Rapid Prediction of Air Traffic for Trajectory Based Operations

Monish D. Tandale*, P. K. Menon†,
Optimal Synthesis Inc., Los Altos, CA 94022-2777

Motivated by the fact that air traffic predictions will play a central role in the next-generation air transportation system, this research examines the feasibility of developing a computational appliance for rapid prediction of aircraft trajectories. The feasibility demonstration followed a two-pronged strategy of accelerating the trajectory propagation algorithms in an air traffic simulation environment and hardware implementation of computationally intensive functions on a field-programmable gate array (FPGA) coprocessor. This research demonstrated a 3.3× acceleration through pure software modifications. Algorithm improvements produced a 47× acceleration of some of the trajectory computation modules, leading to an overall 1.42× acceleration of the simulation. Computationally intensive portions of the trajectory propagation function were implemented on the FPGA producing between 2.3× and 5.6× acceleration of these functions for the prediction of 4096 aircraft trajectories. Acceleration for 10,000 aircraft is projected to be in the range of 3× to 8×. An assessment of the payoffs in employing cluster computing architectures is also explored.

I. Introduction

An important element of the Next Generation Air Transportation System (NextGen) concept being developed by NASA in partnership with the Joint Planning and Development Office (JPDO) is the Trajectory-Based Operations (TBO). This new concept is expected to dramatically change the manner in which traffic is managed in the national airspace, leading to increases in airspace capacity and efficiency. The current Air Traffic Management (ATM) methodology is based on a fixed airspace structure tied to geographic locations within the NAS, and can be termed as Fixed Airspace Operations. The Trajectory-Based Operations is a paradigm shift from the current approach and employs four-dimensional (4-D) trajectories as the basis for managing the ATM system. In Trajectory-based operations (TBO), all ATM decisions across all time horizons, are fundamentally related to 4-D trajectories. In this operational concept, 4-D trajectories are the principal language for information exchange, planning, and analysis, enabling greater use of digital communication and ground-based and airborne automation, and facilitating coordination and collaboration between aircraft operators and air traffic management entities.

Since aircraft 4-D trajectories will play a central role in the NextGen, it is important to be able to rapidly generate trajectory predictions for implementing the ATM functionality, either on the ground or onboard aircraft. While high-end computers can be used for this purpose, emerging capabilities of the field programmable gate arrays (FPGAs) and multi-core processors can be exploited to realize substantial acceleration of trajectory computations at a modest cost increment. Based on these premises, the present work pursued the development of a Computational Appliance for Rapid Prediction of Aircraft Trajectories (CARPAT) that combines the trajectory propagation features of the NASA-FACET software with the emerging computational power of the field programmable gate arrays. Additionally, a recently-developed client-server technology named CARAT# that allows access to the FACET functionality over the Internet Protocol (IP) is employed as the input-output of the present system. This allows host computer to access the computational appliance in a transparent manner over a broadband local area network connection.

A conceptual diagram illustrating the operation of the computational appliance is given in Figure 1. It is configured around one or more blade servers running accelerated versions of the FACET software, with multiple FPGA boards providing the acceleration for repeated computations. Air traffic management applications running on client machines can send-out the flight plans, aircraft type, and the current states to the computational appliance.

* Research Scientist, 94 First Street, Suite 240, Member AIAA.
† Chief Scientist, 94 First Street, Suite 240, Associate Fellow AIAA.
Trajectory predictions will be generated by the computational appliance and sent out to the clients over a broadband network connection. The computational appliance will be configured to periodically download weather data from sources such as ITWS (Integrated Terminal Weather System) in an automatic manner to ensure that the trajectory predictions remain realistic.

The objective of the present research is to demonstrate the feasibility of developing the computational appliance and to investigate potential speed advantages relative to the implementation on a conventional, general-purpose computer. This work follows a two pronged strategy of software acceleration and hardware implementation of computationally intensive functions on the FPGA coprocessor. The hardware setup is assembled by integrating a Windows®-XP® Host PC with a commercial, off-the-shelf FPGA coprocessor.

The following tasks are performed during the course of this research:

1. Disabling FACET Functions Peripheral to Trajectory Prediction
2. Profiling the FACET C code and modifying it for speed improvements.
3. Implementing CARAT# components to serve as the I/O of the computational appliance.
4. Integrating Windows-XP Host PC with the FPGA Coprocessor.
5. Implementing some of the computationally intensive FACET software components on the FPGA coprocessor.
6. Comparing the performance of the accelerated system to the baseline FACET.

Section II describes the process of code profiling and software acceleration. Section III describes the role of CARAT# as the input-output interface of the computational appliance. Section IV describes the hardware set-up of the computational appliance and the implementation of a computationally intensive function on the FPGA coprocessor. A comparison of the accelerated trajectory propagation to baseline FACET trajectory propagation is provided in Section V. Analysis of additional acceleration that can be expected using the emerging cluster computing technologies is given in Section VI followed by conclusions in Section VII.

II. Code Profiling and Software Tuning

The computational appliance being developed in the present research is primarily focused on accelerating trajectory prediction functions in FACET. Hence as a first step in the process, FACET functions which do not contribute to trajectory propagation are disabled, the graphical user interface being an example.
After disabling the functions peripheral to trajectory propagation, the next step is to accelerate the trajectory propagation functions that consume a significant portion of the run-time. Such functions can be identified by code profiling. Code segments that consume highest run-times are then accelerated.

Figure 2 illustrates the performance tuning cycle for tuning the FACET software. The first step involves running of the FACET software for a defined traffic scenario to measure its run-time performance. This is followed by identification of the hot-spots corresponding to the portions of the code which consume significant amount of run-time. AMD CodeAnalyst™ aids in the first two steps. The third step involves the analysis of the source code identified as a hot-spot, to investigate potential improvements that can reduce the run-time. The fourth step involves modification of the program to implement improved code for accelerated performance. This is further followed again by step 1 to measure the performance gains resulting from the code modification.

III. CARAT# Software Components Providing Interface for the Computational Appliance

FACET is designed to operate as a GUI driven software, with the user accessing the FACET functionality though drop-down menus and the frames. Since the GUI is disabled during the process of software optimization, an alternative method must be provided to access the FACET functionality. This access is provided through the recently developed CARAT# software.

CARAT# ("Carat-Sharp", Configurable Airspace Research and Analysis Tool – Scriptable) allows the user to access the FACET functionality through Java programs, or the well-known MATLAB® environment and Jython®, the Java version of the popular Python® environment. CARAT# incorporates a built-in Server-Client architecture, permitting remote and collaborative scripting over the internet protocol.

CARAT# allows the user to access the FACET functionality through a set of Java methods. These Java methods can be accessed through a range of software environments, including interactive, scriptable environments such as MATLAB® and Jython®. Figure 3 illustrates the overall architecture of the CARAT# software.
IV. Implementing Software Components on the FPGA Coprocessor

This section describes the process of hardware acceleration by implementing computationally intensive functions that consume significant portion of the run-time, on the FPGA coprocessor. There is an increasing trend toward using Field Programmable Gate Arrays (FPGAs) as hardware accelerators for high-performance computing. Hardware accelerators take advantage of the parallel processing structure of the FPGA to calculate more computations per clock cycle than general-purpose CPUs, and can sometimes deliver orders-of-magnitude increase in performance [9]. For example, many FPGAs have large numbers of hard-wired multipliers or multiply-accumulate units that can be connected into parallel data paths to make short work of math-intensive routines [10].

An FPGA is a semiconductor device containing programmable logic components and programmable interconnects. The programmable logic components can be programmed to duplicate the functionality of basic logic gates such as AND, OR, XOR, NOT or more complex combinational functions such as decoders or simple math functions [11]. Since FPGAs are programmable devices, the behavior of the FPGA is defined by the user through a Hardware Description Language (HDL). In an attempt to reduce the complexity of designing in HDLs, which have been compared to the equivalent of assembly languages, there are several initiatives to raise the abstraction level of the design. Of late, C/C++ to Hardware Acceleration Compilers (C2H) are become available in the market.

The central idea here is to convert speed critical functions in the FACET ‘C’ code into an HDL by using a C2H compiler and run the code in real-time on an FPGA-based programmable hardware platform to accelerate the trajectory propagation.

Figure 4 illustrates the architecture of the computational appliance developed during this research effort. The CARPAT hardware comprises of a PC host processor and an FPGA coprocessor connected by a PCI-X interface. The host computer executes the trajectory propagation functions within FACET. Computationally intensive functions in trajectory propagation are executed on the FPGA coprocessor. The CARAT# software serves as the I/O for the computational appliance.

After the assembly of the CARAT hardware described above, the next step is to implement computationally intensive FACET functions on the FPGA. FACET functions coded in C are transformed into a bit file to configure the FPGA with the desired functionality using a C to Hardware Description Language Compiler.
V. Performance Comparison of the Accelerated Trajectory Prediction to Baseline \textit{FACET}

The software optimization and hardware acceleration described in the previous sections resulted in significant acceleration in the trajectory computations over the baseline as follows:

A. Acceleration in \textit{FACET} due to Software Tuning

This research demonstrated a 3.3× acceleration of \textit{FACET} through pure software modifications such as disabling the graphical user interface and eliminating disk operations during run time. Algorithm improvements produced a 47× acceleration of some of the trajectory computation modules, leading to an overall 1.42× acceleration in \textit{FACET}.

B. Hardware Acceleration of a computationally intensive trajectory propagation function in \textit{FACET}

The computationally intensive trajectory propagation function under consideration, is evaluated in a \texttt{for} loop which iterates over all the aircraft in the \textit{FACET} simulation.

```c
for (i=0; i<AIRCRAFT; i++){
    function;
}
```

Figure 5 illustrates the implementation of the \texttt{for} loop on the \textit{PC} and \textit{FPGA}. The host processor in the \textit{PC} (2.8 GHz) is significantly faster than the \textit{FPGA} operating at 200 MHz. Hence execution of a single function is significantly faster on the host \textit{PC} than the \textit{FPGA}. Exploiting fine grain parallelism within the function reduces the time taken for the execution of a single call to that function on the \textit{FPGA}.

The host \textit{PC} outperforms the \textit{FPGA} for the execution of a single iteration of the \texttt{for} loop as indicated in Figure 5. However the balance tilts significantly in favor of the \textit{FPGA} with increase in the number of iterations due to pipelining\footnote{14}. For the host \textit{PC}, the time taken to execute \(n\) iterations is:

\[
\text{Execution time of } n \text{ iterations } = n \times \text{execution time of one iteration the host PC} \quad (1)
\]

This behavior can be observed in the execution times recorded for the \textit{PC} implementation of the \texttt{for} loop shown by the blue line in Figure 6. The execution time for the host \textit{PC} implementation increases linearly with the number of aircraft, as indicated by the blue straight line. The broken line beyond 4096 aircraft denotes extrapolated data.

However in the case of the \textit{FPGA}, the iterations in the \texttt{for} loop overlap due to pipelining. In this case, the time taken to complete \(n\) iterations is:
The non-overlap time for the FPGA is considerably lower than the execution time for individual iterations on the host-PC. An additional factor to be considered in the present FPGA implementation is that of the time required for transferring data from the host PC.

The execution times on the FPGA with and without data transfer time are also given Figure 6. Note that the FPGA implementations have significantly smaller slopes than the trend for the host PC implementation. Thus, a break-even point exists where the host PC and the FPGA show similar execution speeds. Beyond this break-even point the FPGA shows faster performance than the PC with increase in the number of iterations. As in the case of the host PC execution time, the broken lines denote extrapolated data.

Beyond the break-even point the FPGA shows faster performance than the PC with increase in the number of iterations. In the current implementation, the execution of the computationally intensive function on the FPGA takes place as a sub-function of the main program running on the host processor. This involves transfer of the function arguments from the host to the FPGA and the transfer of function return value from the FPGA back to the host. The difference between the FPGA lines with and without data transfer in Figure 6 gives the amount of time taken to transfer data between the FPGA and the host. The time for data transfer is approximately equal to the time taken for function execution on the FPGA. Minimizing data transfer time between the host and the FPGA by using shared memory and pointers, using high-speed data interfaces between the host PC and the FPGA will give FPGA execution times which lie between the two FPGA lines in Figure 6.

Due to the memory limitations on the FPGA board used for this research, the maximum number of aircraft that can be simulated on the FPGA is 4096 \((2^{12})\). Hence experimental data was only recorded for up to 4096 aircraft. However, if an FPGA with higher number of logic cells is used, more aircraft trajectory computations can be carried out in parallel. The experimental results are extrapolated linearly to predict the possible execution times for 4096 to 10,000 aircraft. Figure 7 shows the acceleration factor achieved by the FPGA implementation (with and without data transfer time) over that of a PC implementation for the function. This graph is obtained by dividing the execution times for the PC and the FPGA in Figure 6.
Figure 6. Comparison of Execution Times for a Computationally Intensive FACET Function on the Host PC and FPGA

Figure 7. Acceleration Factor Achieved over a PC Implementation

VI. Analysis of Potential Acceleration due to Cluster Computing

The FPGA parallelizes the computation on a fine grain level. Coarse-grain parallelization\(^ {15} \) is possible by splitting the computation over a cluster of conventional processors. An analysis was undertaken to assess the potential acceleration obtainable using distributed or cluster computing. As conceived at present, the trajectory computations for multiple aircraft will be equally divided between the computers in the cluster, and the results will be collected and reported by a host.
In order to simulate cluster computing, a Java program was written to create data input files with multiple copies of the same flight with varying flight identification numbers. FACET simulations were run by varying the number of copies of the flight in the input data file. The recorded execution times are reported in Figure 8.

![Figure 8. Variation in the Simulation Run-Time with the Number of Aircraft in the Simulation](image)

This figure shows that the computational requirements increase linearly (approximately) with the number of aircraft trajectories being propagated. This suggests that the acceleration that can be expected by running this simulation on a distributed computing cluster is proportional to the number of CPUs used.

It is important to note that the foregoing estimate did not consider the overhead involved in task distribution by the host, and output data aggregation. These are expected to be nearly constant with respect to the number of computers in the cluster.

### VII. Conclusions

The research presented in this paper demonstrated the feasibility of developing the computational appliance by integrating a single FPGA coprocessor with a single Host PC. Speed improvements due to software tuning and hardware implementation of computational intensive functions were demonstrated. This research demonstrated a 3.3× acceleration of FACET through pure software modifications such as disabling the graphical user interface and eliminating disk operations during run time. Algorithm improvements produced a 47× acceleration of some of the trajectory computation modules, leading to an overall 1.42× acceleration in FACET. A computationally intensive portion of the FACET trajectory propagation function was implemented on the FPGA producing between 2.3× and 5.6× acceleration of these functions for 4096 aircraft trajectories. Acceleration for 10,000 aircraft is projected to be in the range of 3× to 8×. The potential acceleration using Cluster Computing architecture was analyzed. Expected acceleration is proportional to the number of CPUs used. Hence a compute cluster made up of conventional processors with each processor off-loading critical tasks to one or many FPGA co-processors is a candidate configuration for future research.

Thus this paper showed that the FACET trajectory predictions can be significantly accelerated using software modifications and high-performance computing hardware. Additional opportunities for FACET software acceleration have been identified during the course of the work and must be pursued in the future research. The concepts demonstrated in Phase I can be extended to develop CARPAT as a Cluster-Based FPGA High Performance Computing System Solution. Other concepts that can be explored for software acceleration are the use of a Real-Time Operating System, implementing FACET calculations on the FPGA using fixed-point numbers and multi-threaded FACET execution. Computational appliance for rapid prediction of aircraft trajectories can then be evaluated for use with compute-intensive air traffic management algorithms such as iterative optimization and stochastic simulation.
Acknowledgments

This research was supported under NASA Contract No. NNX07CA13P, with Dr. Karl D. Bilimoria serving as the Technical Monitor. The authors would like to thank Dr. Bilimoria for his valuable inputs and encouragement throughout the course of this work. We thank Dr. Shon Grabbe and Dr. Kapil Sheth of NASA Ames Research Center for their interest in this project.

References