Title	Parallel OpenMP/C++ Programming for the Kalman Filter
All Authors	Myint Myint Thein
Publication Type	Local Publication
Publisher (Journal name, issue no., page no etc.)	Jour. Myan. Acad. Arts & Sc. 2014 Vol. XII. No.3
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 serial C and parallel OpenMP/C program. Parallel implementation of Kalman filter has been suggested to improve the execution time. Single Program Multiple Data (SPMD) and Work Sharing methods of Open Multi-Processing (OpenMP) are applied to the computation of parallel Kalman filter. An evaluation of parallel algorithm on eight different shared-memory multi-core architectures has been performed. Shared memory implementation programs are compiled and executed on Windows Compute Cluster Server 2003 with Microsoft Visual Studio 2008.
Keywords	Kalman filter, OpenMP, Parallel Kalman Filter
Citation	
Issue Date	2014

Jour. Myan. Acad. Arts & Sc. 2014 Vol. XII. No.3

Parallel OpenMP/C++ Programming for the Kalman Filter

Myint Myint Thein[1]

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 serial C and parallel OpenMP/C program. Parallel implementation of Kalman filter has been suggested to improve the execution time. Single Program Multiple Data (SPMD) and Work Sharing methods of Open Multi-Processing (OpenMP) are applied to the computation of parallel Kalman filter. An evaluation of parallel algorithm on eight different shared-memory multi-core architectures has been performed. Shared memory implementation programs are compiled and executed on Windows Compute Cluster Server 2003 with Microsoft Visual Studio 2008.

Keywords: Kalman filter, OpenMP, Parallel Kalman Filter

Introduction

The Kalman filter is named after Kalman R.E, who in 1960 published his famous paper describing a recursive solution to 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. All previous "history" is in fact captured in the most recent estimate of the state of the system. This is an important characteristic when it comes to implementing this type of algorithm in computers. Finally this type of filter is optimal; it calculates the best possible estimate (minimum variance) for the state of the system. Kalman was able to prove useful dual properties of estimation and control for these systems. 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 applications of the Kalman Filtering in real world are diverse. An example application would are diverse. An example application would be providing accurate, continuously updated information about the position and velocity of an object given only a sequence of observations about its

[1]Lecturer(Head), Department of Computer Studies, Dagon University

position, each of which includes somr error. The Kalman filter exploits the dynamics of 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 Fiter[4]. The basic logic in parallel processing is to divide an unmanageable large task into smaller tasks, which are more manageable. The divided smaller tasks could then be run on multiple processors. In some case multiple processors solve a large problem faster than a single high-speed processor [6].

Mathematical Equations of Kalman Filter

Kalman filter addresses the general problem of trying to estimate the state $x \in \Re^n$ of a discrete-time controlled process that is governed by the linear stochastic difference equation

$x_{k+1} = Ax_k + Bu_k + w_k$ (1)

with a measurement $y \in \Re^m$ that is $y_k = Hx_k + v_k$ (2)

A, *B*, and *H* are matrices.

k is the time index.

x is called the state of the system.

u is a known input to the system.

y is the measured output.

w is the process noise.

v is the measurement noise.

The vector *x* contains all of the information about the present state of the system, but x cannot be measured directly.

$y = f(x)$

Then the noise covariance matrices p*(w)* and p(v) are defined as:

Process noise covariance: $p(w) \approx N(0,Q)$ (3)

Measurement noise covariance: $p(v) \approx N(0,R)$ (4)

In practice, the process noise covariance *Q* and measurement noise covariance *R* matrices might change with each time step or measurement. Q and R are assumed as constant. The $n \times n$ matrix *A* relates the state at the previous time step to the state at the current step, in the absence of either a driving function or process noise. The $n \times 1$ matrix *B* relates the optional control input $u \in \Re_l$ to the state x. The $m \times n$ matrix H in the measurement equation relates the state to the measurement y_k .

The Kalman filter process has two steps: the prediction step, where the next state of the system is predicted given the previous measurements, and the update step, where the current state of the system is estimated given the measurement at that time step. The steps translate to equations as follows:

Prediction

$Xk^- = A_{k-1} X_{k-1} + B_k U_k$ (5)

$Pk^- = A_{k-1} P_{k-1} A_{k-1}^T + Q_{k-1}$ (6)

Update Equation

$v_k = Y_k - H_k X_k^-$ (7)

$S_k = H_k P_k^- H_k^T + R_k$ (8)

$K_k = P_k^- H_k^T S_k^{-1}$ (9)

$X_k = X_k^- + K_k V_k$ (10)

$P_k = P_k^- - K_k S_k K_k^T$ (11)

where

V_k is the innovation or the measurement residual on time step k.

S_k is the measurement prediction covariance on time step k.

K_k is the filter gain, which tells how much the predictions should be corrected on time step k.

Computational Kalman Equation

Making Sense of the Raw Data

By tracking both the current angular velocity (gyroscope) and the current linear acceleration (accelerometer) of the system measured relative to the moving system, it is possible to determine the linear acceleration of the system in its inertial reference frame.

Accelerometer Data

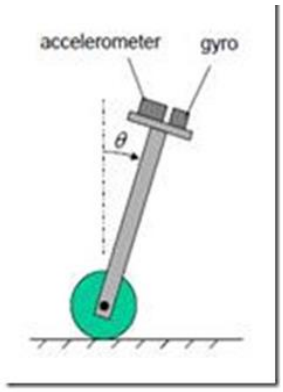


Figure (1) Accelerometers

Accelerometers measure the linear acceleration of a system in the inertial reference frame, but in directions that can only be measured relative to the moving system, since the accelerometers are fixed to the system and rotate with the system, but are not aware of their own orientation. In other words, an accelerometer measures the acceleration and gravity it experiences. Acceleration is the rate of change velocity, and velocity is the rate of change of the position [3].

Acceleration data can be converted – via some integration – into distance (with some error, which Kalman Filtering will take care of). Starting with the definition of instantaneous acceleration, a = dv/dt, which are rewritten as dv = a dt,

Taking the definite integral of both sides:

$$\int_{v=v_0}^{v} dv = \int_{t=0}^{t} a\, dt$$

giving

$$v - v_0 = at$$

Next, with the definition of instantaneous velocity,

$$v = ds/dt,$$

which can be rewritten as $ds = v\, dt$

again, taking the definite integral of both sides, and sub in for v_0.

$$\int_{s=s_0}^{s} ds = \int_{t=0}^{t} (v_0 + at)\, dt$$

giving,

$$s - s_0 = v_0 t + \frac{1}{2}at^2.$$

This double integration yields the Mechanical Physics Basic Kinematic Equations:

$$v - v_0 = at$$

$$s = s_0 + v_0 t + \frac{1}{2}at^2.$$

The accelerometer reads only changes in acceleration, for position (x) in terms of only x and a:

$$s = s_0 + \cancelto{0}{v_0} t + \frac{1}{2}at^2$$

yields:

$$s = s_0 + \frac{1}{2}at^2.$$

Finally, 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)+ ½ a(K)t

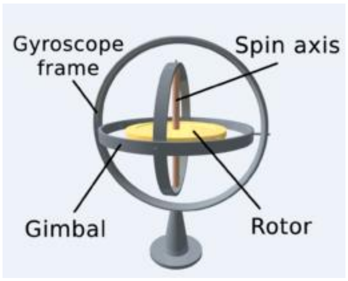
Accelerometer Error

An important thing to note about getting position from an accelerometer is that the error in position "integrates," meaning that if the noise or error in the accelerometer follows a normal distribution (overestimates and underestimates equally) then the position estimate should be reasonable. If however, the accelerometer is biased (tends to overestimate more than underestimate, or vice versa) then the error in your position estimate will grow exponentially. On top of this, ANY error is kept in your calculation through the iterative integration, so calculating position the accelerometer can have large errors. There are several error sources that cause an accelerometer output to deviate from its correct value. They are configuration (or misalignment) errors and the accelerometer errors embedded in the device itself. The configuration errors of an accelerometer are the *location* and *orientation* errors of the accelerometer. The error sources of a MEMS accelerometer are: *scale factor error*, *bias*, and *noise*[3].

Fixing the error associated with integrating

One way to eke out better information from accelerometers is to use a complicated form of time dependent probability theory. This is known as Kalman Filtering. Kalman Filtering is commonly used in the navigation systems of airplanes, where knowing the location accurately, and precisely if possible, is important.

Gyroscope Data



Figure(2) Gyroscopes

Gyroscopes measure the angular velocity of the system in its inertial reference frame. By using the original orientation of the system in the inertial reference frame as the initial condition and integrating the angular velocity, the system's current orientation is known at all times.

Gyroscopic data can be converted – via some integration – into angular attitude, or orientation (with some error, which Kalman Filtering will take care of). Starting with the definition of instantaneous velocity, the time rate of change of distance and velocity is found as, $dx = v_x\, dt$ with x being the position on the x-axis and v_x being the velocity along the x-axis. The same definition holds for angular motion. While velocity is the speed at which the position changes, angular velocity, ω, is nothing more than the rate at which the angle is changing, so

$$\partial\omega = \measuredangle rate = gyro\ output,$$,

Finally, knowing that the inverse of a derivative is an integral, we alter our equalities into:

$$\int \measuredangle rate = \int gyro\ output = \Delta\omega,$$,

In other words, integrating the gyroscope data, gives us the attitude angle, and since data from gyroscopes measure changes in degree of rotation as proportionally conditioned changes in voltage:

$$\Delta\omega \propto \Delta V$$

So with that knowledge, individual gyroscopes can be characterized simply by collecting ω vs. V data [3].

Summary of Kalman Equation

Kalman filtering is an iterative filter that requires two things. These two inputs consist of the gyroscope and accelerometer data.

$$x_{k+1} = A.\, x_k + B.u_k$$

$$\begin{pmatrix} alpha \\ bias \end{pmatrix}_{k+1} = \begin{pmatrix} 1 & -dt \\ 0 & 1 \end{pmatrix} \begin{pmatrix} alpha \\ bias \end{pmatrix}_k + \begin{pmatrix} dt \\ 0 \end{pmatrix} u_k$$

These are some formulas using matrix algebra and statistics. They are listed as follows:

u =measurement1	Read the value of the last measurement from the gyroscope
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 calculated from the accelerometer.
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’ – K · C · P · A’ +Sw	Calculate the covariance of the prediction error

Computational Kalman Equation with C programming

Sample data were gathered from the SparkFun IMU 5 Degrees of Freedom. The first data is the rate from the gyro (degrees/sec) and the second data is the accelerometer pitch attitude from horizontal in degrees.

```
 float gyro_input;    float accel_input;
float kalman_output;
double sample_data[SAMPLE_COUNT][2] = {{0.016088, 1.668337},...}
```

// **Update the State Estimation and compute the Kalman Gain**.

```
// The estimated angle is returned.
   float kalman_update(float gyro_rate, float accel_angle)
  {
  // Inputs.
    float u = gyro_rate;    float y = accel_angle;
 // Output.
    static float x_00 = 0.0;    static float x_10 = 0.0;
 // Persistant states.
    static float P_00 = 0.001;     static float P_01 = 0.003;
    static float P_10 = 0.003;     static float P_11 = 0.003;
 // Constants.

 // These are the delta in seconds between samples.
     const float A_01 = -0.019968;     const float B_00 = 0.019968;

 // Data read from 512 samples of the accelerometer had a variance of
0.07701688.
   const float Sz = 0.07701688;
 // Data read from 512 samples of the gyroscope had a variance of
0.00025556.
      const float Sw_00 = 0.001;     const float Sw_01 = 0.003;
      const float Sw_10 = 0.003;     const float Sw_11 = 0.003;
// Temp.
    float s_00;     float inn_00;     float K_00;     float K_10;     float AP_00;     float
AP_01;
    float AP_10;     float AP_11;     float APAT_00;     float APAT_01;     float
APAT_10;
    float APAT_11;     float KCPAT_00;     float KCPAT_01;     float KCPAT_10;
    float KCPAT_11;
// Update the state estimate by extrapolating current state estimate with
input u.
//  x = A * x + B * u
 x_00 += (A_01 * x_10) + (B_00 * u);
// Compute the innovation -- error between measured value and state.
// inn = y - c * x
    inn_00 = y - x_00;
// Compute the covariance of the innovation.
// s = C * P * C' + Sz
   s_00 = P_00 + Sz;
 // Compute AP matrix for use below.
 // AP = A * P
     AP_00 = P_00 + A_01 * P_10;      AP_01 = P_01 + A_01 * P_11;
    AP_10 = P_10;                          AP_11 = P_11;
// Compute the kalman gain matrix.
// K = A * P * C' * inv(s)
   K_00 = AP_00 / s_00;    K_10 = AP_10 / s_00;
 // Update the state estimate.
 // x = x + K * inn
    x_00 += K_00 * inn_00;
    x_10 += K_10 * inn_00;
// Compute the new covariance of the estimation error.
 // P = A * P * A' - K * C * P * A' + Sw
    APAT_00 = AP_00 + (AP_01 * A_01);     APAT_01 = AP_01;
    APAT_10 = AP_10 + (AP_11 * A_01);     APAT_11 = AP_11;
   KCPAT_00 = (K_00 * P_00) + (K_00 * P_01) * A_01;      KCPAT_01 = (K_00 *
   P_01);          KCPAT_10 = (K_10 * P_00) + (K_10 * P_01) * A_01;
    KCPAT_11 = (K_10 * P_01);
    P_00 = APAT_00 - KCPAT_00 + Sw_00;      P_01 = APAT_01 - KCPAT_01 +
Sw_01;
    P_10 = APAT_10 - KCPAT_10 + Sw_10;      P_11 = APAT_11 - KCPAT_11 +
Sw_11;
// Return the estimate.
    return x_00;
}
```

Parallel Programming Models

There are several parallel programming models in common use

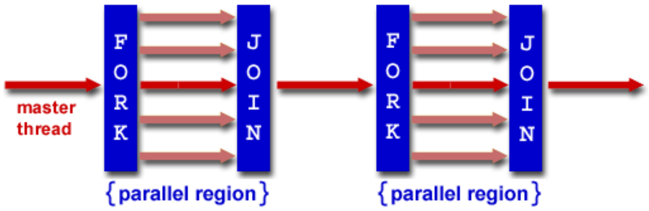
- Threads
- Shared Memory (without threads)
- Distributed Memory / Message Passing
- Data Parallel
- Hybrid
- Single Program Multiple Data (SPMD)
- Multiple Program Multiple Data (MPMD)

Threads (OpenMP), Single Program Multiple Data (SPMD) and Work-Sharing are applied in this paper.

OpenMP Programming Model

OpenMP runs on a shared memory architecture. OpenMP is based upon the existence of multiple threads in the shared memory programming paradigm. A shared memory process consists of multiple threads. OpenMP is an explicit (not optimically) programming model, offering the programmer full control over parallelization. OpenMP uses the fork-join model of parallel execution as shown in Figure (3). All OpenMP programs begin a single process: the master thread. The master executes sequentially until the first parallel region construct is encountered.

- **FORK**: the master thread then creates a team of parallel threads. The statements in the program that are enclosed by the parallel region construct are then executed in parallel among the various team threads.
- **JOIN**: When the team threads complete the statements in the parallel region construct, they synchronize and terminate, leaving only the master threads[7].



Figure(3) Fork-Join Model

Single Program Multiple Data (SPMD)

SPMD is actually a "high level" programming model that can be built upon any combination of the previously mentioned parallel programming models. All tasks execute their copy of the same program simultaneously. This program can be threads, message passing, data parallel or hybrid. All tasks may use different data. SPMD programs usually have the necessary logic programmed into them to allow different tasks to branch or conditionally execute only those parts of the program they are designed to execute as shown in figure. That is, tasks do not necessarily have to execute the entire program - perhaps only a portion of it[8]. Figure (4) shows a SPMD model.

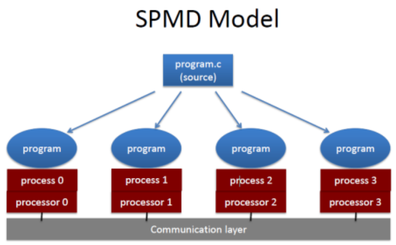
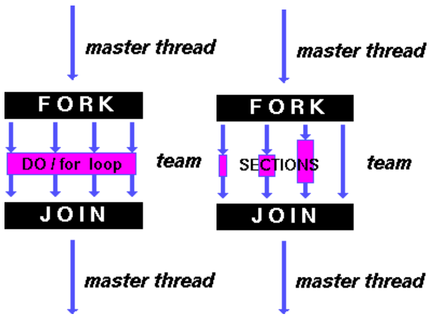


Figure (4) SPMD Model

Work-Sharing Constructs

A work sharing construct distribute the execution of the associated statement among the members of the team that encounter it. The work sharing directives do not launch new threads, and there is no implied barrier on entry to a work sharing construct. The sequence of work sharing constructs and **barrier** directives encountered must be the same for every thread in a team. OpenMP defines the following work sharing constructs, and these are described in the sections that follow: for directive, sections directive and single directive[8]. Process of Work Sharing Construct is shown in Figure (5).



Figure(5) Process of Work Sharing Construct

Code segment of Kalman filter for SPMD

The following directive `#pragma omp parallel` defines a parallel region , which is a region of the program that is to be executed by multiple threads in parallel. This is the fundamental construct that starts parallel execution.

```
int main(int argc, char **argv)
{
   int i;
  double wtime;
  omp_set_num_threads(NUM_THREADS);
  wtime = omp_get_wtime ( );
  #pragma omp parallel
  {
     int id=omp_get_thread_num();

     int nthreads=omp_get_num_threads();
     for (i = id; i < SAMPLE_COUNT;i=i+nthreads)

{
                   // Get the gyro and accelerometer input.
       gyro_input = sample_data[i][0];
       accel_input = sample_data[i][1];
       // Update the Kalman filter and get the output.
       kalman_output = kalman_update(gyro_input,
accel_input);
     }
   }
   wtime = omp_get_wtime ( ) - wtime
```

Code Segment of Kalman Filter for Work Sharing

OpenMP defines for directive to use work sharing construct. Combined parallel work sharing constructs are shortcuts for specifying a parallel region that contains only one work sharing construct. The parallel for directive is a shortcut for a parallel region that contains only a single for directive. The syntax of the **parallel for** directive is #pragma omp for.

```
int main(int argc, char **argv )
          {  int i;
             double wtime;
             omp_set_num_threads(NUM_THREADS);
             wtime = omp_get_wtime ( );
             #pragma omp parallel
             {
               int id=omp_get_thread_num();

               #pragma omp for
               for (i = 0; i < SAMPLE_COUNT;i++)
              {
          // Get the gyro and accelerometer input.
                      gyro_input = sample_data[i][0];
                      accel_input = sample_data[i][1];
            // Update the Kalman filter and get the output.
                      kalman_output = kalman_update(gyro_input, accel_input);
                 }
            }
            wtime = omp_get_wtime ( ) - wtime;
          }
```

Results and Discussion

By analyzing the computing data at Table (1) and Figure (6), the number of threads is increased and the executing time for SPMD and work sharing go dropped. The result using Work Sharing method and the SPMD (Simple Program Multiple Data) method are nearly the same. In Figure (7) and Table (2), the execution for OpenMP (parallel computing) is faster than the execution for serial computing. The parallel computing has better performance than serial computing. The parallel Kalman filter really needs to filter the noise quickly. Parallel computing has shorter time span than the serial computing. Parallel computing is suitable for handling the large amount of data.

Computation(1)

Table (1)The relation between number of threads and execution time for SPMD and work sharing

No. of Threads	Execution time for SPMD	Execution time for work sharing
2	0.000045	0.000046
4	0.000043	0.000045
6	0.000042	0.000043
8	0.000028	0.000029

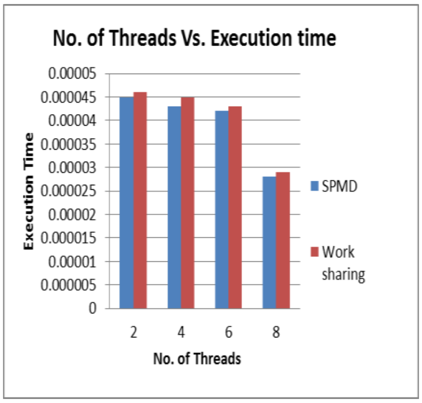


Figure (6) Dependence of execution time on a number of threads

Computation(2)

Table (2) Comparison between the execution time for Serial and OpenMP programs

Program	Execution time (s)
Serial	0.15
OpenMP	0.000054

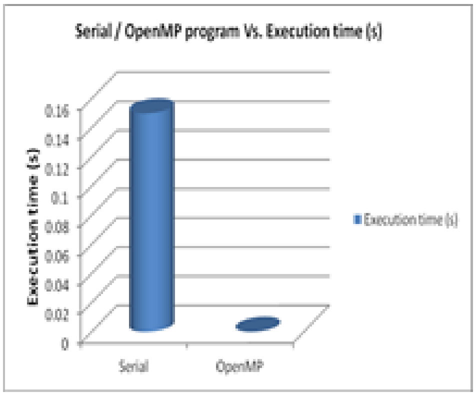
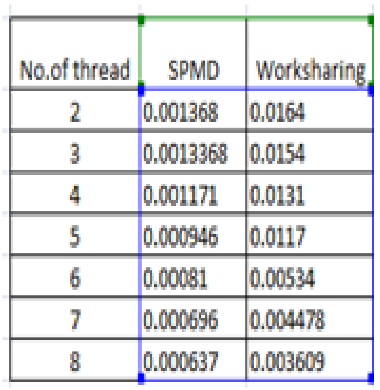


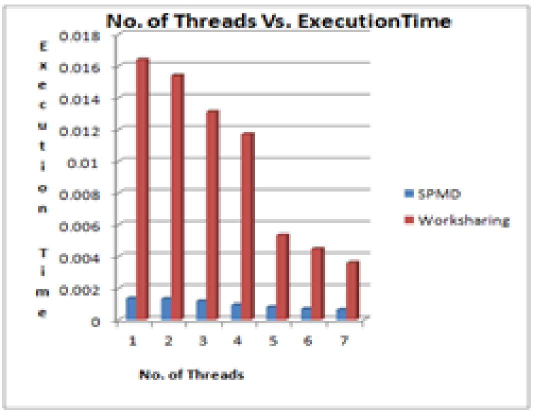
Figure (7) Comparison between execution time taken by Serial and OpenMP program

Computation(3)

Table (3) The relation between no. of threads and execution time for SPMDand work sharing for Dell sever.



No.of thread	SPMD	Worksharing
2	0.001368	0.0164
3	0.0013368	0.0154
4	0.001171	0.0131
5	0.000946	0.0117
6	0.00081	0.00534
7	0.000696	0.004478
8	0.000637	0.003609

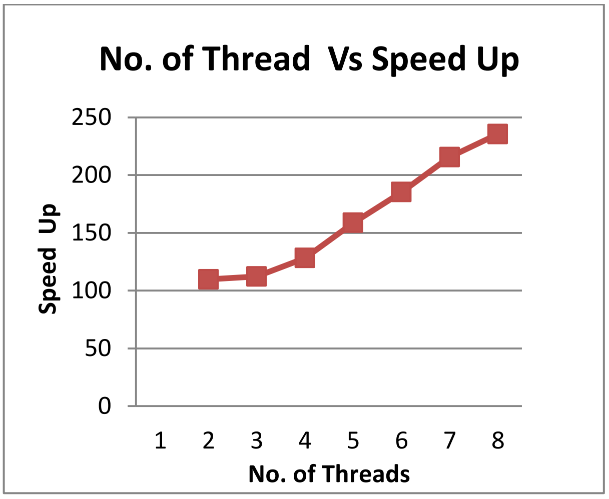


Figure(8)The relation of No. of Thread and Execution time

Computation(4)

Table(4) The relation data of number of thread and speed up

No. of Threads	Speed Up
1	2777.8
2	109.65
3	112.21
4	128.09
5	158.56
6	185.18
7	215.51
8	235.47



Figure(9) The relation graph of number of CPUs and SpeedUp

The data of Table (3) and the graph of Figure (8) describe the relation between number of threads and execution time for SPMD and Work sharing

Dell® PowerEdge 2900 server computer, 8G RAM. The result using SPMD method is better performance than Work Sharing method when using Dell® PowerEdge 2900 server computer, 8G RAM. The data of Table (4) and the graph of Figure (9) describe the relation between number of threads and execution time for SPMD. The computation (1),(2)and (3) are used same program and same data but the computer specification is different. The computation (1) is used Core i7 computer and 2G memory. The computation (2), (3) and (4) is used Dell® PowerEdge 2900 server computer, 8G RAM. When analyzing the computation (1),(2), (3) and (4) the execution time depends on specification of computer such as number of processors, memory capacity, and hard disk capacity . For parallel computing, the execution time depends on amount of data. If the amount of data is small amount, serial computing is faster than parallel computing. If the amount of data is extremely large, the parallel computing is better performance than serial computing.

Conclusion

Efficient parallelization of the Kalman filter has been carried out on a shared-memory multi-core architecture. The parallelization is achieved by re-ordering the Kalman filter (KF) equations so that the data dependencies are broken and allowed for a well parallelized program implementation. The result exhibits linear speed-up in number of cores. The increased speed of parallel processing holds special advantages for real-time systems. A parallel system increases reliability through simple redundancy. In many cases, these new changes will require more computing power. When the capabilities of a single processor are exceeded, the entire system must be replaced. This often requires major changes to the software. With a parallel system, increased capability can be added with additional processors. Parallel systems can be constructed from relatively cheap, mass-produced processors. The relatively slow step times in these processors minimizes heat dissipation and transmission delay problems. In general, parallel systems can claim a price/performance advantage over traditional systems. OpenMP is a widely accepted programming model for shared memory systems. Kalman filter is a software tool and it is best filtering and applied in many fields. The implementation of Kalman filter using MPI will be researched in future.

Acknowledgement

The author is deeply indebted to Dr Pho Kaung, Pro-rector/ Head, Universities' Research Center, University of Yangon for all his valuable guidance.

References

Kalman. R. e. (1960). "a New Approach to Linear Filtering and Prediction Problems," Transcation of the SME- Journal of Basic Engineering , pp. 35-45(March 1960).

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

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

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

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

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

www.openmp.org/wp/about-openmp.

www.computing.llnl.gov/ introduction to parallel computing.htm