Optimizing Mars Airplane Trajectory with the Application Navigation System

May 2004

Michael Frumkin,
NASA Advanced Supercomputing Division
M/S T27A-1, NASA Ames Research Center
Moffett Field, CA 94035-1000
frumkin@nas.nasa.gov
Derek Riley
Vanderbilt University
Nashville, TN 37235
derek.riley@alumni2004.wartburg.edu

Abstract

Planning complex missions requires a number of programs to be executed in concert. The Application Navigation System (ANS), developed in the NAS Division, can execute many interdependent programs in a distributed environment. We show that the ANS simplifies user effort and reduces time in optimization of the trajectory of a martian airplane. We use a software package, Cart3D, to evaluate trajectories and a shortest path algorithm to determine the optimal trajectory. ANS employs the "GridScape" to represent the dynamic state of the available computer resources. Then, ANS uses a scheduler to dynamically assign ready tasks to machine resources and the GridScape for tracking available resources and forecasting completion time of running tasks. We demonstrate system capability to schedule and run the trajectory optimization application with efficiency exceeding 60% on 64 processors.

Keywords: dynamic scheduling, performance, navigation, trajectory optimization.

1 Introduction

Planning of future NASA missions involves a number of applications working in concert to provide an admissible or an optimal planning decision. These applications simulate various aspects of the anticipated mission environment, evaluate various scenarios of achieving mission goals, and evaluate multiple configurations of the vehicles to be involved in the mission. The quantity of applications, scenarios, and configurations could be significant and would require a substantial amount of computer resources and human effort to perform the calculations. To reduce the time required for the simulation and evaluation of various scenarios and configurations we have developed an Application Navigation System (ANS).

The ANS works with applications composed of communicating tasks. It uses a task graph representation of such applications. This model can be used to represent many data flow applications c.f. [2]; ANS makes scheduling decisions depending on the state of the tasks and the state of the resources of the distributed system. It takes into account proporties of the distributed resources such as latency of the communications and bandwidth of the network. ANS performs an automatic characterization of the resources, extrapolates the application's performance to the available resources, and assigns the tasks to the best resources with a goal to minimize the application turn around time. In [8] we have shown that this approach reduces some applications turnaround time by 25%-35%.

In this paper we demonstrate that ANS can efficiently manage the problem of optimizing a trajectory of a martian airplane, Section 5. The plane has a base and a number of fly-over targets located in a canyon on Mars. The goal is to find the most energy efficient trajectory that starts at the base, visits all the targets and returns to the base. To solve this problem we choose a number of intermediate points in the martian airspace and calculate the optimal trajectories between each point and its closest neighbors. Then we use a combination of the shortest path algorithm and a heuristic traveling salesman algorithm to find the most efficient trajectory. For calculation of the most efficient trajectory between a pair of points in the martian airspace we use the *Cart3D* package [3].

The architecture of the system is shown in Figure 1. We describe the interaction between servers and navigators in Section 3.1, and the acquisition and use of the GridScape in Section 3.2.

The results in Section 6 show that our navigation system achieves 60% efficiency relative to the optimal running

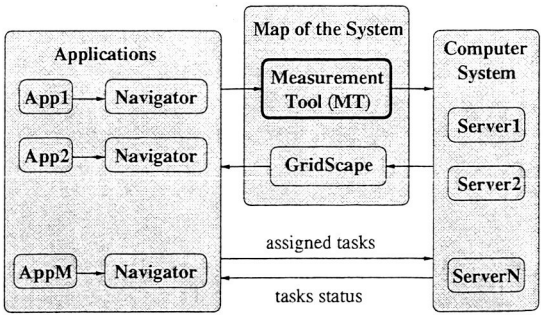

Figure 1. The Architecture of our Navigation System. Each navigator executes the iterations of the Navigational Cycle (see Figure 2).

time of the tasks without dependencies. In our experiments we used 5 machines containing 1952 processors and having peak performance of 1.5 TFLOPS, Table 1.

As an abstraction of system resources we use a *GridScape*, a map of the system resources represented by a directed graph, where each machine and router is represented as a node of the graph with an attached list of machine resources and a history of their dynamics. Communication links between machines are represented by the arcs of the graph labeled with the observed bandwidth and latency of the links. The initial information about the GridScape is acquired during installation of the benchmark/application servers (Figure 1). The dynamic component of the GridScape is measured by means of the NAS Grid Benchmarks (NGB) [9] during the monitoring, and by requests of the navigators. The benchmarks represent typical activity of Computational Fluid Dynamics applications, and their performance results may be used as an indicator of the efficiency of using the computer resources for these applications.

2 Related Work

2.1 Scheduling

Scheduling in a grid environment has been a popular topic of research for several years. One type of grid application is the so called parameter-sweep application comprised of many independet experiments. In [4] the authors discuss the methods of scheduling a dynamic parameter-sweep application on a computational grid. In our case the scheduler must deal with large numbers of dependent tasks with complex I/O relations. Fortunately, our main application *Cart3D* allows us to compose the tasks so that they can be executed in a data flow manner: a task can be launched as soon as all its input data have arrived.

To manage dependencies between the tasks, we are using proposed in [5] lists to keep track of which nodes can be executed and which nodes cannot. The authors required the scheduler to update the "ready nodes" list after every local schedule action it performs. They also used a critical path scheduling algorithm to improve the performance of their tasks. We do not use the critical path algorithm for two reasons. First, large numbers of paths in our graphs are close to critical and, second, the length of a path can only be calculated when all tasks have finished. Also, our tasks are all virtually the same size and require the same amount of computing power, so they are somewhat more predictable than the tasks mentioned in other research papers.

Scheduling data flow graphs is an NP-complete problem unless gross assumptions are made to simplify it. Most other research on data flow graphs conclude that simplifying assumptions are detrimental to the quality of the scheduler. Memory, processor, and cluster architecture are generally accepted to be too unpredictable to simplify the scheduling problem [10]. However, in our case, our simplifying assumptions have been tested and have been shown through our experiments to be reliable and predictable. This is most likely the case because we are using similar, batch scheduler controlled systems to avoid interference with other users.

Two models of data flow graphs are generally accepted for modeling the tasks and interconnections: task interaction graphs (TIGs) and task precedence graphs (TPGs) [10]. The TIG model is generally used for static scheduling and the TPG model is used for dynamic scheduling. In this paper we use the TPG model, which is also referred to as the directed, acyclic graph (DAG) model. When scheduling a DAG, it is important to decide whether to sacrifice efficiency of the scheduler to find an optimal solution or to sacrifice the optimality of the solution for efficiency of the scheduler. As our initial scheduler we used simple but efficient scheduling method that achieved about 60% of the efficency of the schedule (relative to the optimal schedule of the tasks without dependencies).

2.2 Mars Airplane

The dream of a Mars airplane has been pursued and researched by many different scientists throughout the last decade. Many felt that an unpiloted vehicle would be unable to make the judgement calls necessary for a successful flight, but recent demonstrations of unpiloted flying vehicles on Earth has eroded their theorized evidence. It is important that a flying vehicle be unpiloted because it makes the vehicle lighter, and in the case of Mars, it also allows the vehicle to conduct research long before humans can get there.

A flying vehicle on Mars is important because the wheeled rover missions are limited to a small area with relatively simple terrain. An airplane would allow researchers to cover much more distance and analyze more difficult terrain. One of the most interesting areas on Mars to researchers is the Valles Marineris, which is a canyon that is longer than 1860 miles and up to 5 miles deep. Scientists hope that the steep canyon walls will provide clues on the sedimentation of the crust and possibly existence of life on Mars.

In 1998 NASA offered a grant to an organization that came up with the most compelling space exploration proposal. Several of the 29 proposals were for aircraft that would explore Mars. Even though none of the missions have been fulfilled, there has been a significant amount of research that has gone into designing an airplane that will fly on Mars [14].

All of the researchers who have worked on the concept of a Mars airplane have had to deal with the same problems: Mars' thin atmosphere and its lack of abundant oxygen. The thin atmosphere is equivalent to flying about 100,000 or more feet above Earth, which has been accomplished, but only by experimental, rocket-powered craft. To fly in thin air, most of the designs have been developed to be as light as possible with large wing surfaces. Since the gravity on Mars is only 1/3 the gravity on Earth, the large wing structures work quite well. Most designers have also decided to simplify the control surfaces to further decrease the weight and complexity. It is important to reduce the weight because every extra pound sent into space costs a significant amount of money.

The lack of oxygen requires designers to find propulsion methods unlike those most commonly used on Earth. Many propulsion systems have been considered for a Mars airplane, but most designs have decided to use an electrical/propellor system. The benefit to an electrical system is the fact that solar panels can be used to recharge the batteries and extend the useful life of the craft. The solar electrical power system works well on Spirit and Opportunity rovers. Another proposed propulsion system uses hydrazine, which also requires no oxygen to provide power. Whatever the researchers have decided to use for power, they are all interested in making the craft have the least amount of drag and use the least amount of fuel.

The proposed Mars airplane will be able to collect a wealth of scientific data. It can be designed to hold a gravity gradiometer, magnetometer, electric field meter, laser altimeter, several optical and infrared cameras, and some deployable probes. The data collected would be sent directly back to Earth to be analyzed, so the craft would have to have the ability to communicate to Earth without a base station like many other missions [13].

3 The Application Navigation System

3.1 The Navigational Cycle

In our navigation system, tasks are assigned to the machine resources by the navigators, Figure 1. The decision to accept or reject an assigned task is performed by a server based on it's own criteria such as task priority and availbale resources. The navigators run on launch machines, while the servers are running on compute engines. For a given application, a navigator performs the following functions:

- Obtains a list of tasks that are ready to be executed;
- Consults the "GridScape" to find resources that are able to execute these tasks;
- Submits the tasks to the resources that will provide the fastest advance of the application;
- Repeats this sequence until all tasks are executed.

In order for a navigator to accomplish these functions, it must understand an application's requirements and know the current state of the system resources.

In our computational environment, the application description includes the application performance model (expected execution time, parallel efficiency, memory size, size of I/O data). This performance model is based on several runs of the tasks on an average (not extreme) initial geometry. Despite the fact that convergency of Computational Fluid Dynamic codes may significantly vary depending on small changes in initial conditions, during our runs we observed very consistent, predictable convergency of the *flowCart* solver.

To describe the system resources, we use the GridScape which lists the capabilities of the machines and the interconnections between them. When deciding to submit a task, a navigator uses the GridScape to match task requirements with abilities of the current resources. It estimates the time it will take to execute the task and assigns to it the number of processors that minimizes the application turnaround time. In summary, the navigator performs the routine of the *navigational cycle*, Figure 2. A detailed description of the ANS can be found in [8].

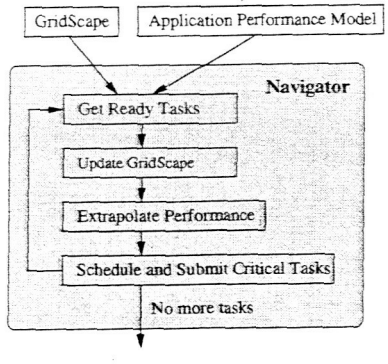

Figure 2. The Navigational Cycle.

3.2 The GridScape

The GridScape serves as an abstract description of grid resources that represents the current state of computing resources. The navigators use it to make submission decisions, while the servers use it to qualify submitted jobs. As a result, the quality of scheduling and the overall efficiency of the navigation system depend on how well the GridScape is synchronized with task submissions and changes in the state of grid resources.

We use three ways to update the GridScape to achieve a good synchronization. First, each server updates the GridScape when it changes the state of a task to (or from) Running, by subtracting from (or adding to) the GridScape the resources used by the task. Second, each monitor periodically updates the GridScape based on timings of the NAS Grid Benchmarks. Finally, each navigator can request an update to the GridScape. The details of the GridScape acquisition process is described in [8].

4 The Martian Mission Site and Environment

The site of the mission at 278.8° EL and 8.3° SA on the Martian surface is shown on Figure 3 (left pane) and the elevation map is shown on Figure 3 (right pane).

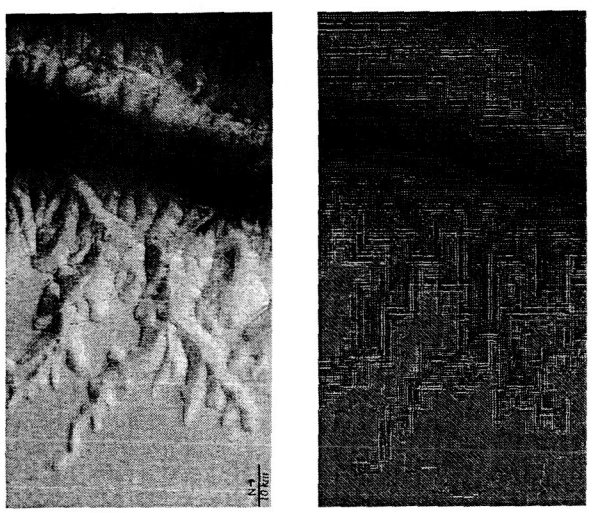

Figure 3. A stereo map (left pane) and elevation map (right pane) of the Martian airplane mission area.

4.1 Atmosphere

The atmosphere of Mars is significantly different from the atmosphere of Earth. Carbon Dioxide comprises about 95 percent of the atmosphere, and Nitrogen is second most common at 3 percent. The density of the Martian atmosphere is also significantly less than that of Earth's. The air pressure on Earth averages 1000 millibars, whereas the pressure on Mars is about 8 millibars. It fluctuates throughout the four seasons from a low of about 6.5 millibars in summer to a high of about 10 millibars in winter. The low air pressure makes lift and maneuvering more difficult for aircraft.

The temperature is also fairly variable on Mars because of the thin atmosphere. The surface temperatures range from 145 degrees Kelvin to almost 300 degrees Kelvin. There is no ozone shielding, so the UV rays from the sun can make the local temperatures less consistent. Even though there is very little water on the surface of the planet, clouds of water vapor commonly form. Dust storms are also common and can last for many days at a time. All these factors contribute to highly variable temperature and pressure conditions. The speed of sound is a value that is critical to the performance of an aircraft, and is directly related to the temperature and pressure. Therefore, an estimate will be used for both values to give an approximate idea of a normal flight on Mars.

Mars has another physical property that must be acknowledged to accurately predict the flight of an aircraft. Gravity is a force that must be opposed to fly an aircraft in close proximity to any planet. Mars's surface gravity is also about one third of the strength of Earth's. The lesser gravity will make the aircraft more efficient when flying, but it also affects an aircraft's ability to take off and land. With less force on the wheels traditional braking methods must be updated and augmented. [6]

4.2 The Martian Flyer

The Martian Flyer[1] (MF) is a delta-plane comprised of a wing-integrated body, two movable elevons, and two fixed winglets, Figure 4. The flier was designed for data-gathering missions on the surface of Mars. The flier has been designed for a cruise speed of about Mach 0.2. The elevons can be adjusted individually from deflections of 10 degrees down to 20 degrees up to change the pitch, roll, and yaw of the aircraft. The winglets were added to reduce drag and increase lift. Because the design for the Martian Flier only exists in data format on computers, it has not been physically tested; however, it has been through rigorous virtual testing using the Cart3D package. The Cart3D package allows users to adjust the elevon settings, Mach number, and other environmental characteristics to simulate conditions on Mars.

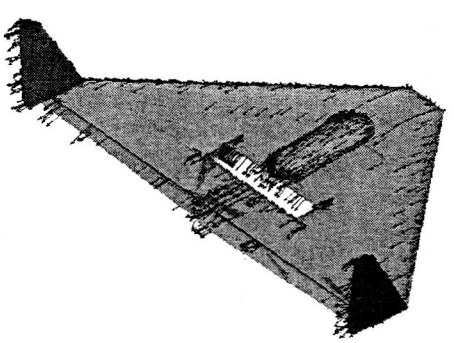

Figure 4. MF configuration with the right elevon down and a flow around it obtained with *Cart3D*.

[1] The MF geometry was provided by Michael Aftosmis group

4.3 The Trajectory Space

The MF starts from a base on the Mars surface, flies over a number of target points in the canyon, and returns back to the base while spending as little energy as possible to overcome the drag. The flyer takes advantage of the air updraft along the warm slopes of the canyon while avoiding the downdraft along the cold slopes. We estimate the updraft as a function of the amount of the solar radiation obtained by the slope during the current Martian day. It is a simple function of the angle between the normal to the surface and direction to the Sun[2].

As an approximation of the space of all possible trajectories we create a mesh of points at an elevation of H feet above the surface and connect each point with the nearest neighbors by edges, see Figure 5. Each trajectory consists of parts of edges and parts of the circular arcs connecting the edges (not shown in the picture). It can start/end at the base, way point, or a target point. The vector of the MF velocity at the beginning of each segment is the calculated velocity at the end of the previous segment. The optimal trajectory is one that minimizes the drag along it.

5 Trajectory Optimization

To find the optimal trajectory for the MF we need to perform a complex data flow management. We formulate this management problem as scheduling of the tasks representing flow calculations along all fragments of the trajectory. The dependencies between pairs of tasks are derived from the trajectory continuity condition.

5.1 The Cart3D Package

The *Cart3D* package [3] is a production NASA package used for high-fidelity inviscid analysis in conceptual aerodynamic design. It performs CFD analysis on complex geometries. A data flow graph of the *Mars Flier* test case is shown in Figure 6. It encapsulates six executables written in FORTRAN and C. The package includes utilities for geometry import, surface modeling, surface intersection, mesh generation, flow simulation, and analysis.

The geometry of an aircraft in *Cart3D* is represented as a collection of components, called a *configuration*. The configuration is defined by the triangulation file (*.tri) which describes the geometry of the aircraft. The triangulation file is first run through a program called *intersect* which produces a `preSpec` file. The `preSpec` file and the `input.c3d` file are what the *cubes* program requires for execution.

The flow simulation starts with mesh generation by *cubes*. *Cubes* produces topologically unstructured, adaptively refined, Cartesian meshes around the configuration. *Reorder* reorders the meshes (Mesh.c3d) produced by *cubes* using a space-filling-curve ordering. *FlowCart* takes these re-ordered meshes and partitions them on-the-fly onto the assigned number of processors. *MgPrep* is the mesh coarsening module which creates coarse meshes from an initial input mesh. These meshes are used in *flowCart* for multigrid convergence acceleration. The flow simulation in *flowCart* is a scalable, multilevel solver for the Euler equations governing the inviscid flow of a compressible fluid. The *clic* program is used to analyze the simulated flow.

In our "Mars Flyer" test case *cubes* created a mesh with about 825,000 cells. *FlowCart* performed 100 iterations on the original mesh, 92 iterations on the first and second levels of refinement, and 8 iterations on the third level of refinement. There were 180,000 cells produced, and the final efficiency for 16 processors was estimated at approximately 70-75 percent.

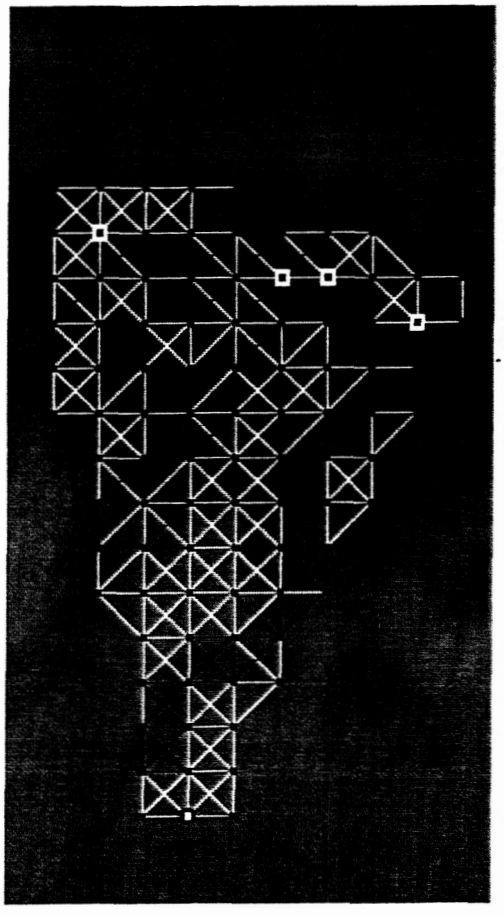

Figure 5. Approximation grid for the trajectories of MF. White edges indicate edges with updraft and black edges indicate edges with down draft.

5.2 The Wave Front Algorithm

Predicting a full flight trajectory on Mars is quite complicated. Breaking it down into smaller pieces ensures that we will test most possibilities; however, the pieces should

[2]Currently we ignore effect of shadows on the updraft.

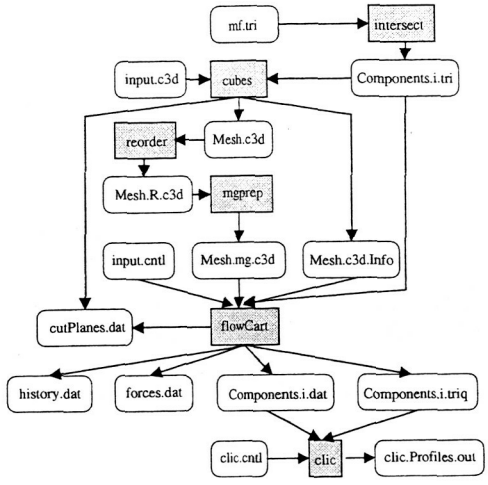

Figure 6. The Mars Flyer test case of the Cart3D package. Grey boxes show executables, other boxes show I/O files.

not be tested independently because a flight path depends on the incoming trajectory from a previous path. Therefore, our wavefront algorithm ensures that each trajectory waits to be calculated until the optimal incoming trajectory is found. Once the optimal incoming trajectory is established, the next trajectories can be tested to find the subsequent optimal trajectories until a trajectory from the home base to the desired destination is found.

The problem of finding a tour of minimum length that visits given graph vertices is known as the Traveling Salesman Problem. The general problem is NP-hard, however, the 3-dimensional Euclidian problem can be solved approximately in polynomial time by Arora's algorithm. In our case, there are only 4 graph vertices to visit so that even complete enumeration looks efficient. However, the length of an edge in a tour (represented as work of a drag along it) depends on the previous edge of the tour (due to the continuity of the MF trajectory) so that our problem can not be directly reduced to the salesman problem. Therefore, we use the heuristic wave-front algorithm to solve the trajectory optimization problem .

We define depth of a vertex (way points or the targets) of the trajectory graph as the minimum number of edges on a path connecting the vertex with the base. Obviously, vertices of depth d are connected only with vertices of the depth, $d-1$, d, and $d+1$ The algorithm starts by sending copies of the MF state from the base along edges to the way points of depth 1. Each copy is processed by *Cart3D* to calculate drag along the edge. Then, on even iterations it sends copies of the MF state from all way points of depth d along edges to points of depth d. On odd iterations it sends copies of the MF state from all way points of depth d to points of depth $d+1$. The algorithm records the path of minimal energy from the base to the current way point. To maintain continuity of the trajectory the algorithm takes the speed and altitude of the MF on the last edge of the path and uses them as initial conditions of the copies of the MF state sent to the other way points. Then assuming that the velocity vector is constant along the segment, the drag is computed using *Cart3D*. If newly incoming edge belongs to a path of smaller energy then the current path is replaced with the more efficient one. The iterations continue until all targets are visited.

Then the algorithm takes the speed and altitude of the MF on the optimal path incoming to each target as initial conditions for the outgoing edges and repeats the iterations using the targets as a new bases. As a result, we will get pairwise shortest paths among the base and all targets. Then we use the "go to the nearest unvisited neighbor" heuristic to build a a trajectory from the base that visits all targets and comes back to the base. This path may have discontinuities in velocity at some targets. To compensate for the discontinuity we compute a maneuver that transits the MF between the trajectories in these targets and add these maneuvers to the final trajectory.

Currently, our algorithm only calculates the drag along the straight line segments of the trajectory, ignoring transitions and corners. The final version of our algorithm will account for the drag generated by turns and use all the drag values to determine the final optimal trajectory.

5.3 Scheduling Dependent Tasks

For finding the optimal trajectory by the Wave Front Algorithm we need to execute many tasks using *Cart3D* with various flight parameters. The trajectory continuity condition implies dependencies between the flight parameters along the incoming and outgoing edges of each way point. These dependencies are represented as an acyclic directed task graph (e.g the task graph of the trajectories from the base to the targets had 393 nodes and 1402 arcs).

An algorithm called the "Allocator" was devised to schedule the tasks of the graph using subsets of the processors of a supercomputer. The algorithm uses information from the GridScape to determine the available resources and ultimately matches all the nodes (i.e. tasks) with a certain number of processors to minimize the total turn-around time. Nodes are executed in a data flow manner: a node is assigned for execution as soon the data have arrived along all incoming arcs.

We keep the information about current state of the system resources (available, busy, estimated release time) in the GridScape. The scheduler keeps track of the available and unavailable processors, along with the nodes that are ready and those that are waiting for other nodes to complete. During each cycle of the scheduler, it attempts to allocate all the available, idle processors to tasks that are ready to execute. Processors are distributed evenly between the ready nodes, and the left-over processors that cannot be evenly distributed are allocated to the nodes that have the largest number of outgoing arcs. Once all the processors have been distributed, the nodes allocated to processors be-

gin execution. When a node has finished, the processors it used become idle and are used to execute another task assigned to them by the Allocator.

The Allocator scheduler currently statically schedules the processors because it is the simplest way to not only predict the turn around time, but also improve the schedule without having to run lengthy, expensive tests using valuable supercomputer time. The run time for each task is very predictable because the tasks differ only slightly in the areas of Mach number, sideslip angle, and angle of attack, which negligibly affects turn-around time. The *Cart3D* code was tested with different numbers of processors to determine accurate simulated run times. The simulation abilities of our scheduler allow the user to see speedup capabilities of applications using different numbers of processors or configurations of the scheduling algorithm, see Figure 7.

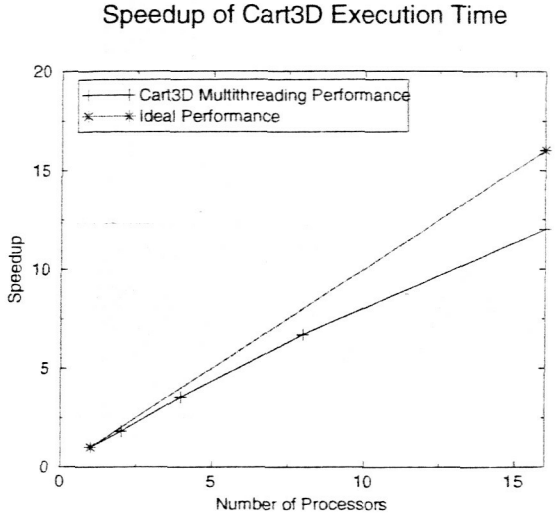

Figure 7. Scalability of Cart3D when computing a single MF configuration.

In Figure 8 the graph of the performance gains/losses from the simulation predictions can be seen. Running the tasks with no dependencies (where no trajectories depend on incoming trajectories) is the ideal case because no tasks are required to sit idle while other tasks finish. The figure shows that using our algorithm with dependencies returns only small performance losses from the ideal case for small number of processors. The tuned algorithm in the figure gives more processors to tasks that pass their results to many other tasks. The tuned algorithm performs better in most, but not all cases.

6 Experiments

The navigation system is implemented in Java. It uses the Java Registry to install task services on hosts used for experiments and the Java Remote Method Invocation (RMI) to run the benchmark tasks and to communicate data between them. In addition, it uses the Java Native Interface

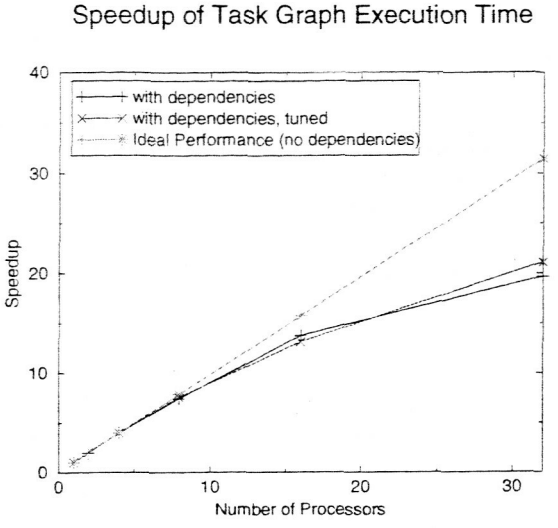

Figure 8. Speedup of the task graph execution time. The schedules were generated by two variants of the Allocator scheduler. For comparison we show the speedup which can be obtained while ignoring all intertask dependencies.

(JNI) to invoke the *Cart3D* tasks written in C or FORTRAN. We tested the navigation system on the grid – its nodes are shown in Table 1. During our experiments all grid machines had normal production loads.

To launch jobs, we implemented the `jgrun` command, which has an interface similar to `mpirun`. GridScape performance was acquired by using the Java version of ED.S of the NAS Grid Benchmarks [9]. We used automatic submission of the servers to the queue, requesting 16, 32 or 64 processors on the machines controlled by the PBS batch scheduler.

At the time of submission of the paper we were not able to finish our computations and find the optimal trajectory itself due to a significant configuration change of the NAS computer system. In a future study we are planning to have the optimal trajectory and measured (not estimated) efficiency of the schedule produced by the Allocator.

7 Conclusions and Future Work

We have described an architecture and implementation of an Application Navigation System (ANS) that automates execution of the applications comprised of many dependent tasks. The ANS system automatically acquires a map of the available compute resources and assigns tasks to them. The system chooses resources that provide the fastest advance of application tasks and, as a result, achieves 64% efficiency when solving a trajectory optimization problem.

In this paper we laid out a framework for scheduling and execution of dependent tasks. This opens many areas of research involving the trajectory optimization problem and improving scheduling of the tasks. In particular we are

Table 1. The the grid machines used in our experiments.

Machine Name	NP	Clock Rate (MHz)	Peak Perf. (GFLOPS)	Memory (GB)	Maker	Architecture	Batch System
O2K	32	250	16	8	SGI	Origin2000	-
O2K1	128	250	64	32	SGI	Origin2000	PBS
O3K1	512	400	400	262	SGI	Origin3000	PBS
O3K2	1024	600	1200	256	SGI	Origin3800	PBS
O3K3	256	400	200	98	SGI	Origin3000	PBS

planning to increase precision of our model. The first step would be to calculate general drag values for various types of turns. Ultimately the model should calculate the drag needed for the exact turn. The second step is to use a trajectory space which is adapted to the terrain and use specifics of the trajectory space to improve scheduling by looking ahead. We will improve our scheduling algorithm and reduce total amount of work by pruning trajectories with high drag to eliminate computations along suboptimal edges.

Acknowledgments. We want to thank Michael Aftosmis, Scott Murman, and Marian Nemec (all NASA Ames) for providing the Cart3D package and the Martian Flyer geometry triangulation. Also we thank Timothy Sandstrom for providing Mars elevation data.

References

[1] A. Al-Theneyan, P. Mehrotra, M. Zubair. "A Resource Brokering Infrastructure for Computational Grids." LNCS 2552, pp. 463-473, 2002.

[2] F. Berman, R. Wolski, S. Figueira, J. Schopf, S. Shao. "Application Level Scheduling on Distributed Heterogeneous Networks." In *Proceedings of Supercomputing 1996*, 1996.

[3] Cart3D. http://people.nas.nasa.gov/~aftosmis/cart3d/cart3Dhome.html

[4] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. "Heuristics for scheduling parameter sweep applications in grid environments". In *9th Heterogeneous Computing Workshop (HCW)*, pp. 349-363, 2000.

[5] H. Chen, M. Maheswaran. "Distributed Dynamic Scheduling of Composite Tasks on Grid Computing Systems". In *16th International Parallel and Distributed Processing Symposium, IPDPS 2002*, April 15-19 2002.

[6] P. Cattermole. Mars: "The Story of the Red Planet." Chapman & Hall, London, England: 1992

[7] Grid Application Development Software (GrADS) Project. http://www.hipersoft.rice.edu/grads.

[8] M.A. Frumkin, R. Hood. "Using Grid Benchmarks for Dynamic Scheduling of Grid Applications" *Proceedings of the 15th IASTED International Conference "Parallel and Distributed Computing and Systems" (PDCS'2003)*, Marina Del Rey, USA, Nov 3-5, 2003, p. 31-36.

[9] M.A. Frumkin, Rob F. Van der Wijngaart. "NAS Grid Benchmarks: A Tool for Grid Space Exploration." *Cluster Computing* 5, pp. 247-255, 2002.

[10] Y.-K. Kwok, I. Ahmad "Static Sceduling Algorithms for Allocating Directed Task Graphs to Multiprocessors." *ACM Computing Surveys* Vol. 31, issue 4 pp. 406-471, 2002.

[11] S.S. Vadyhiyar, J.J. Dongarra. "A Metascheduler for the Grid." *Proceedings of HPDC-11*, 23-26 July, 2002, Edinburgh, Scotland, pp. 343-351.

[12] R. Wolski, N.T. Spring, J. Hayes. "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing." *J. Future Generation Computing Systems*, 1999, http://nws.npaci.edu/NWS/.

[13] A. Boyle. *Proposal for Mars Exploration by Robotic Plane*, 1998 http://www.robotbooks.com/Mars-plane.htm/.

[14] M. Ravine. *Airplane Propsed for Mars Survey*, 1998 http://www.msss.com/mage_release/.