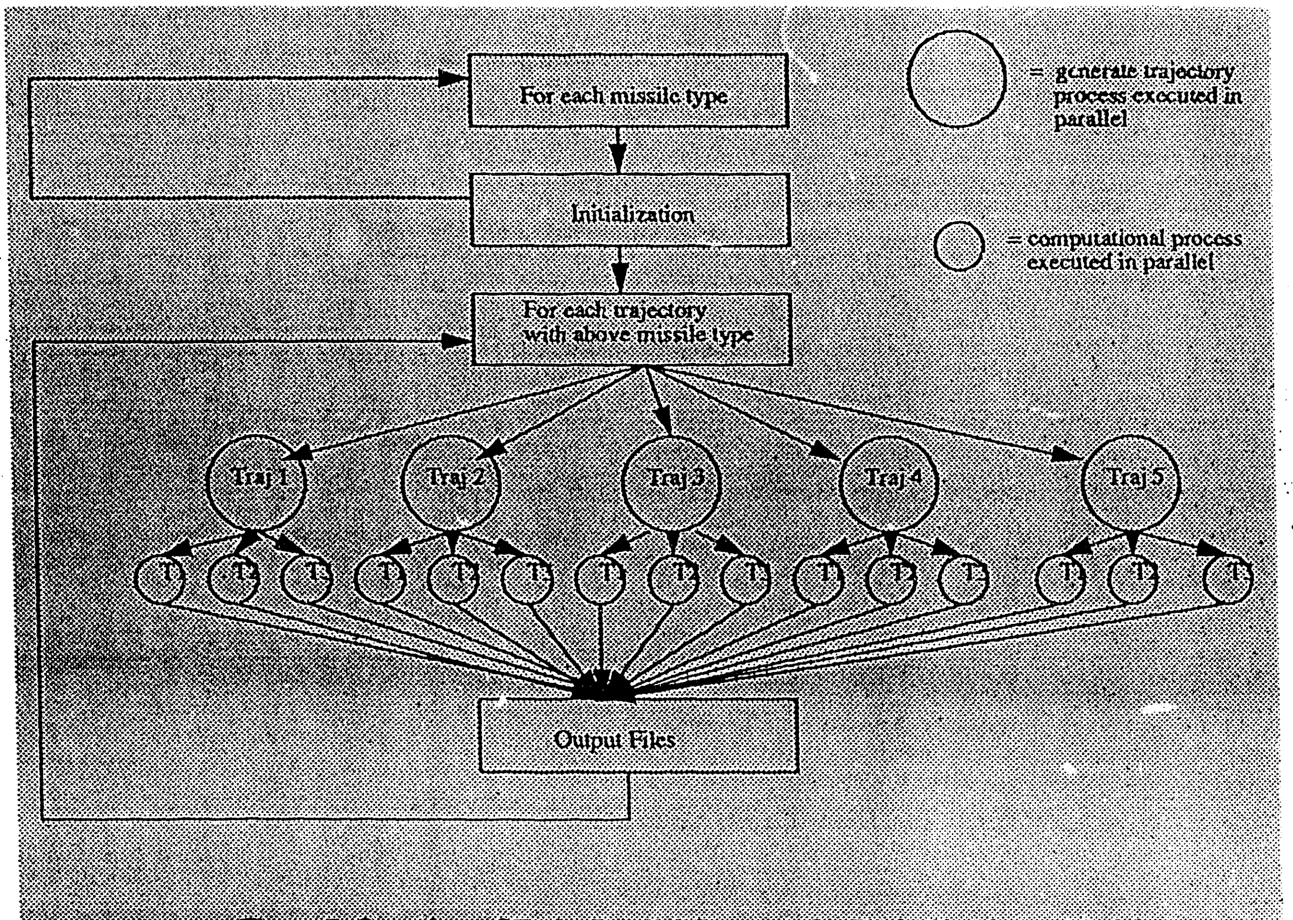

For each missile type
Initialization
For each trajectory with above missile type
Traj 1
Traj 2
Traj 3
Traj 4
Traj 5
T1
T2
T3
Output Files
= generate trajectory process executed in parallel
= computational process executed in parallel

Figure 7. Overview of Entire Parallel Structure for Flight Simulation Model

We used the following techniques to create a parallel Flight Simulation Model. To insure proper synchronization, we built an internal table with an index keys for the process identifier. This allowed each process to write data independently to an array held in memory. We also restructured several procedures from the Runge-Kutta algorithm which reduced the repeated overhead of multiple procedure calls. All I/O statement used for the diagnostics file where saved in the internal arrays and written out after the parallel computation was done.

5. PERFORMANCE

To measure the performance of parallel algorithms implemented on the multiprocessor system, the speedup and efficiency are the important metrics. The common definition of speedup achieved by a parallel algorithm running on n processors, is the ratio between the elapsed time taken by that multiprocessor system executing the serial algorithm and the elapsed time taken by the same system executing the parallel algorithm using n processors. The efficiency of a parallel algorithm running on n processors is the speedup divided by n.

In our parallel flight simulation model, we have carefully studied the output of the parallel version, before conducting any timing tests. The correct output of the parallel version was verified by comparing the results with the output from the serial version. The results of the two versions were verified to be identical.

Next, performance of the parallel version was measured using two sets of test data, one contained 5 missile types with 76 trajectories, another contained 5 missile types with 142 trajectories. The results from the experiments on the Alliant FX/8 are shown as below in table 1 for timing data and table 2 for efficiency data.

Table 1: Timing Data

Number of trajectory	Elapsed time (serial)	Elapsed time (parallel)	Speedup
76	8126.4 seconds	1480.6 seconds	5.6
142	15102.3 seconds	2398.4 seconds	6.3

Table 2: Efficiency Data

Number of trajectory	Speedup	Number of processors	Efficiency
76	5.6	8	70%
142	6.3	8	~80%

The above speedup factor is the elapsed time (wall clock time) of the serial version divided by the times for the parallel version. The result above indicates that an increase in the problem size would improve the speedup. This reflects the Amdahl effect.

6. CONCLUSIONS

In this project, we developed a parallel application specifically designed for execution on multiprocessor systems. The objective is to improve the state of the art in developing software for high-performance computing applications. This project has demonstrated the feasibility of parallel processing technology at application level with respect to performance of the software. This project has also demonstrated an innovative application of parallel processing techniques to solve differential equations using a fourth-order Runge-Kutta method.

Parallel processing is still a highly experimental science in which the designs of both user and system programs are undergoing simultaneous study. To gain further insight into the behavior of parallel programs, experiments can be conducted to address issues such as implementing the parallel Flight Simulation Model upon a distributed memory system such as the Intel Touchstone Delta System [6].

The idea of formulating parallel algorithms in terms of processes that pass messages back and forth, but share no memory, is very attractive. In our approach, the parallel structure of the model is well suited to this type of paradigm. If this abstraction is chosen, an algorithm can be implemented on an extremely broad class of machines with minimal differences in the actual code, thereby enhancing portability.

ACKNOWLEDGMENTS

We are grateful to Peter Campbell, Lou Kvitek, and Roger Hanscom for supporting much of this project.

* This work was supported under a military interdepartmental purchase request from the U.S Department of Defense, J-8 Directorate of the Joint Chiefs of Staff, through U.S. Department of Energy contract W-31-109-Eng-38

REFERENCE

[1]. CORBIN and LISA, *The supercomputer: Faster than a speeding Machine,* Government Executive, Sept., 1993, pp. 58-60

[2]. Q. J. MICHAEL, *Designing Efficient Algorithms For Parallel Computer,* McGraw-Hill Book Company, 1987, pp. 17-18

[[3]. *Technical Reference Manual,* Mission Effectiveness Model Version 11, United States Air Force Head Quarter Space Systems Division, September, 1989, pp, 2-4

[4]. T*hreat Version 10.B* (Threat Flight Routine), Fortran Program Document, Science Application International Corporation, March, 1 1988

[5]]. *Using the Alliant FX/8,* Argonne National Laboratory/MCS-TM-69, Revision 3, Argonne, Illinois, October 1988, pp. 1-3

[6]. *Touchstone Delta System User's Guide,* Supercomputer Division, Intel Corporation, Oct., 1991, pp. 1-7