# Parallel MPI/C++ Programming for the Kalman Filter

*Myint Myint Thein*[*1], *Pho Kaung*[2]

[*1.]*Department of Computer Studies, Dagon university, Yangon, Myanmar*
[2]*Universities' Research Centre, University of Yangon, Myanmar*

**Abstract**: The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) means to estimate the state of a process. Computational Kalman equations can be described with the process of sequential C++ and parallel Message Passing Interface (MPI)/C ++program. Parallel implementation of Kalman filter has been suggested to improve the execution time. MPI is applied to the computation of parallel Kalman filter. Distributed memory implementation programs are compiled and executed on Windows High Performance Computing (HPC) Server 2008 with Microsoft Visual Studio 2012 including MPI.

**Keywords**: Sequential Kalman filter, Message Passing Interface , Parallel Kalman filter
**PACS** : 02 Mathematical Physics

## INTRODUCTION

The Kalman filter is named after Kalman .R.E, who in 1960 published his famous paper describing a recursive solution to the discrete-data linear filtering problem [1]. A Kalman filter is a linear estimator. It is used to estimate the state of a linear dynamic system by using measurements linearly related to the state of the system but corrupted with noise. It is a recursive data processing algorithm or a software tool that does not require all previous data to be kept in memory. Kalman filtering shows how the incoming raw measurements of signal should be processed to produce the best parameter estimates as a function of time[6].

The Kalman filter exploits the dynamics of the target, which govern its time evolution, to remove the effects of the noise and get a good estimate of the location of the target at the present time (filtering), at a future time (prediction), or at a time in the past (interpolation or smoothing)[2]. Parallel processing offers speed-up or higher and reliable performance at affordable prices to the implementation of Kalman Filter.

Olivier Cadet, 2003 September is presented "Introduction to Kalman Filter and its Use in Dynamic Positioning Systems"[5]. Landry, C., 2008[3] found that the references of working with ever-changing accelerations, current samples of acceleration are referred with the constant, "K," and modify Kinematic Equations:

$$v(K) = v(K-1) + a(K)t$$

$$s(K) = s(K-1) + \frac{1}{2} a(K)t^2$$

Moreover , Landry, C., 2008[3] studied that fixing the accelerometer error by the gyroscope data. One way to take out better information from accelerometers is to use a complicated form of time dependent probability theory. This is known as Kalman Filtering.

The Kalman filter process has two steps: the (time update) prediction step, where the next state of the system is predicted given the previous measurement (the angular velocity measurement1 from the gyroscope), and the (measurement) update step, where the current state of the system is estimated given the measurement2 (the angle from the accelerometer) at that time step. The steps translate to equations using matrix algebra and statistics. They are listed as follows [12].

| | |
|---|---|
| *u =measurement1* | Read the value of the last measurement from the gyroscope |
| Time Update: *x = A · x + B ·u* | Update the state *x* of our model |
| *y =measurement2* | Read the value of the second measurement/real value. Here this will be the angle from the accelerometer. |
| Measurement Update: *Inn = y – C ·* x | Calculate the difference between the second value and the value predicted by the model. This is called the innovation |
| *s = C · P · C' +Sz* | Calculate the covariance |
| *K = A · P · C' ·* inv(_s_) | Calculate the Kalman gain |
| *x = x + K · Inn* | Correct the prediction of the state |
| *P = A · P · A' –* | Calculate the covariance of |

| | |
|---|---|
| $K \cdot C \cdot P \cdot A' + Sw$ | the prediction error |

Computational Performance: The algorithms' computational performance was measured in terms of the following measures, commonly used in parallel computing:

Speed up = Serial Execution Time / Parallel execution Time

The speedup characterizes the scalability of a parallel algorithm. Ideally, the best speedup is linear. The efficiency characterizes how well the processors are utilized. Ideally, it has values between 0 and 1. Algorithms with linear speedup have efficiency of 1 (the best)[10].

## PARALLAL KALMAN FILTER

The parallel Kalman filter is computationally intensive and hence complex due to the inherent matrix operations. Often, the filter cannot be computed in real time when the system has a large number of state variables. A method is discussed for achieving almost real-time performance. In addition, a method for determining stability of the Kalman filter is presented. This method involves the use a paradigm that achieves high performance and close to real time results while maintaining accuracy for the Kalman filters. In addition, the paradigm shows an improvement of the processor utilization over other implementations and provides flexibility in terms of the hardware used for implementation. In order to speed up KF computations, parallel processing is performed at two levels:(1) the predictor and corrector equations of the KF are decoupled so that the predictor and corrector can be computed on separate processors.(2) the measurement data are pipelined into each processor. Therefore, both multiprocessing and pipelining are considered to achieve large improvements in computational speed[11].

The MPI is the most widely used parallel programming paradigm, particularly for distributed memory parallel computers. The MPI provides the user with a programming model in which processes communicate with other processes by explicitly calling library routines to send and receive messages. The model for parallelism is related relatively closely to the hardware the model runs on. MPI is appropriate for clusters, where the memory (and processors) are distributed. The advantages of the MPI programming model include the programmer's complete control over data distribution, process synchronization, explicit communication, and a permitting of the optimization of data locality . This gives MPI programs high performance and scalability; however, it also has the drawback of making the MPI difficult to program and debug[10].

## KALMAN FILTERING WITH THE MPI/C++ PROGRAMMING

```
int main(int argc, char **argv)
{
 int i;
 int  rank;
 int  np,modu,recvsize;
 double starttime;
MPI_Init(&argc, &argv);// Initialization
MPI_Comm_size(MPI_COMM_WORLD, &np);//
find total number of processes
MPI_Comm_rank(MPI_COMM_WORLD, &rank);//
find process label (rank) in group
if(rank==0)
{
     modu =SAMPLE_COUNT%np;
   recvsize=SAMPLE_COUNT/np;
             for (i = 0; i < SAMPLE_COUNT; ++i)
    {
       // Get the gyro and accelerometer input.
       sample_datar[i][0] = sample_data[i][0];
       sample_datar[i][1] = sample_data[i][1];
     }
}
starttime=MPI_Wtime();
MPI_Bcast(&recvsize, 1, MPI_INT, 0,
MPI_COMM_WORLD);//
MPI_Bcast(&modu, 1, MPI_INT, 0,
MPI_COMM_WORLD);//
cout<<"now printing the receive size and the
rank"<<endl;
cout<<"value of receieve is  "<<recvsize<<"   and
rank is"<<rank<<endl;
MPI_Scatter(&sample_datar,recvsize,
MPI_DOUBLE, recvbuf,recvsize, MPI_DOUBLE, 0,
MPI_COMM_WORLD);//
 for (i = 0; i < SAMPLE_COUNT; ++i)
 //for (i = 0; i < recvsize; i++)
    {
     // Get the gyro and accelerometer input.
      gyro_input[i] = sample_datar[i][0];
      accel_input [i]= sample_datar[i][1];
    // Update the Kalman filter and get the output.
             recvbuf[i]=kalman_update(gyro_input[i],
accel_input[i]);
    }
             MPI_Gather(recvbuf,recvsize ,
MPI_DOUBLE, &sample_datar[modu][modu],
recvsize, MPI_DOUBLE, 0, MPI_COMM_WORLD);//

             if(rank==0)
             {
              for (i = 0; i < SAMPLE_COUNT; ++i)
               {
                printf("%d gyro=%7.5f deg/sec
accel=%7.5f deg      kalman_output=%5.5f deg\n",
i,gyro_input[i],accel_input[i],recvbuf[i]);
        }
            }
                printf("Done! The elapse time for
Kalam Filter using MPI  Kalman_Using_mpi_2.cc is
%g seconds. \n",MPI_Wtime()-starttime);//
MPI_Finalize();//End Up
   return 0;
}
```

## RESULTS AND DISCUSSION

**Table1**. The execution time for Serial and MPI programs

| Program | Executing Time(s) |
|---|---|
| Serial | 4.025 |
| MPI | 0.0014 |

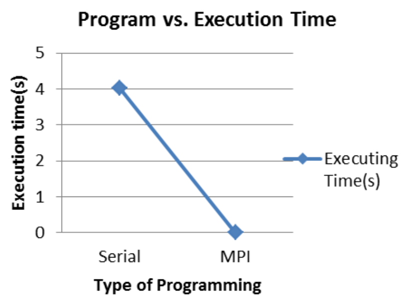

**FIGURE 1**. Comparison of execution time for Serial and MPI

Time taken for serial and parallel MPI program are described in Table1. Figure 1 shows that the parallel MPI computing has shorter time span than serial computing. The parallel computing is more suitable for better performance. Table2. and Figure2. show that an execution time depends on the number of CPUs which numbered are represented 8,12 and 16. In addition, the execution times are shorter while the number of CPUs increased.

**TABLE 2**. Execution time for 8 CPUs, 12 CPUs, 16 CPUs using MPI

| No. of CPUs | Execution Time(s) |
|---|---|
| 8 | 0.00193403 |
| 12 | 0.00167983 |
| 16 | 0.00128992 |

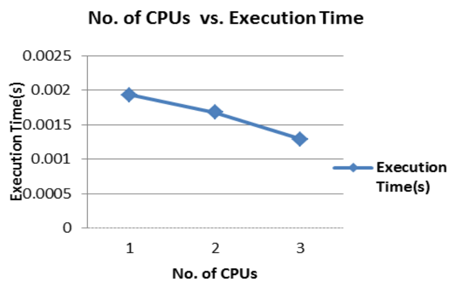

**FIGURE 2.** Executing time for 8CPUs, 12 CPUs, 16CPUs using MPI program

**TABLE 3.** The relation of multiple processors and their speedup

| No. of CPUs | Speed Up |
|---|---|
| 8 | 2081.14 |
| 12 | 2396.07 |
| 16 | 3120.34 |

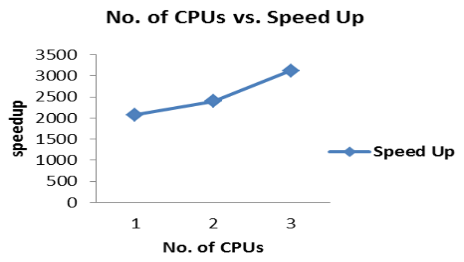

**FIGURE 3.** Speed up for multiple processors

The relationship between the number of CPUs and speedup is shown in Figure(3). The speedup of CPUs keep on increasing by adding the number of CPU using the implementation of parallel kalman filter. MPI method is more suitable to execute the large numbers of data. Parallel Kalman filter could reduce the computational time. When the problem is less amount of data, the executing time for MPI is longer than OpenMP. When using the same amount of sample data 512, the time taken for OpenMP is faster than MPI. The analysis can be seen at Figure 4 and Table 4. The execution time for table1 and table4 are computed with amount of sample data 512 and table 2 and table 3 are computed with sample data 1024.

**TABLE 4.** The comparison of the execution time of OpenMP and MPI program

| program | Executing Time(s) |
|---|---|
| OpenMP | 0.000955 |
| MPI | 0.0014 |

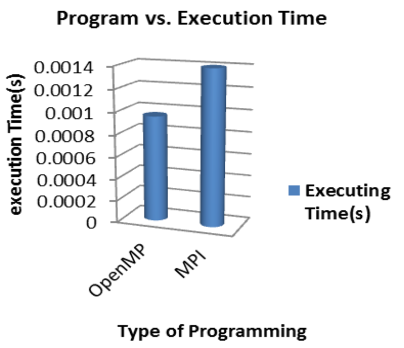

**FIGURE 4**. The relation of type of programming and execution time

## CONCLUSION

The results finding are: (1) the parallel (MPI) computing has shorter time span than serial computing. (2) while the number of CPUs increased, the execution times are shorter so that an execution time depends on the number of CPUs. (3) the speedup keeps on increasing by adding the number of CPUs.(4) the problem is less amount of data, the executing time for OpenMP is faster than MPI. The increased speed of parallel processing holds special advantages for real-time systems. A parallel system increases reliability through simple redundancy. This often requires major changes to the software. With a parallel system, increased capability can be added with additional processors. Multiple processors solve a large problem faster than a single high-speed processor. In general, parallel systems can claim a performance advantage over traditional systems. The era of practical parallel programming has arrived, marked by the popularity of the MPI and OpenMP software standards and the emergence of commodity clusters as the hardware platform of choice for an increasing number of organizations. It also demonstrates, through a wide range of examples, how to develop parallel programs that will execute efficiently on today's parallel platforms.

## ACKNOWLEDGEMENT

The author is deeply indebted to Dr Pho Kaung, Pro-rector/ Head, Universities' Research Center, University of Yangon for all his valuable guidance. I am also grateful to Dr. Zeya Oo, Senior Lecturer, Department of Electrical and Computer Engineering, Curtin University, Sarawak, Malaysia for useful suggestions for improving this paper.

## REFERENCES

1 R. E. Kalman., *"A New Approach to Linear Filtering and Prediction Problems," Transaction of the SME-Journal of Basic Engineering,* 1960.

2 M. Laaraiedh, "*Implementation of Kalman Filter with Python Language". IETR Labs, University of Rennes,* 2009

3 C. Landry, Papasideris, K., Sutter, B., and Wilson, A., "*Inertial Navigation Systems: The Physics behind Personnel Tracking and the ExacTrak System",* P.W.L.S. Innovations, 2008.

4 W. J. Lemanski, "*Parallel ADA Implementation of a Multiple model Kalman Filter Tracking System:*ASoftware Engineering approach", Thesis, ( Ohio: Air Force Institute of Technology), 1989.

5 S. G. Mohinder, A. P.Andrews, *"Kalman Filtering : Theory and Practice Using Matlab*" ( New York:John Wiley ) , 2001.

6 Olivier Cadet, "*Introduction to Kalman Filter and its Use in Dynamic Positioning Systems"* Dynamic Positioning Conference (Houston: Transocean Offshore Deepwater Drilling Inc.), 2003.

7 K. Rameshbabu, J. Swarnadurga, G. Archana, K. Menaka, *"Target Tracking System Using Kalman Filter"*. International Journal of Advanced Engineering Research and Studies,Vol.II, Issue I, 2012.

8 O. Rosén,, and A. Medvedev, "*Efficient Parallel Implementation of a Kalman Filter for Single Output Systems on Multicore Computational Platforms",* 50th IEEE Conference on Decision and Control and European Control Conference, Orlando, FL, USA, 2011 December.

9 P. Vesselin, J.J. Wu, *"Implementation and Performance of a Parallel Multi target Tracking Particle Filter",* 14th International Conference on Information Fusion Chicago, Illinois, USA, 2011.

10 J. Turk, *"Hybrid MPI+UPC parallel programming paradigm on an SMP cluster",* Elec Eng & Comp Sci, Vol.20, No.Sup.2, Department of Computer Engineering, Kadir Has University, 2012.

11 A. A. Walied, K.S. Assma, "*Parallel Kalman Filtering for Real –Time Signal Identification",* IJCCCE, Vol.6, NO.3, Baghdad University, 2006.

12 http://www.tom.pycke.be/mav/71/kalman-filterig-of-imu-data

13 http://www.cal.ypso.inesc-id.pt/FCUL/psm/docs/kalman-dan-simon