International Conference on Field-Programmable Technology, Dec. 15-17, 2003, Tokyo, Japan.

Multisensor Inversion with High-Performance FPGA Computation

Yongxiang Hu[1], Yang Cai[2], Mark Tomzak[2], Tsengdar Lee[3]
1. NASA Langley Research Center. 2. Carnegie Mellon University. 3. NASA Code Y.
Email: yongxiang.hu-1@nasa.gov

Abstract

An FPGA-based reconfigurable generalized inversion processor is developed for real-time physical property retrievals of the atmosphere, land and ocean. The processor is based on Generalized Non-Linear Regression algorithm and trained with radiative transfer simulations and observations for autonomous detection of satellite measurement signatures and retrievals of atmospheric physical properties. The FPGA approach is faster than traditional computing methods in orders of magnitude. The generalized inversion processor is able to automatically adapt to multi-platform sensors, eliminate redundant algorithm development and provide a vehicle for autonomous onboard image analysis and physics-based data compressions.

1. Introduction

Sensory data fusion and analysis from multiple satellite platforms have been increasingly important tasks in Atmospheric science research. Unfortunately, seamless multi-platform data analysis currently does not exist because of difficulties in algorithm development and huge cost. For example, developing an ensemble of operational algorithms for a single instrument, such as CERES and MODIS, took about a decade and cost millions. As a result, these Earth observing missions are not fully achieving their capabilities and could find more applications. With the growing number of satellites and improving spatial and spectral resolutions, users have been coping with massive data on daily basis.

This study is to build a near real-time tool for solving nonlinear inverse problems. It intends to develop a generalized inversion processor and significantly reduce the cost of physical inversion for combined observations. With this tool, scientists do not have to spend too much time on repeatedly learning and developing complicated mathematical and computational tricks, such as the Rodgers iterative methods [7]. GNR requires lots of computations, especially for atmospheric remote sensing problems with many dimensions of observations. Thus, the centerpiece of this study is to implement GNR on custom FPGA chips and maximize the computational performance. Authors have been investigating the utilization of FPGA based computing in the processing of remote sensing scientific algorithms.

2. Physical-Inversion Models

Given physical properties [**P**] of the atmosphere, most multi-spectral observations [**O**] can be simulated [**O** = f(**P**)], thanks to known physics models (also called "forward models"). Physical-inversion is the process of retrieving physical properties from observations [**P** = f^{-1}(**O**)]. Physical inversion is a key component of an Earth observing mission. For example, near sea surface wind-speed (W) can be retrieved from the TRMM Microwave Imager (TMI) brightness temperatures (**B**). For given wind speed, brightness temperatures can be estimated from physics models [**B** = f(W)]. The simplest inversion to retrieve wind speed W from observation **B** [W = f^{-1}(**B**)] is a linear regression

$$W = C_1 B_{1,h} + C_2 B_{1,v} + C_3 B_{2,h} + \ldots$$

For given physical properties, we will generate a huge library of high spectral resolution radiative transfer calculations for UV, visible, near IR, thermal IR and microwave wavelengths. To create the database, we will use radiative transfer models such as MODTRAN for gas absorption and DISORT for multiple scattering.

Most remote sensing related inverse problems are *nonlinear* in nature. A generalized nonlinear regression (GNR) method is used specifically for high-performance computing implementation. Using GNR, the wind speed W can be derived from microwave measurements **B** as:

$$W(\vec{B}) = \sum_{i=1}^{N} \hat{W}_i D_i \Big/ \sum_{i=1}^{N} D_i$$

$$D_i = \exp[-\sum_{j=1}^{M} \frac{(\hat{B}_{j,i} - B_j)^2}{(r_i s_i)^2}]$$

where $\hat{W}_i$ and $\hat{B}_{ji}$ are wind speed and microwave brightness temperatures from forward model simulations and previous retrieval results. ? is a correlation factor

between the brightness temperature of channel j ($\hat{B}_j$) and the wind speed, and ? is measurement error of $\hat{B}_j$. Based on radial-bases neural networks, GNR is a non-parametric estimation method. Retrievals using GNR are as straightforward as linear regressions but yield more accurate results. GNR has been tested on other remote sensing applications as well [9]. With all the existing physics models, we can produce equivalent $\hat{W}_i$ and $\hat{B}_{ji}$ for all instruments from forward simulation. Comparing with other inverse methods, GNR is more universal since it does not require *a priori* information. However, GNR is not necessary an ideal candidate for high performance parallel computations. To improve the parallelism of the algorithm, we modify the GNR to its subset, the Radial Basis Function (RBF) model, which is simpler and easy to parallelize.

$$W(\vec{B}) = W_0 + \sum_{i=1}^{k} W_i \exp[\frac{(\hat{B}_i - \vec{B})^2}{2s_u^{\ 2}}]$$

Where each $\hat{B}_i$ is a kernel center and where $dist()$ is a Euclidean distance calculation. The kernel function D_l is defined so that it decreases as the distance between B_u and $B_{j,i}$ increases. Here k is a user-defined constant that specifies the number of kernel functions to be included. The Gaussian weight function is centered at the point $\hat{B}_i$ with some variance s_u^2. The function provides a global approximation to the target function, represented by a linear combination of many local kernel functions. The value for any given kernel function is non-negligible only when the input $\vec{B}$ falls into the region defined by its particular center and width. Thus, the network can be viewed as a smooth linear combination of many local approximations to the target function. The key advantage of RBF networks is that they contain only summation of kernel functions rather than compounded calculation so that RBF networks are easier to be parallelized. In addition, they can be trained much more efficiently with genetic algorithms.

3. System Architecture

We used National Instrument's FPGA module for rapid prototyping. We used a PC processor as a host that transfers data into and out of the FPGA board. The two interface with a PXI bus. Current reconfigurable FPGA boards function like coprocessor cards, which are plugged into desktop or large computer systems, called the host. By attaching a reconfigurable coprocessor to a host computer, the computation intensive tasks can be migrated to the coprocessor forming a more powerful system.

A prototype of the physical inversion model solver was constructed on the NI PXI-7831R FPGA prototyping board. The FPGA Vertex II 1000 contains 11,520 logic cells, 720 Kbits Block RAM, and 40 embedded 18x18 multipliers. This board is primarily designed for implementation of custom electronic logic probes. It can be programmed using the tools provided by National Instruments. These tools convert LabVIEW VI block-diagram code into VHDL, which is then compiled to a layout for the FPGA using the standard Xilinx toolset. With this system, rapid prototyping of a solution and alteration of the number of parallel channels can be done without significant development time.

In our design, the computation intensive portion of the multi-spectral image classification algorithm resides on the calculations within the processor. The user interface, data storage and IO, and adaptive coprocessor initialization and operation are performed on the host computer. The inversion algorithm is mapped to the FPGA board. The FPGA has a direct connection to the host through the PCI bus. Eighteen FPGA boards or other instruments can be plugged into the chassis. This architecture allows users to integrate sensors and physical inversion systems in one place. It is possible to bring the system to an airborne craft for onboard experiments.

4. Implementation

We have overcome the limitations of the current development tool Labview™ and developed new functions for solving our problems, e.g. exponential functions and memory control functions as well as floating point functions. We have implemented a parallel neural network Generalized Regression Model on FPGA with historical satellite data about wind speed and surface temperature. To increase the capacity and speed, we have also implemented Radial Basis Function, a subset of Generalized Non-linear Regression model. The RBF model increased the capacity in two folders and up. With fixed point computing, the RBF presented ideal parallelism behavior on the FPGA. In addition, we have prototyped a Genetic Algorithm (GA) for training the neural networks on FPGA. The followings are the implementation details:

4.1 Floating Point Representation

We have implemented both fixed-point and floating-point representations on the FPGA. Due to the limited number of gates available on the FPGA chip, it is desirable to use fixed-point arithmetic in our implementation of the inversion algorithm. However, the sensitivity to small values that was demonstrated during testing of the GNR algorithm necessitated the development of a floating-point solution. The RBF algorithm did not demonstrate such sensitivity and therefore allowed fixed-point representation to be used in all of its aspects. In our floating-point representation, we transform the algorithm into fixed point prior to hardware

implementation. The width of the fixed-point data path is determined by simulating variable bit operations in Labview™ and by comparing the results obtained from the original algorithm in floating point.

4.2 Exponential-Function

Exponential function is the most computing-intensive component of this application. We use a Look-Up Table (LUT) for computing the exponential function. A LUT is used to determine the value of e^{-a}.

4.3 Saturated 16-Bit Multipliers

Due to limited number of the embedded multipliers on the FPGA chip, we used 16-bit saturated multipliers instead of 32-bit full-length multipliers. We trade the accuracy with computing resources and time.

4.4 Memory Control

A memory initialization utility has been developed to arbitrate memory address so that the memory blocks can be accessed simultaneously.

4.5 Training Algorithm Implementation

We used Genetic Algorithm (GA) to train the weights in the inversion models. GA is a general purposes optimization algorithm that is inspired by genetics. It normally generates better convergence solutions for neural networks. To make an efficient implementation of the algorithm on FPGA, we focused on gate-based approach rather than matrix-based approach. The gate-based approach takes advantages of FPGA architecture and yields a simpler and more efficient solution for GA implementation. For example, we used bit shifting and XOR to implement the random number generator that used minimal FPGA resources. Also, we used on chip memory to store the chromosome strings, rather than the 'expensive' arrays.

4.6 Parallelism Algorithm Implementation

We have implemented the parallelism algorithms at two levels: First, on-chip parallelism, where we use Labview™ "Parallel While Loop" to execute the parallel modules. Second, we put multiple FPGA boards on the PXI bus where they can run in parallel.

4.7 Host Software

We use Labview™ to configure the FPGAs and transfer the data in and out from FPGAs. A Graphical User Interface is developed to accommodate user inputs. In addition, the TCP/IP enabled utility allows user to access the FPGA server remotely.

4.8 Inversion Algorithm Implementation

The example prototype designed used two-dimensional input vectors (M=2 in the GNR equation). One hundred of these inputs were chosen from a set of 5000 input/output pairings and used as the set of known values. The remaining inputs were used as test data to generate results, which were then compared against the expected results paired with the inputs. All values of r_i and s_i were set to produce a $(r_i s_i)^2$ value of 0.001. Input values and output results were normalized between 0 and 1; all inputs and outputs are represented using fixed-point notation. Note that fixed-point values will be denoted as x.y, where x is the number of bits representing the integer value of the number and y the number of bits of precision in the number. For example, a 4.1 representation would allow for the numbers 0 through 15.5 to be represented at a resolution of 0.5. The GNR solver algorithm first notes the tick count on the FPGA timer, then acquires the input values from the host computer and stores them in memory. The solver then iterates 100 times to update both the D_i value and the value of $\hat{W}_i D_i$. Finally, these values are summed and divided as described by the algorithm, and the result sent to the host computer. Another tick count is taken when the calculation is complete, and the difference between the two tick counts is sent to the host so that the processing time could be calculated. This sub-VI takes two inputs: an index i to determine which known value is being compared at this iteration and an input index that denotes which input is being compared. We first tried multiple inputs, but later on, we realized that due to storage and input-output limitations on the PXI-7831R board, it was more efficient to handle one input at a time. The GNR system sets this input to 0.

The upper and lower boxed regions are identical sections that compute $(\hat{B}_{j,i} - B_j)^2$. These results are then summed, and the lower twelve bits from 2^{-6} to 2^{-17} are used as a reference into a lookup table containing solutions to the equation $e^{\frac{-x}{.001}}$. If any of the bits besides the reference bits are set, a solution of 0 is returned; this is justified because any solution to $e^{\frac{-x}{.001}}$ for $x \geq 2^{-5}$ is less than 2^{-15}, and therefore cannot be represented at the resolution of 1.15 used as the final output for D_i. Once the lookup table generates a value for D_i, another lookup table is used to retrieve the value of $\hat{W}_i$. These values are multiplied together, and the SubVI generates two outputs: D_i and $\hat{W}_i D_i$.

The lookup tables used to store the known values and calculate the value of D_i are constructed using the FPGA Memory Extension Utility in the LabVIEW system, which allows for RAM to be initialized during the FPGA design process. Fig. 5 shows the memory initialization code for the D_i lookup table and the set of known values, respectively.

To convert the fixed-point results to floating-point and to verify the results of the GNR calculation, a host VI was constructed. This is a program that operates in the

Windows environment and interacts with the FPGA, sending it input values and retrieving the resulting outputs. It consists of an input converter that translates the floating-point inputs into 4.12 fixed-point notation, an input/output block to communicate with the FPGA, an output converter, and a verifier system that calculates the expected GNR solution in floating-point notation independently of the FPGA. These values can be compared to determine the accuracy of the FPGA results.

Once the system was constructed, the number of parallel processing channels was changed and the processing time relative to the changes was observed. The calculation of D_i was tested both with the parallel version of the algorithm as described (2 channels) and another version, which calculated the values of $(\hat{B}_{j,i} - B_j)^2$ using a loop through j=0 and j=1. Also, the 100-iteration loop in the main body of the FPGA design was divided into multiple loops of less iterations: a pair of loops (0-50 and 51-99), four loops (0-24, 25-49, 50-74, 75-99), and five loops (0-19, 20-39, 40-59, 60-79, 80-99). Each of these configurations was compiled and tested, and no variation in output values was detected between them.

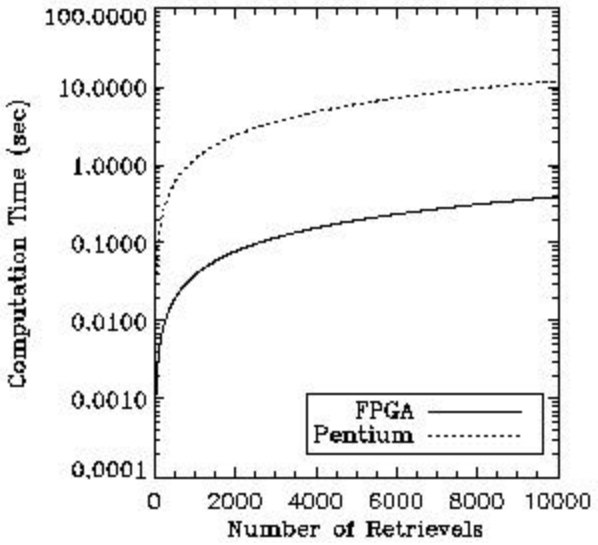

Figure 1: Computation speed: Comparisons between FPGA and Pentium.

5. Results

We have the following preliminary results through our benchmark experiments:

We found that FPGA over-performance Pentium at least two to three orders of magnitude in terms of speed. For example, for RG model, the FPGA uses 39 ns (with 10 MHz clock speed). Pentium uses between 1000 ns and 2000 ns (with 1 GHz clock speed) (See Figure 1).

We also found that the fixed point Radial Basis Function algorithm demonstrated ideal parallelism as the number of simultaneous basis compares increases.

One limitation in the design of the GNR equation was the number of multiplier units available on the Xilinx FPGA. On the Xilinx FPGA, multiplication is executed by the use of a pre-constructed 18x18 MULT unit on the chip; This unit is useful because multiplication is an expensive operation in terms of number of gates used. However, with only forty of these units on the chip the amount of multiplication allowed in the GNR calculation was extremely limited. Potential applications of this generalized FPGA-based neural network processor include: multi-platform combined retrievals; universal application of atmospheric remote sensing; real-time data analysis and atmospheric corrections for ocean and land image processing; improvements in data processing capabilities of Atmospheric Science Data Centers (ASDC). An example of accelerating atmospheric retrievals in ASDC: currently ASDC host computer has all the input “neurons”. The I/O of generalized inversion FPGA processor would be linked to ASDC using bi-directional “shared memory”.

5. Conclusions

First, we found that FPGA over-performance Pentium at least two to three orders of magnitude in terms of speed. For example, for RG model, the FPGA uses 39 ns (with 10 MHz clock speed). Pentium uses between 1000 ns and 2000 ns (with 1 GHz clock speed)

Second, we found that the fixed point Radial Basis Function algorithm demonstrated ideal parallelism as the number of simultaneous basis increases.

Third, we found the bottlenecks of the FPGA-based computing is the data I/O. How to get data in and out of the FPGA is on a critical path in terms of speed. Currently we use PXI bus that contains only 16-bit parallel channels. It can be saturated when the data exchanges are frequent. For high-speed data exchange, we plan to build our own board with a matrix of inter-connected FPGAs and use the available data I/O lines on FPGA chips for inter-chip data communication.