Appears in Int. Conf. on High-Performance Computing 1999

I/O Implementation and Evaluation of Parallel Pipelined STAP on High Performance Computers

Wei-keng Liao[1], Alok Choudhary[2], Donald Weiner[1], and Pramod Varshney[1]

[1] EECS Department, Syracuse University, Syracuse, NY 13244
{wkliao, dweiner, varshney}@syr.edu
[2] ECE Department, Northwestern University, Evanston, IL 60208
choudhar@ece.nwu.edu

Abstract. This paper presents experimental performance results for a parallel pipeline STAP system with I/O task implementation. In our previous work, a parallel pipeline model was designed for radar signal processing applications on parallel computers. Based on this model, we implemented a real STAP application which demonstrated the performance scalability of this model in terms of throughput and latency. The parallel pipeline model normally does not include I/O task because the input data can be provided directly from radars. However, I/O can also be done through disk file systems if radar data is stored in disks first. In this paper, we study the effect on system performance when the I/O task is incorporated in the parallel pipeline model. We used the parallel file systems on the Intel Paragon and the IBM SP to perform parallel I/O and studied its effects on the overall performance of the pipeline system. All the performance results shown in this paper demonstrated the scalability of parallel I/O implementation on the parallel pipeline STAP system.

1 Introduction

In this paper we build upon our earlier work where we devised strategies for high performance parallel pipeline implementations, in particular, for Space-Time Adaptive Processing (STAP) applications [1, 2]. A modified Pulse Repetition Interval (PRI)-staggered post-Doppler STAP algorithm was implemented based on the parallel pipeline model and scalable performance was obtained both on the Intel Paragon and the IBM SP. Normally, this parallel pipeline system does not include disk I/O costs. Since most radar applications require signal processing in real time, thus far we have assumed that the signal data collected by radar is directly delivered to the pipeline system.

In practice, the I/O can be done either directly from a radar or through disk file systems. In this work we focus on the I/O implementation of the parallel pipeline STAP algorithm when I/O is carried out through a disk file system. Using existing parallel file systems, we investigate the impact of I/O on the overall pipeline system performance. We ran the parallel pipeline STAP system portably and measured the performance on the Intel Paragon at California Institute of Technology and on the IBM SP at Argonne National Laboratory (ANL.) The parallel file systems on both the Intel Paragon and the IBM SP contain multiple stripe directories for applications to access disk files efficiently. On the Paragon, two PFS file systems with different stripe factors were tested and the results were analyzed to assess the effects of the size of the stripe factor on the STAP pipeline system. On the IBM SP, the performance results were obtained by using the native parallel file system, PIOFS, which has 80 stripe directories.

Comparing the two parallel file systems with different stripe sizes on the Paragon, we found that an I/O bottleneck results when a file system with smaller stripe size is used. Once a bottleneck appears in a pipeline, the throughput which is determined by the task with maximum execution time degrades significantly. On the other hand, the latency is not significantly affected by the bottleneck problem. This is because the latency depends on all the tasks in the pipeline rather than the task with the maximum execution time.

The rest of the paper is organized as follows: in Section 2, we briefly describe our previous work, the parallel pipeline implementation on a STAP algorithm. The parallel file systems tested in this work are described in Section 3. The I/O design and implementation are given in Section 4 and the performance results are given in Section 5.

2 Parallel pipeline STAP system

In our previous work [1], we described the parallel pipelined implementation of a PRI-staggered post-Doppler STAP algorithm. The parallel pipeline system consists of seven tasks: 1)Doppler filter processing, 2)easy weight computation, 3)hard weight computation, 4)easy beamforming, 5)hard beamforming, 6)pulse compression, and 7)CFAR processing. The design of the parallel pipelined STAP algorithm is shown in Figure 1.

The input data set for the pipeline is obtained from a phased array radar and is formed in terms of a coherent processing interval (CPI). Each CPI data set is a 3-dimensional complex data cube. The output of the pipeline is a report on the detection of possible targets. Each task i, $0 \leq i < 7$, is parallelized by evenly partitioning its work load among P_i compute nodes. The execution time associated with task i is T_i. For the computation of the weight vectors for the current CPI data cube, data cubes from previous CPIs are used as input data. This introduces temporal data dependency. Temporal data dependencies are represented by arrows with dashed lines in Figure 1 where $TD_{i,j}$ represents temporal data dependency of task j on data from task i. In a similar manner, spatial data dependencies $SD_{i,j}$ can be defined and are indicated by arrows with solid lines.

Throughput and latency are two important measures for performance evaluation on a pipeline system.

$$throughput = \frac{1}{\max\limits_{0 \leq i < 7} T_i}. \tag{1}$$

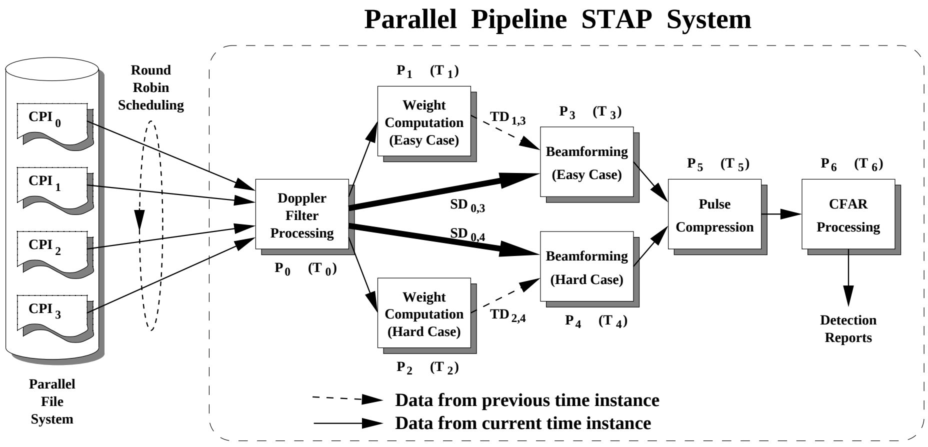

Fig. 1. The parallel pipelined STAP system. Arrows connecting task blocks represent data transfer between tasks. I/O is embedded in the Doppler filter processing task

$$latency = T_0 + \max(T_3, T_4) + T_5 + T_6. \tag{2}$$

Equation (2) does not contain T_1 and T_2. The temporal data dependency does not affect the latency because weight computation tasks use CPI data from the previous time instance rather than the current CPI. The filtered CPI data sent to the beamforming task does not wait for the completion of its weight computation. A detailed description of the STAP algorithm we used can be found in [3, 4].

3 Parallel file systems

We used the parallel I/O library developed by Intel Paragon and IBM SP systems to perform read operations. The Intel Paragon OSF/1 operating system provides a special file system type called PFS, for Parallel File System, which gives applications high-speed access to a large amount of disk storage [5]. In this work, two PFS file systems at Caltech were tested : one has 16 stripe directories (stripe factor 16) and the other has a stripe factor of 64. We used the Intel Paragon NX library to implement the I/O of the parallel pipeline STAP system. Subroutine `gopen()` was used to open CPI files globally with a non-collected I/O mode, `M_ASYNC`, because it offers better performance and causes less system overhead. In addition, we used asynchronous I/O function calls: `iread()` and `ireadoff()` in order to overlap I/O with the computation and communication.

The IBM AIX operating system provides a parallel file system called Parallel I/O File System (PIOFS) [6]. There are a total of 80 slices (striped directories) in the ANL PIOFS file system. IBM PIOFS supports existing C read, write, open and close functions. However, unlike the Paragon NX library, asynchronous parallel read/write subroutines are not supported on IBM PIOFS. The overall performance of the STAP pipeline system will be limited by the inability to overlap I/O operations with computation and communication.

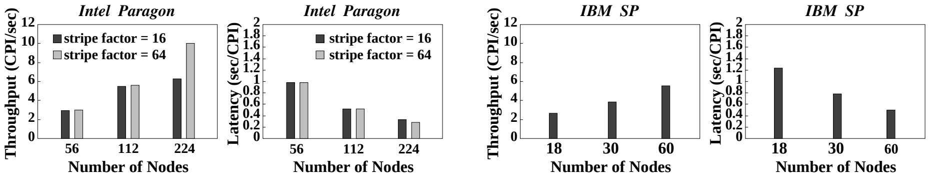

Fig. 2. Performance results for the STAP pipeline system with parallel I/O embedded in the Doppler filter processing task

4 Design and implementation

A total of four CPI data sets stored as four files in the parallel file systems were used on both the Caltech Paragon and the ANL SP. Each of the four CPI files is of size 8M bytes. All nodes allocated to the first task (the I/O nodes) of the pipeline read exclusive portions of each CPI file with proper offsets. Because the number of I/O nodes may vary due to different node assignments to the I/O task, the length of data for the read operations can be different. The read length and file offset for all the read operations are set only during the STAP pipeline system's initialization and is not changed afterward. Therefore, in each of the following iterations, only one read function call is needed. The design for the I/O task implemented in the STAP pipeline system is illustrated in Figure 1.

5 Performance results

In the I/O implementation on the Paragon, the Doppler filter processing task reads its input from CPI files using asynchronous read calls. A double buffering strategy is employed to overlap the I/O operations with computation and communication in this task. Figure 2 shows the timing results for this implementation on the Paragon PFS file system with 16 and 64 stripe directories. Three cases of node assignments to all tasks in the pipeline system are given, each doubles the number of nodes of another. In the case of using the PFS with 16 stripe directories, the throughput scales well in the first two cases, but degrades when the total number of nodes goes up to 224. In this case, the I/O operations for reading CPI data files here become a bottleneck for the pipeline system. This bottleneck forces the rest of the following tasks in the pipeline system to wait for their input data from their previous tasks. On the other hand, both throughput and latency showed linear speedups when using the PFS with 64 stripe directories. In the case with 224 nodes, the I/O bottleneck is relieved by a PFS with more stripe directories.

From the latency results, a linear speedup was obtained. The I/O bottleneck problem does not affect the latency significantly. Unlike the throughput that depends on the maximum of the execution times of all the tasks, the latency is determined by the sum of the execution times of all the tasks except for the tasks with temporal dependency. Therefore, even though the execution time of the Doppler filter processing task is increased, the delay does not contribute much to the latency.

The timing results for the IBM SP at ANL are also given in Figure 2. Because PIOFS does not provide asynchronous read/write subroutines, the I/O operations do not overlap with computation and communication in the Doppler filter processing task. Hence, the performance results for throughput and latency on the SP did not show the scalability as on the Paragon, even though the SP has faster CPUs.

6 Conclusions

In this work, we studied the effects of parallel I/O implementation on the parallel pipeline system for a modified PRI-staggered post-Doppler STAP algorithm. The parallel pipeline STAP system was run portably on Intel Paragon and IBM SP and the overall performance results demonstrated the linear scalability of our parallel pipeline design when the existing parallel file systems were used in the I/O implementations. On the Paragon, we found that a pipeline bottleneck can result when using a parallel file system with a relatively smaller stripe factor. With a larger stripe factor, a parallel file system can deliver higher efficiency of I/O operations and, therefore, improve the throughput performance. The performance results demonstrate that the parallel pipeline STAP system scaled well even with a more complicated I/O implementation.

7 Acknowledgments

This work was supported by Air Force Materials Command under contract F30602-97-C-0026. We acknowledge the use of the Intel Paragon at California Institute of Technology and the IBM SP at Argonne National Laboratory.

References

1. A. Choudhary, W. Liao, D. Weiner, P. Varshney, R. Linderman, and M. Linderman. Design, Implementation and Evaluation of Parallel Pipelined STAP on Parallel Computers. *International Parallel Processing Symposium*, 1998.
2. W. Liao, A. Choudhary, D. Weiner, and P. Varshney. Multi-Threaded Design and Implementation of Parallel Pipelined STAP on Parallel Computers with SMP Nodes. *International Parallel Processing Symposium*, 1999.
3. R. Brown and R. Linderman. Algorithm Development for an Airborne Real-Time STAP Demonstration. *IEEE National Radar Conference*, 1997.
4. M. Linderman and R. Linderman. Real-Time STAP Demonstration on an Embedded High Performance Computer. *IEEE National Radar Conference*, 1997.
5. Intel Corporation. *Paragon System User's Guide*, April 1996.
6. IBM Corp. *IBM AIX Parallel I/O File System: Installation, Administration, and Use*, October 1996.