Improving the Performance of Parallel Backprojection on a Reconfigurable Supercomputer

Ben Cordes, Miriam Leeser, Eric Miller
Department of Electrical and Computer Engineering
Northeastern University, Boston MA
{bcordes,mel,elmiller}@ece.neu.edu

Richard Linderman
Air Force Research Labs, Rome NY
richard.linderman@rl.af.mil

Introduction

Backprojection is an image reconstruction algorithm that is used in a number of applications, including synthetic aperture radar (SAR). For radar processing applications, backprojection provides a two-step method for reconstructing an image from the radar data that are collected. First, the radar traces are filtered according to a linear time-invariant system. These filtered traces make up the input "projection" data to our system. The backprojection process then "smears" these traces across the image plane along contours that are defined by the nature of the data acquisition process (i.e. the flight path of the plane carrying the radar). Coherently summing each of the projected images provides a final reconstructed version of the scene. This is a highly effective method of processing SAR images, but is generally regarded as being computationally complex compared to traditional Fourier-based image formation techniques, and thus prohibitively difficult to implement on traditional computers. However, the smearing process contains a high degree of parallelism, which makes it suitable for implementation on reconfigurable devices.

We have previously reported our work on implementing backprojection for SAR processing on the Heterogeneous High-Performance Cluster (HHPC) reconfigurable supercomputer, located at the AFRL facility in Rome, NY. Our implementation took advantage of multiple nodes of the cluster, with each node consisting of a pair of Xeon microprocessors as well as a COTS FPGA board. We reported [1] a performance gain of 26x over a single-node, software-only implementation. Those results indicated several directions in which further performance gains could be achieved. The major obstacles to improved performance were reading data from the file system and improving the utilization of the PCI bus as well as exploiting all of the reconfigurable hardware resources available. We expect significant performance gains by eliminating these bottlenecks. In this work we report on tuning the application to the cluster on which it is running, and provide some estimates[1] for the performance gain that we expect to achieve.

Architectural Overview

Figure 1 shows a block diagram of the FPGA design that is programmed onto each FPGA on the coprocessor board. The FPGA is divided into two clock domains. The left-hand side of the picture, known as the "L-clock" domain, is synchronous to the PCI bus that connects the FPGA board to the host PC. The right-hand side, or "P-clock" domain, contains the logic that stores and processes the backprojection data.

The system operates in the following sequence. First, projection data are sent from the host PC to the FPGA. Once the on-chip projection BlockRAMs have been loaded, the FPGA is directed to run one iteration of processing. A single iteration consists of reading pixel data from the on-board SRAMs, calculating the new values based on the projection data, and writing the new pixel data back to the SRAMs. After the iteration is complete, the next set of projection data is sent to the FPGA, at which point the FPGA is ready for another iteration. Breaking the computation into multiple iterations in this fashion allows us to process larger amounts of input data than will fit at one time in on-chip memory. When all input projection data have been processed, the final target image is read from the FPGA and saved in the host PC's memory.

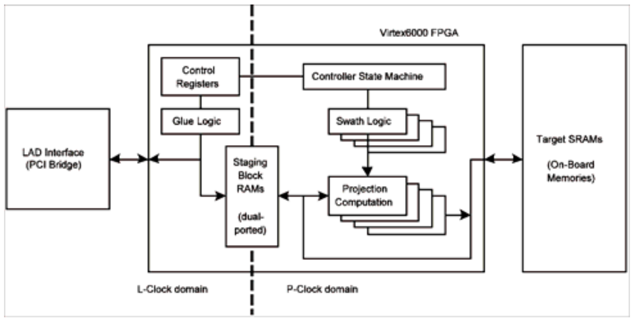

Figure 1. Block Diagram of FPGA Design

Before each iteration, projection data are sent to the board through a two-step process. In the first step, data are transferred across the PCI bus from the host PC to the FPGA, where they are stored in staging BlockRAMs in the L-clock domain. In the second step, data from the staging BlockRAMs are read in the P-clock domain and written into projection data BlockRAMs, where they will be used during processing. In order to improve the throughput, a double-buffering mechanism is employed. By using two staging BlockRAMs instead of one, data can be written into one while it is being read from the other, thus overlapping the two steps.

Pixel data that eventually form the final target image are stored in on-board SRAMs. These SRAMs are zeroed before processing begins. Pixel values of the reconstructed target image are accumulated during

[1] Here we present estimates; by the time of the conference, experimental results will be available for presentation.

processing. In order to maximize the available memory bandwidth, two copies of the pixel data are kept. In each iteration, pixel data are read from one SRAM, new values are accumulated, and the results are written to the other. For the next iteration, the roles of the two SRAMs are switched. Extra logic keeps track of which SRAM is the most up-to-date copy of the target image so that the correct final result is read.

The final target image is copied to the host PC using the staging BlockRAMs. Blocks of image data are first transferred from the SRAM to the staging BlockRAMs, and from there transferred to the host PC over the PCI bus. The same double-buffering mechanism is used to improve the throughput of this transfer.

Architectural Improvements

The first performance bottleneck to be removed is the use of the clustered file system. Performance studies showed that reading in the backprojection data and writing the final consolidated target image to disk consumed more than 50% of the total runtime of the system. The new system will be adapted to eliminate the use of the file system, so incoming data will be streamed in over the Gigabit Ethernet interface. Output data will use a publish/subscribe [2] mechanism, where each published block is the size of a target image produced by a single node. This fine level of granularity is advantageous, since it reduces the amount of uninteresting data transmitted to a subscriber. Removing the effects of file I/O from both the reference (software-only) model and the previous (hybrid) system reveals that the hybrid system runs 270x faster than the reference model. We will base the rest of our estimates on this number.

Second, we wish to increase the utilization of the PCI bus for transferring data between the host PC and the FPGA board. The current system implements a double-buffering mechanism to improve data transfer performance, but studies show that the full bandwidth of the PCI bus is not utilized. Two architectural improvements are available which will increase the utilization factor. First, the current system uses only one of the two available FPGAs on the coprocessing board. Programming both FPGAs will allow us to use a four-way buffering scheme instead of the previous two-way buffering, which will as much as double the activity on the PCI bus. Second, when returning target images from the FPGA to the host PC, we will use the DMA transfer mode, which will increase the performance of data transfer by as much as 10x by both reducing overhead and increasing utilization. Currently, the less efficient Programmed I/O (PIO) transfer mode is used, since the target image transfer times were not on the critical path for our design.

Finally, we can increase the processing power of the new system by taking advantage of resources that were unused in the current design. We have already described the use of two FPGAs instead of one per board; this will allow each board to process two sub-images. We will further double the size of each sub-image by increasing the utilization of the SRAMs attached to each node. The current system fills one SRAM with image data; the new system will fill two. Overall, we will increase the size of the target image processed by a single node by 4x. Using two SRAMs provides an additional benefit of increasing the amount of data that can be read in parallel, allowing the addition of a second processing pipeline to the hardware implementation. Despite increasing the size of the target image, all additional processing occurs in parallel so it requires no additional time to process.

We will also increase the number of projections that are processed in parallel. Increasing the number of projections processed in parallel from four to eight does not significantly increase the latency of a single processing iteration, and overall requires half as many iterations to process the image. Thus the processing time will be reduced by approximately 2x. We expect the improvements in the data transfer performance will make this factor significant, where in the current design the performance of the processing step is overshadowed by other factors.

Conclusions

We have presented the implementation of backprojection for SAR on a reconfigurable supercomputer. Our results indicate that significant speedup of the algorithm can be achieved by exploiting the strengths and avoiding the bottlenecks of the architecture. Our approach demonstrates significant gains in performance over an earlier implementation while at the same time increasing the size of the image processed on each node by a factor of four. The architectural improvements described here should more than offset the 4x increase in data to be transferred and processed. We expect to gain performance by removing the reliance on the file system, improving the utilization of the PCI bus during data transfer, and adding additional computation resources. Estimates show that, if we assume that a software implementation will take 4x longer to process an image that is 4x larger, our new hardware system will process data three orders of magnitude faster than the software solution, and approximately 40 times faster than our previous solution. Despite the fact that the numbers presented here are estimates, a speedup of three orders of magnitude not only shows a greatly improved solution to the original problem, but also a compelling argument in favor of the use of reconfigurable resources for highly parallel applications such as backprojection.

References

[1] A. Conti, B. Cordes, M. Leeser, E. Miller, and R. Linderman, "Adapting Parallel Backprojection to an FPGA Enhanced Distributed Computing Environment," *HPEC 2005*.

[2] P. Eugster, P. Felber, R. Guerraoiu, and A. M. Kermarrec, "The Many Faces of Publish/Subscribe," *ACM Computing Surveys*, 35(2):114-131, June 2003.

[3] M. Soumekh, Synthetic *Aperture Radar Signal Processing with MATLAB Algorithms*, John Wiley & Sons, New York NY, 1999. ISBN 0-471-29706-2.