Office of Strategic National Alien Planning

Parallel Kalman Filters

Negative results count, right?

Project Team:
REDACTED
Brian J. Gravelle
REDACTED
REDACTED

June 9, 2017

0.1 Executive Summary

The Office of Strategic National Alien Planning (OSNAP) has made a substantial investment in a Secure Alien Surveillance System (SASS). As part of the acquisition, control software is required to track potential targets in surveillance video using Kalman Filters.

This report details a study into how the performance of Kalman filters can be improved through parallel execution. Several versions of kalman filters were produced with different level and techniques for parallelization. The underlying linear algebra was parallelized in two ways: using OpenMP and using the Intel compiler's built in parallelization tools. The Kalman filter built on top of this linear algebra was parallelized using OpenMP and tested with each of the linear algebra designs. The results indicate that Kalman filters would not make a good target for thread level parallelization.

0.2 Original Description

The overarching goal of this project is to design and construct a prototype of an alien surveillance system for OSNAP. The surveillance system will analyze separately collected video on various server architectures to take advantage of larger core counts and advanced architecture features. The matrix computations associated with Kalman Filters and independent nature of tracking separate objects present excellent possibilities for parallelism.

The matrix calculations used in the computation of the Kalman Filters can make use of Intel's vector instructions to exploit small-scale data parallelism. This processes will be completed using the vector operations associated with Cilk. Similarly, thread level parallelism can be exploited in the area of tracking. Each tracked object requires a separate Kalman Filter operating on almost entirely independent data which should allow for significant performance improvement through thread level parallelism. TBB will be used to implement this improvement.

In figure 1 of the system architecture, it is obvious that the main area of interest for this project is the design and implementation of the object tracking portion of the system since it is in parallel. Although sources of parallelism exist in other aspects of the system those will be left for future work. The object tracking section can be parallelized by treating individual objects separately and by utilizing vector architectures to parallelize the underlying calculations.

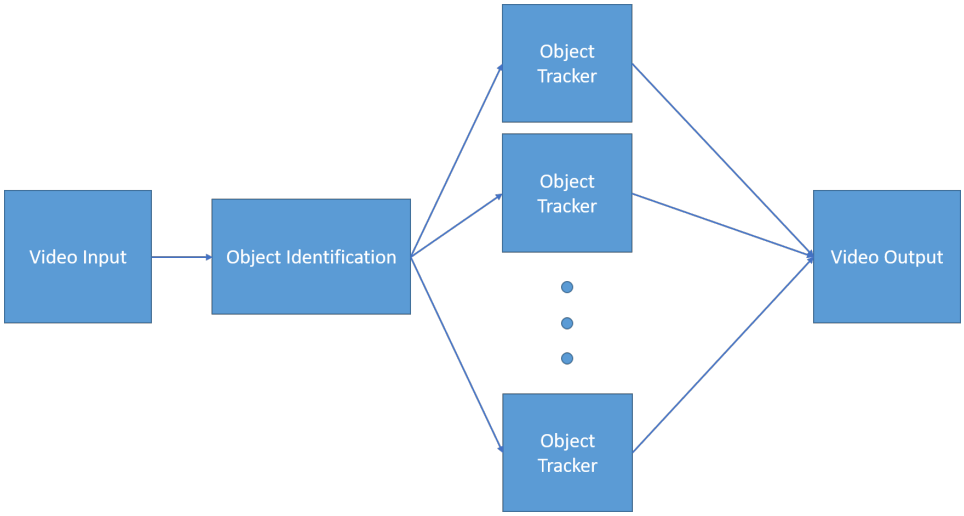

Figure 1: General system architecture

0.3 Methodology

Due to limited time frame this report is limited to the Kalman filter portion of the proposed project. The other aspects will be constructed and available in the future, provided continued operation of the OSNAP program.

Kalman filters are a mathematic way to improve understanding of a time-series linear system. These systems can generally be measured at time steps and have predictive mathematic models, but both of these options have noise or error associated with them. Kalman filters combine the two and use the error values to adjust how much each is weighted at a given time step.

The Kalman filter can be parallelized at two levels. It is based on a series of linear algebra equations many of which are independent of each other. The independence of the different equations suggested that they would be good candidates for task-level parallelism. These equations are based on dense linear algebra equations which have a long history of being efficiently parallelized. This study explored both of these options.

0.3.1 Equation Parallelism

The Kalman equations (eqs. 1- 5) can be decomposed into seven sections (figure 2) with some dependencies between them. These sections were then

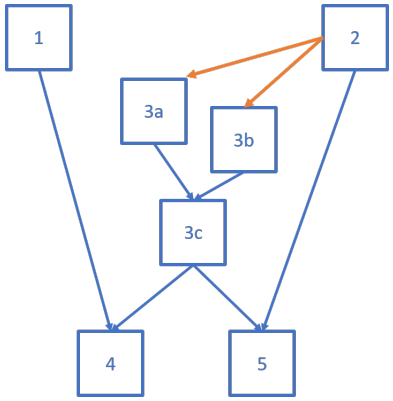

Figure 2: Task graph of Kalman filter, number correspond to the equation in that section. Blue indicates dependencies programmed into the system. Red indicates dependencies erroneously ignored.

sent to separate threads to run concurrently. Several of the equations rely on data computed in previous equations meaning that the computation was divided into two parallel sections. The first of these ran four parallel tasks while the second ran two and the seventh task had to occur sequentially. Additionally, each time step was serialized since they rely heavily on each other.

$$\hat{x}_{new} = A\hat{x} \tag{1}$$

$$P = APA^T + Q \tag{2}$$

$$t_1 = PC^T \tag{3a}$$

$$t_2 = (CPC^T + R)^{-1} \tag{3b}$$

$$K = t_1 t_2 \tag{3c}$$

$$\hat{x} = \hat{x}_{new} + K(y - C\hat{x}_{new}) \tag{4}$$

$$P = (I - KC)P \tag{5}$$

0.3.2 Linear Algebra Parallelism

Linear algebra operations are highly parallel in that many of the row or elements operations are completely independent of each other, enabling parallel computation. This feature of linear algebra is commonly exploited in scientific computation, graphics and many other domains. It can be applied

using hardware accelerators (GPUs, vector instructions, etc), shared memory threads, or distributed processors.

For this report we parallelized the linear algebra functions by applying simple OpenMP pragmas to each function. Primarily this resulted in the parallelization of the outermost for loop of each operation. Some reductions were used as well in the determinant and LU factorization. Generally the parallelization did not include nesting of omp parallel regions, but some resulted because of functions calls within the linear algebra library. For example, the cofactor function includes calls to the LU factorization both of which include omp regions.

For comparison, the library was compiled using the automatic parallelization feature available with Intel's compiler. This feature is intended to automatically exploit parallelism without the assistance of the programmer.

0.3.3 Experimental Environment

Experiments were run on a single node of the Cerberus cluster. These nodes consist of a quad-core Intel i5 Processor running at 1.3GHz with 16GB of memory and 32KB of L1 cache. Results were recorded using TAU.

For the full Kalman filter experiments we generated 1000 points of noisy projectile motion data. This data is run through the Kalman filter and the process is repeated 1000 times to average out measurement noise. The Kalman filter was tested in four configurations: 1) purely sequential; 2) parallel Kalman with sequential linear algebra; 3) parallel Kalman with OMP linear algebra; and 3) parallel Kalman with Intel parallelized linear algebra.

Similarly, for the linear algebra functions were tested to see how the parallelization affected them independent of the Kalman filter. For these experiments small (6x6) matrices were randomly generated and computations applied to them. Each of the functions was applied to the matrices 100000 times so that the length of computation would be sufficient to compare.

0.4 Results

The results of the study were disappointing to say the least. Timing data for each of the kalman filter versions is shown in figure 3; figure 4 is the same graph without the Parallel-Kalman, OMP-LA version to make it more legible. These graphs show that the Kalman update suffers a 10x or 100x

slow down compared to the sequential code. It is clear that other methods of performance improvement should be considered for this part of the program.

In figures 5 and 6 the results of the linear algebra only experiments are shown. These again demonstrate that attempts to parallelize the operations using OpenMP were largely unsuccessful, but the Intel compiler's parallel flag did improve the performance of the Linear Algebra operations.

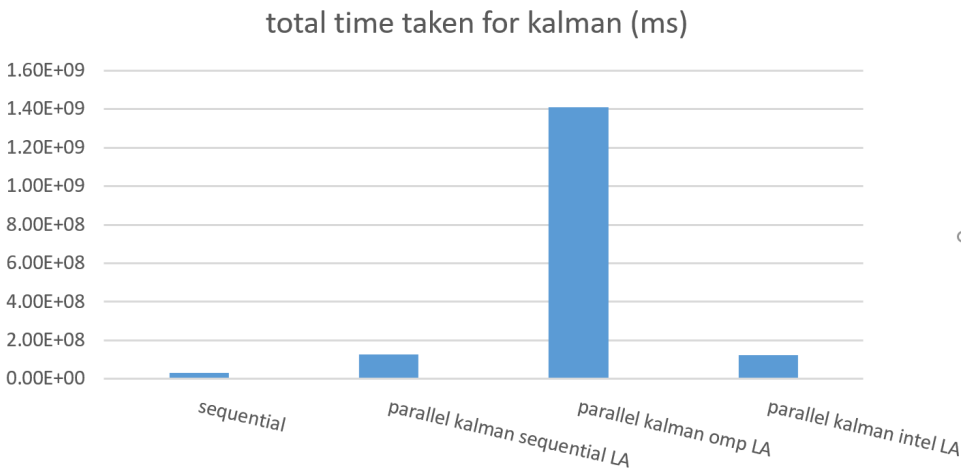

Figure 3: Timing results of each Kalman implementation

There are several possible reasons for the slowdown. First, the matrices involved are quite small, so the overhead of creating the threads may overwhelm the gains causing the slowdown. The extreme case of parallelizing Kalman filters and Linear algebra with OpenMP results in nested thread that adds to scheduling overheads as well.

The other primary suspect for poor performance is programmer error; the DAG was incorrectly designed and the error not noticed until the writing of this report. Figure 2 indicates the dependencies of the system with those in red left out of the implementation. It seems likely that OpenMP recognized the dependence and refused to parallelize that section.

Notably the Intel parallelization did not create new threads but still managed to improve over the serial linear algebra. This is likely the result of vectorization of some of the matrix operations. Although these improvements did not translate to the Kalman filter it suggests that there is more work to be done in that area.

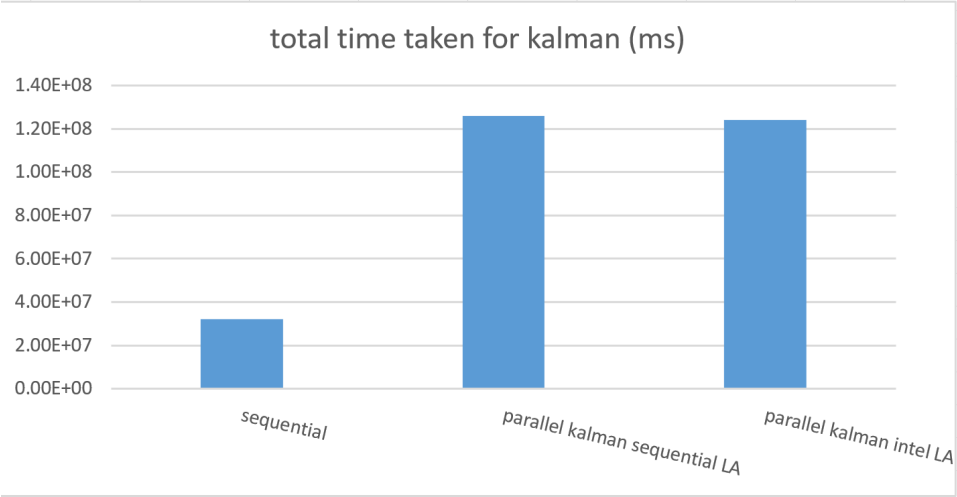

Figure 4: Timing results of each Kalman implementation

0.5 Alternatives and Future Work

While these experiments produced largely negative results, there are still options available to improve the performance of the system. The Intel compiler results suggest that vectorization provides good opportunities for improvement. This could be furthered with unrolling loops, removing dependencies, and rearranging data to make it slightly more friendly to vector instructions.

This project presented a good learning opportunity for those involved. It was a clear case study for the importance of analyzing a problem theoretically before diving into the programming. Had the authors done a better job of this prior to the project we would have not attempted to parallelize the matrix operations through threads, but focused on the potential for pipelining the broader application.

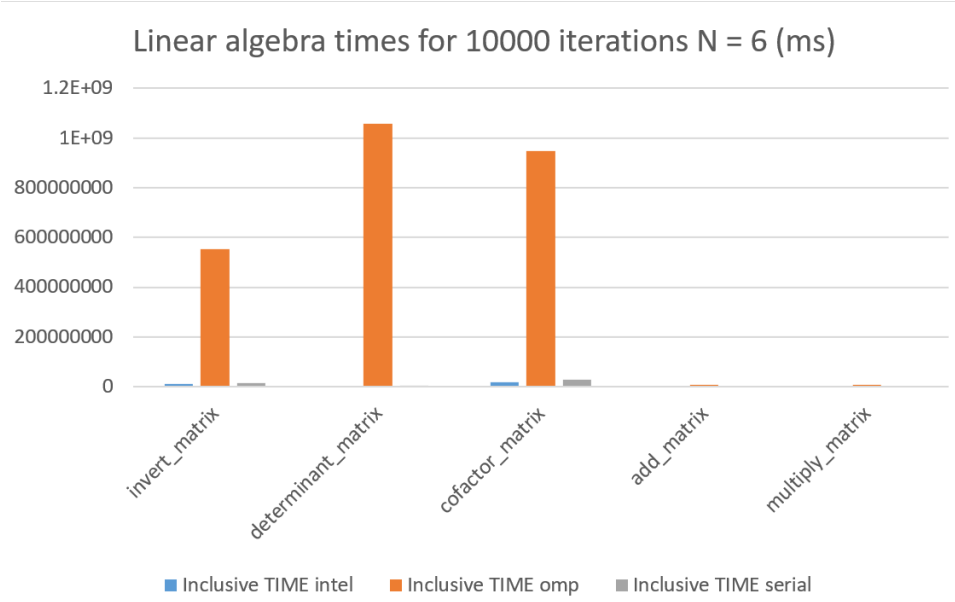

Figure 5: Timing results of each Linear Algebra experiments

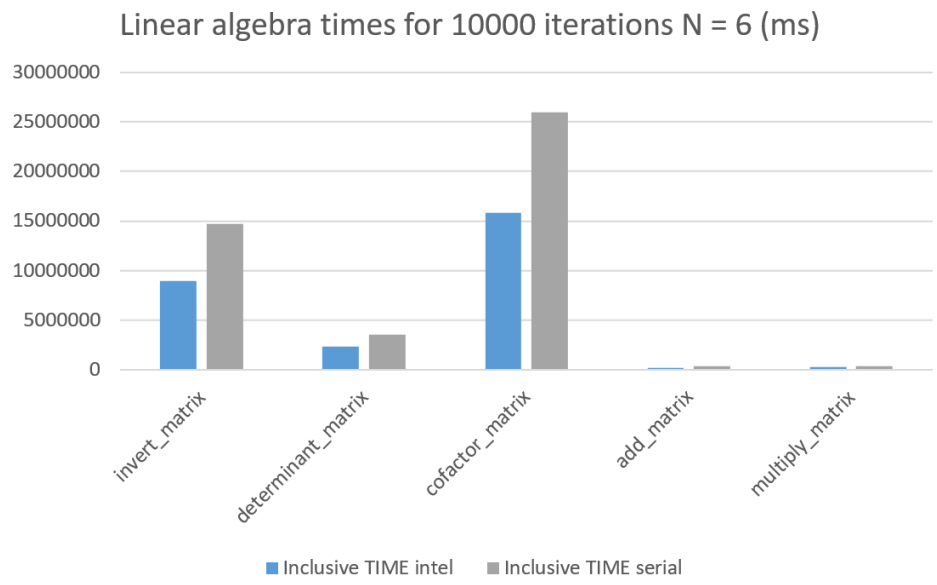

Figure 6: Timing results of each Linear Algebra experiments