**UNIVERSITAS INDONESIA**

# KAJIAN KOMPUTASI PARALEL UNTUK MODEL PERSAMAAN DIFERENSIAL STOKASTIK PADA MESIN MULTICORE DENGAN MATLAB

**SKRIPSI**

**FERDY JAMANTA**
**0706261650**

**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**
**PROGRAM STUDI MATEMATIKA**
**DEPOK**
**JULI 2011**

**UNIVERSITAS INDONESIA**

# KAJIAN KOMPUTASI PARALEL UNTUK MODEL PERSAMAAN DIFERENSIAL STOKASTIK PADA MESIN MULTICORE DENGAN MATLAB

## SKRIPSI

**Diajukan sebagai salah satu syarat untuk memperoleh gelar sarjana sains**


**FERDY JAMANTA**
**0706261650**



**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**
**PROGRAM STUDI MATEMATIKA**
**DEPOK**

**JULI 2011**

## HALAMAN PERNYATAAN ORISINALITAS

Skripsi ini adalah hasil karya saya sendiri, dan semua sumber baik yang dikutip maupun dirujuk telah saya nyatakan dengan benar.

| | |
|---|---|
| Nama | : Ferdy Jamanta |
| NPM | : 0706261650 |
| Tanda Tangan | :  |
| Tanggal | : 7 Juli 2011 |

## HALAMAN PENGESAHAN

Skripsi ini diajukan oleh :
Nama : Ferdy Jamanta
NPM : 0706261650
Program Studi : Matematika
Judul Skripsi : Kajian Komputasi Paralel untuk Model Persamaan Diferensial Stokastik pada Mesin *Multicore* dengan MATLAB

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Sains pada Program Studi Matematika, Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Indonesia.

DEWAN PENGUJI

Pembimbing : Gatot F. Hertono, PhD. ( )

Penguji I : Dr. Alhadi Bustamam, M.Kom. ( )

Penguji II : Dr. rer. nat. Hendri Murfi, M.Kom. ( )

Penguji III : Bevina D. Handari, PhD. ( )

Ditetapkan di : Depok
Tanggal : 9 Juni 2011

# KATA PENGANTAR

Alhamdulillah, puji syukur penulis panjatkan kepada Allah SWT, karena atas berkat dan rahmat-Nya, penulis dapat menyelesaikan skripsi ini. Penulisan skripsi ini dilakukan dalam rangka memenuhi salah satu syarat untuk mencapai gelar Sarjana Sains Jurusan Matematika pada Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Indonesia. Penulis menyadari bahwa, tanpa bantuan dan bimbingan dari berbagai pihak, dari masa perkuliahan sampai pada penyusunan skripsi ini, sangatlah sulit bagi penulis untuk menyelesaikan skripsi ini. Oleh karena itu, penulis mengucapkan terima kasih kepada:

(1) Gatot F. Hertono, PhD, selaku pembimbing yang telah banyak meluangkan waktu dan pikiran serta memberikan masukan-masukan untuk penulis dalam menyelesaikan tugas akhir ini.
(2) Dra. Yahma Wisnani, M.Kom selaku pembimbing akademik penulis selama menjalani masa kuliah.
(3) Dra. Denny Riama Silaban, M.Kom, Dr. Kiki Ariyanti S, dan Bevina D. Handari, PhD, Dr. Sri Mardiyati M.Kom, Dr. Alhadi Bustamam, M.Kom, dan Dr. rer. nat. Hendri Murfi, M.Kom yang telah hadir dan memberikan saran serta masukan bagi penulis pada SIG 1, SIG 2, Kolokium.
(4) Seluruh staf pengajar di Matematika UI atas ilmu pengetahuan yang telah diberikan.
(5) Seluruh karyawan di Departemen Matematika UI atas bantuan yang telah diberikan.
(6) Orang tua, kakak, adik-adik, dan Widiyani Suciati S.Si yang selalu memberikan doa, semangat, dan dukungan bagi penulis.
(7) Bowo yang telah meluangkan waktu untuk membantu penulis menjalankan program dalam skripsi ini.
(8) Syahrul Syawal dan Riski Defri Heriyanto yang telah berjuang bersama selama kuliah hingga penyusunan skripsi ini.

(9) Nora, Lois, Winda, Dita, Farah dan Toto, terima kasih atas semangat dan dukungannya

(10) Seluruh teman-teman angkatan 2007 yang telah memberikan pengalaman per-kuliahan yang tak terlupakan.

(11) Kepada semua teman-teman di Matematika UI angkatan 2006, 2008, 2009, dan 2010 yang tidak bisa disebutkan satu per satu. Terima kasih atas dukungannya.

Penulis juga ingin mengucapkan terima kasih kepada seluruh pihak yang tidak dapat disebutkan satu per satu, yang telah membantu dalam penyusunan skripsi ini. Akhir kata, penulis mohon maaf jika terdapat kesalahan atau kekurang-an dalam skripsi ini. Penulis berharap semoga skripsi ini bermanfaat bagi pengembangan ilmu.

Penulis

2011

**HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS**

---

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini:

| | |
|---|---|
| Nama | : Ferdy Jamanta |
| NPM | : 0706261650 |
| Program Studi | : Sarjana Matematika |
| Departemen | : Matematika |
| Fakultas | : Matematika dan Ilmu Pengetahuan Alam |
| Jenis karya | : Skripsi |

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia Hak Bebas Royalti Noneksklusif (*Non-exclusive Royalty Free Right*) atas karya ilmiah saya yang berjudul:
Kajian Komputasi Paralel untuk Model Persamaan Diferensial Stokastik pada Mesin *Multicore* dengan MATLAB

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/format-kan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Depok
Pada tanggal : 7 Juli 2011
Yang menyatakan

( Ferdy Jamanta )

# ABSTRAK

Nama : Ferdy Jamanta
Program Studi : Matematika
Judul : Kajian Komputasi Paralel untuk Model Persamaan Diferensial Stokastik pada Mesin *Multicore* dengan MATLAB

Model-model Persamaan Diferensial Stokastik (PDS) memiliki peranan yang sangat penting di berbagai bidang industri, misalnya ekonomi, keuangan, biologi, kimia, epidemiologi, juga mikroelektronik (Higham D. J., 2001). Metode numerik seringkali digunakan untuk mengaproksimasi solusi dari suatu model PDS, sehingga dibutuhkan suatu proses komputasi untuk memperoleh solusi dari suatu model PDS tersebut. Model-model PDS biasanya melibatkan data dalam jumlah besar ataupun proses komputasi yang banyak, sehingga berdampak pada waktu komputasi yang semakin lama. Untuk mempercepat waktu komputasi, maka diterapkan komputasi paralel. Komputasi paralel adalah salah satu teknik melakukan komputasi secara bersamaan dengan memanfaatkan beberapa komputer/prosesor pada suatu waktu tertentu. Dalam skripsi ini diberikan algoritma paralel untuk mengaproksimasi solusi dari suatu model PDS. Algoritma-algoritma ini diimplementasikan dalam program yang dijalankan pada mesin *multicore* dengan MATLAB dan *Parallel Computing Toolbox* (versi *trial*). Diberikan juga kinerja algoritma paralel yang diukur dengan *speed up* dan efisiensi paralel.

Kata Kunci : persamaan diferensial stokastik, komputasi paralel, mesin *multicore*, MATLAB.
xiii+88 halaman ; 25 gambar; 19 tabel; 4 lampiran
Daftar Pustaka : 12 (1998-2010)

# ABSTRACT

Name : Ferdy Jamanta
Program Study : Mathematics
Title : Study of Parallel Computing for Stochastic Differential Equation Models on Multicore Machine with MATLAB

Stochastic Differential Equations (SDEs) models play a prominent role in a range of application areas, including biology, chemistry, epidemiology, mechanics, microelectronics, economics, and finance (Higham D. J., 2001). Numerical method is usually used to get an approximate solution of SDEs models which often involve huge data or many computation steps, hence need more computation time. Parallel computing is an alternative that can reduce the computation time. This *skripsi* discuss some parallel techniques to solve SDEs problems especially in finance models. The parallel techniques is designed to utilize several processors simultaneously. In this case the algorithms run on multicore machine with MATLAB and Parallel Computing Toolbox (trial version). Parallel perfomance of the algorithms are also given which compared the speed up and efficiency of several parallel techniques.

Key Words : stochastic differential equation, parallel computing, multicore machine, MATLAB.
xiii+88 pages ; 25 pictures; 19 tables ; 4 attachments
Bibliography : 12 (1998-2010)

## DAFTAR ISI

# DAFTAR TABEL

## DAFTAR GAMBAR

## DAFTAR LAMPIRAN

# BAB 1
# PENDAHULUAN

## 1.1 Latar Belakang

Model-model Persamaan Diferensial Stokastik (PDS) memiliki peranan yang sangat penting di berbagai bidang industri, misalnya ekonomi, keuangan, biologi, kimia, epidemiologi, juga mikroelektronik (Higham D. J., 2001). Solusi dari suatu model PDS dapat berupa solusi eksplisit ataupun implisit. Metode numerik seringkali digunakan untuk mengaproksimasi solusi dari suatu model PDS, sehingga dibutuhkan suatu proses komputasi untuk memperoleh solusi dari suatu model PDS.

Pada bidang tertentu, umumnya model-model PDS melibatkan data dalam jumlah besar. Jika hanya satu komputer (prosesor) yang digunakan untuk memperoleh solusi dari suatu model PDS, maka akan memakan waktu yang lama, karena satu prosesor hanya dapat melakukan komputasi sekuensial. Oleh karena itu, diperlukan komputasi paralel untuk mempercepat proses komputasi dalam mendapatkan solusi dari suatu model PDS tersebut.

Komputasi paralel adalah salah satu teknik melakukan komputasi secara bersamaan dengan memanfaatkan beberapa komputer/prosesor pada suatu waktu tertentu. Komputasi paralel umumnya memiliki efektifitas saat kapasitas data yang diproses sangat besar, baik karena harus mengolah data dalam jumlah besar ataupun karena tuntutan proses komputasi yang banyak. Hal ini berdampak pada lamanya waktu komputasi yang diperlukan. Dengan komputasi paralel diharapkan waktu komputasi dapat lebih cepat.

Klasifikasi komputasi paralel ada berbagai macam, misalnya *multicore computing, symmetric multiprocessing, distributed computing, cluster computing, massive parralel processing, grid computing* (Barney, 2010). *Multicore computing* adalah komputasi paralel dengan menggunakan satu mesin yang memiliki dua atau lebih prosesor (mesin *multicore*) yang saat ini sudah banyak diterapkan pada jenis prosesor misalnya Intel Core Duo, AMD Phenom II X2,

Intel Core i5, AMD Phenom II X4, Intel Core i7 Extreme Edition 980X, ataupun AMD Phenom II X6. MATLAB versi R2008a dengan *Parallel Computing Toolbox* atau yang lebih baru dapat menjadi salah satu perangkat lunak yang digunakan untuk *multicore computing* (Burkardt, 2009). Pada skripsi ini akan digunakan MATLAB versi R2009a dengan *Parallel Computing Toolbox* (versi *trial*) untuk melakukan simulasi suatu model PDS secara paralel.

Tujuan utama dari skripsi ini adalah menerapkan algoritma paralel pada suatu model PDS sehingga diharapkan adanya peningkatan kecepatan (*speed up*) proses komputasi. Oleh karena itu, pada skripsi ini akan dibangun algoritma paralel yang efisien dan memiliki *speed-up* yang baik untuk menyelesaikan suatu model PDS pada mesin *multicore*. Selain itu, pada skripsi ini juga akan dikaji *speed up* dan efisiensi komputasi paralel dalam menyelesaikan suatu model PDS.

## 1.2 Permasalahan dan Ruang Lingkup Penelitian

Bagaimanakah membangun algoritma paralel yang efisien dan memiliki *speed-up* yang baik dalam menyelesaikan suatu model PDS pada mesin *multicore*?

## 1.3 Pembatasan Masalah

Pada skripsi ini, mesin *multicore* yang digunakan memiliki maksimum 4 prosesor dan metode numerik yang digunakan untuk mengaproksimasi solusi PDS adalah metode Euler-Maruyama.

## 1.4 Jenis Penelitian dan Metode Penelitian

Jenis penelitian yang digunakan adalah studi literatur, dan metode yang digunakan adalah pengembangan algoritma dan simulasi.

## 1.5 Tujuan Penelitian

Tujuan dari penulisan skripsi ini adalah membangun algoritma paralel yang efisien dan memiliki *speed-up* yang baik untuk menyelesaikan suatu model PDS pada mesin *multicore*. Pada skripsi ini juga diberikan algoritma paralel untuk model PDS pada bidang keuangan. Algoritma-algoritma ini akan disimulasikan pada MATLAB versi R2009a dengan *Parallel Computing Toolbox* (versi *trial*). Selain itu, pada skripsi ini juga akan dikaji *speed-up* dan efisiensi yang bertujuan untuk mengukur kinerja dari algoritma paralel pada suatu model PDS.

# BAB 2
# LANDASAN TEORI

Pada bab ini akan dibahas teori dasar mengenai komputasi paralel dan Persamaan Diferensial Stokastik (PDS) yang diperlukan pada pembahasan bab-bab berikutnya, diantaranya komputasi paralel, klasifikasi komputasi paralel, arsitektur memori pada pemrograman paralel, model-model pemrograman paralel, dan kinerja komputasi paralel. Berikutnya juga akan dibahas mengenai Persamaan Diferensial Stokastik (PDS), proses *Wiener*, dan skema Euler-Maruyama. Pada akhir bab ini, akan dibahas mengenai Sistem PDS berdimensi $N$ dengan proses *Wiener* berdimensi $M$.

## 2.1 Komputasi Paralel

Seiring dengan perkembangan teknologi yang semakin pesat, perkembangan industri juga berkembang semakin pesat. Industri yang melibatkan pengolahan data dalam jumlah besar biasanya menggunakan komputer untuk mengolah data tersebut. Sebagai contoh, pada industri yang bergerak di bidang prakiraan cuaca terdapat masalah yang melibatkan pengolahan data dalam jumlah besar (Wilkinson & Allen, 1999). Jika hanya menggunakan satu komputer untuk mengolah data pada masalah prakiraan cuaca, maka diperlukan (Fouque, Papanicolaou, & Sircar, 2000) waktu lebih dari 100 hari (Wilkinson & Allen, 1999). Pada industri lain seperti pada bidang keuangan, untuk menghitung 100 harga opsi *call* Asia dengan melakukan simulasi sebanyak $10^6$ *sample path* dengan menggunakan satu prosesor, maka dibutuhkan waktu sedikitnya 7 jam (Kanniainen & Piche, 2009). Penggunaan satu komputer yang hanya dapat melakukan komputasi sekuensial untuk menyelesaikan masalah tersebut sudah tidak relevan lagi. Komputasi sekuensial akan mengalami keterbatasan dalam hal kecepatan pengolahan data. Hal inilah yang mendasari munculnya komputasi paralel yang bertujuan untuk mempercepat proses komputasi.

Komputasi paralel adalah salah satu teknik melakukan komputasi secara bersamaan dengan memanfaatkan beberapa komputer/prosesor pada suatu waktu tertentu. Komputasi paralel umumnya memiliki efektifitas saat kapasitas data yang diproses sangat besar, baik karena harus mengolah data dalam jumlah besar ataupun karena tuntutan proses komputasi yang banyak. Hal ini berdampak pada lamanya waktu komputasi yang diperlukan. Dengan komputasi paralel diharapkan waktu komputasi dapat menjadi lebih cepat.

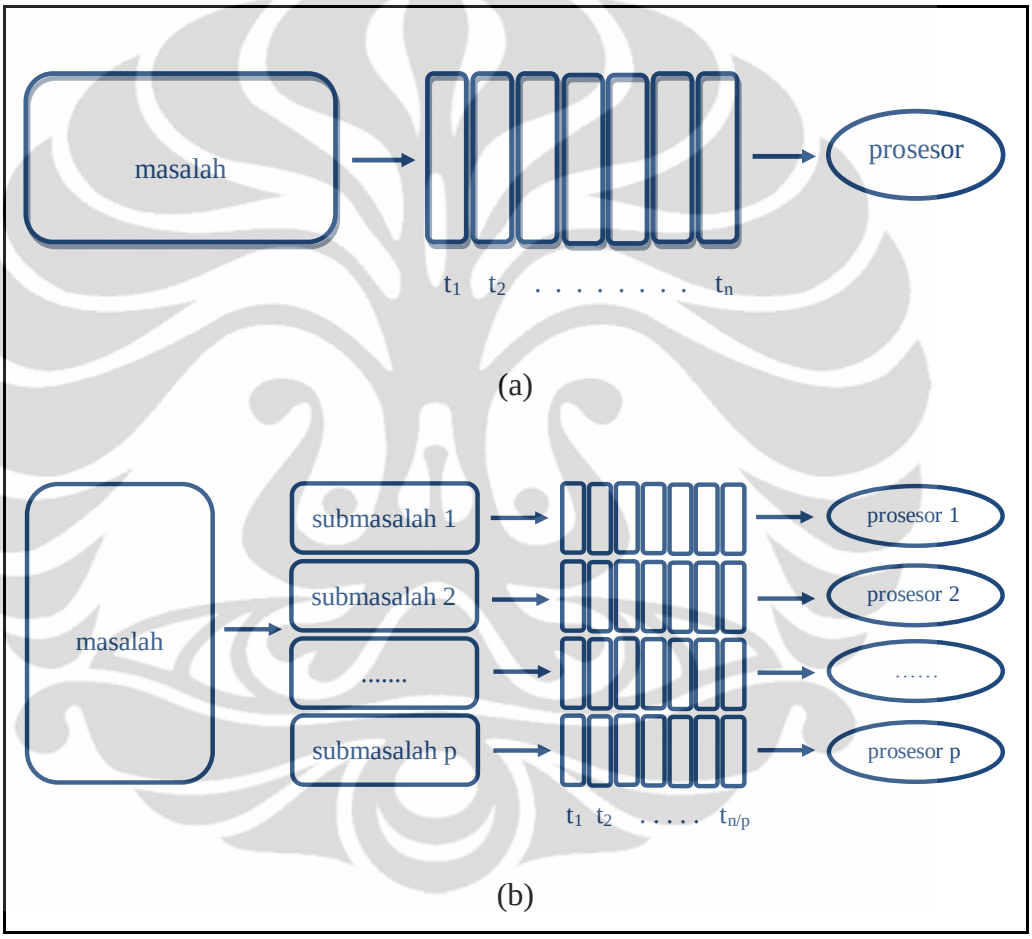
Pada komputasi paralel, biasanya suatu masalah yang besar dipartisi menjadi submasalah-submasalah yang lebih kecil dimana masing-masing dari submasalah tersebut dapat dikerjakan secara bersamaan (Barney, 2010). Setiap prosesor yang tersedia ditugaskan untuk menyelesaikan masing-masing submasalah tersebut. Untuk lebih jelas mengenai perbedaan antara komputasi sekuensial dan komputasi paralel, perhatikan Tabel 2.1.

Tabel 2.1 Perbedaan komputasi sekuensial dan komputasi paralel

| No | Komputasi Sekuensial | Komputasi Paralel |
|---|---|---|
| 1 | Program berjalan pada satu prosesor | Program berjalan pada dua atau lebih prosesor |
| 2 | Program diproses secara sekuensial | Program didekomposisi menjadi beberapa subprogram yang dapat diproses secara paralel |
| 3 | Hanya satu instruksi yang bisa di eksekusi pada suatu waktu tertentu | Masing-masing subprogram di eksekusi secara bersamaan pada prosesor yang tersedia |
| | | Setiap prosesor memiliki subprogram yang sama atau berbeda dan data yang diproses sama atau berbeda. |

Sebagai contoh, misalkan suatu masalah dapat dikerjakan secara sekuensial dengan satu prosesor memakan waktu $t_{sekuensial} = t_1 + t_2 + \ldots + t_n$ dimana $n > 0$ dan $n$ bilangan bulat (lihat Gambar 2.1a). Misalkan pula masalah tersebut dapat didekomposisi menjadi $p$ submasalah yang saling bebas dan $p$

prosesor ditugaskan untuk menyelesaikan setiap submasalah-submasalah. Jika setiap submasalah dikerjakan secara bersamaan, maka masalah tersebut akan selesai dalam waktu $t_{paralel} = t_1 + t_2 + \ldots + t_{n/p}$ dimana $n > 0$, $p > 0$, $n \geq p$ dan $n,p$ bilangan bulat (lihat Gambar 2.1b). Dengan demikian diharapkan jika masalah tersebut dikerjakan secara paralel, maka waktu komputasi akan menjadi lebih cepat.




Gambar 2.1 Contoh visualisasi komputasi sekuensial dan komputasi paralel

Dalam melakukan komputasi paralel, tentunya dibutuhkan suatu algoritma paralel. Untuk membangun algoritma paralel yang baik dan efisien, ada beberapa hal yang harus diperhatikan. Hal tersebut akan dibahas pada subbab berikutnya. Oleh karena itu, pada subbab berikutnya akan dibahas teori dasar mengenai algoritma paralel yang terdiri dari arsitektur memori dan model-model pemrograman paralel, serta kinerja algoritma paralel.

## 2.2 Algoritma Paralel

Membangun algoritma paralel lebih kompleks dibandingkan membangun algoritma sekuensial. Ada beberapa hal yang harus diperhatikan dalam membangun algoritma paralel (Barney, 2010), yaitu:

1. *Data dependence*

   Dalam suatu algoritma, suatu proses komputasi adakalanya bergantung pada proses komputasi sebelumnya. Misalkan pada skema Euler-Maruyama, yaitu:

   $$X(t_j) = X(t_{j-1}) + f\left(X(t_{j-1})\right)\Delta t + g\left(X(t_{j-1})\right)dW(t_j)$$

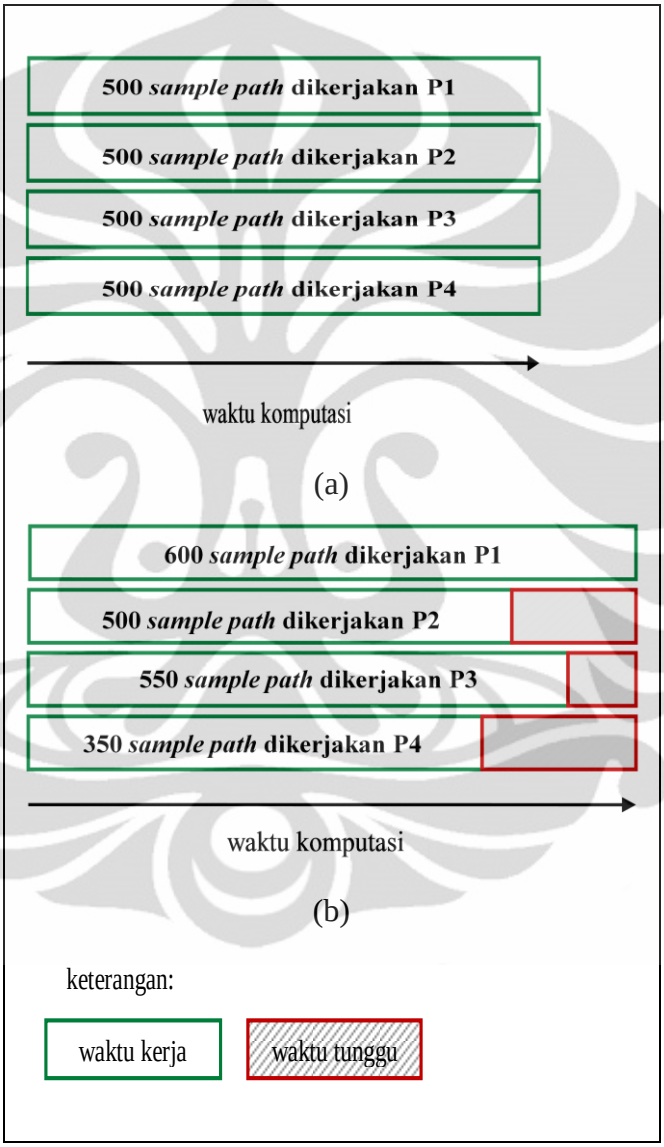
   dimana $j = 1,2,\dots,N$. Untuk menghitung $X(t_1)$ diperlukan nilai $X(t_0)$, untuk menghitung $X(t_2)$ diperlukan nilai $X(t_1)$, dan seterusnya hingga untuk menghitung $X(t_N)$ diperlukan nilai $X(t_{N-1})$. Ini adalah contoh dari *data dependence*, yaitu kebergantungan proses penghitungan data pada suatu tahap dengan proses penghitungan data pada tahap sebelumnya. Pada kasus *data dependence,* algoritma paralel yang bekerja dengan cara dekomposisi data tidak efektif untuk diterapkan karena proses penghitungan data pada tiap tahap tidak dapat dikerjakan secara bersamaan.

2. *Load balance*

   Pada algoritma paralel, pembagian tugas ke setiap prosesor yang tersedia sebaiknya sama besar. Hal ini bertujuan agar setiap prosesor dapat bekerja secara bersamaan dan selesai secara bersamaan pula. *Load balance* adalah pembagian tugas ke masing-masing prosesor sedemikian sehingga tugas yang diberikan ke masing-masing prosesor sama besar. Sebagai contoh, misalkan terdapat masalah untuk menyelesaikan 2000 *sample path*. Gambar 2.2a adalah contoh *load balance* yang baik dimana pembagian tugas ke masing-masing prosesor sama besar, yaitu setiap prosesor mengerjakan 500 *sample path*. dan Gambar 2.2b adalah contoh *load balance* yang tidak berimbang dimana pembagian tugas ke masing-masing prosesor tidak sama besar. *Load balance* yang baik akan mempercepat waktu komputasi.

3. *Concurrency*

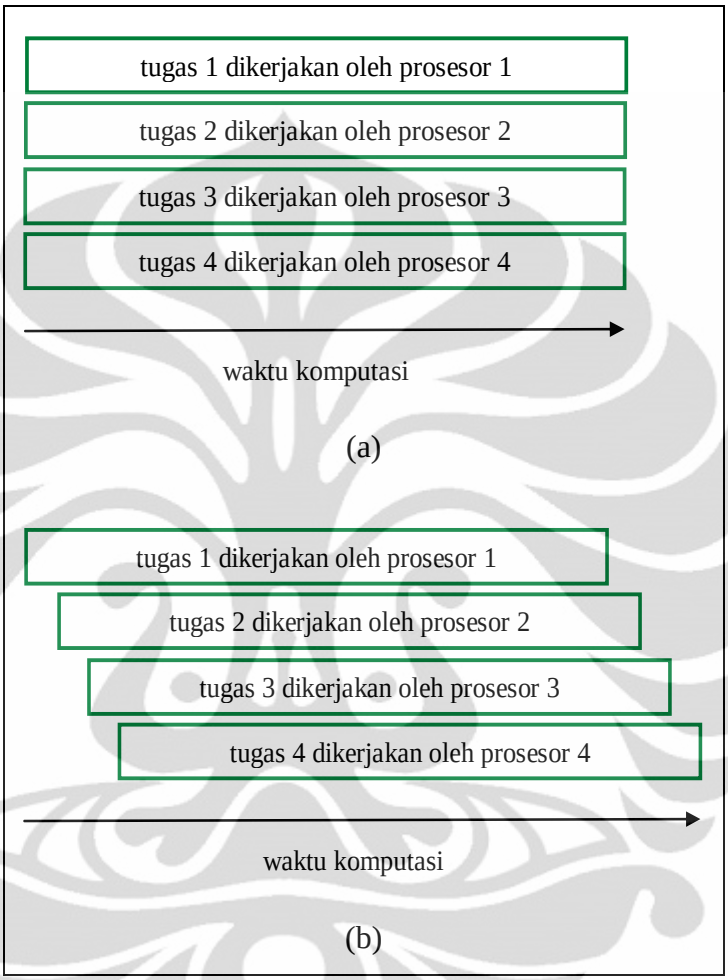
Hal yang harus diperhatikan dalam membangun algoritma paralel adalah apakah tugas-tugas yang diberikan ke masing-masing prosesor yang tersedia dapat dieksekusi secara bersamaan atau tidak. Hal ini dikenal dengan istilah *concurrency*.




Gambar 2.2 Contoh *load balance*

Misalkan tugas 2 yang diberikan ke prosesor 2 mulai dieksekusi setelah beberapa bagian dari tugas 1 telah dieksekusi (lihat Gambar 2.3b). Pada kasus ini, tentunya tugas-tugas pada prosesor 1 dan 2 tidak dapat dieksekusi pada saat yang sama. Hal ini sebaiknya dihindari dalam

menbangun algoritma paralel. Gambar 2.3a adalah contoh *concurrency* yang baik karena masing-masing tugas pada tiap prosesor dapat dieksekusi secara bersamaan.

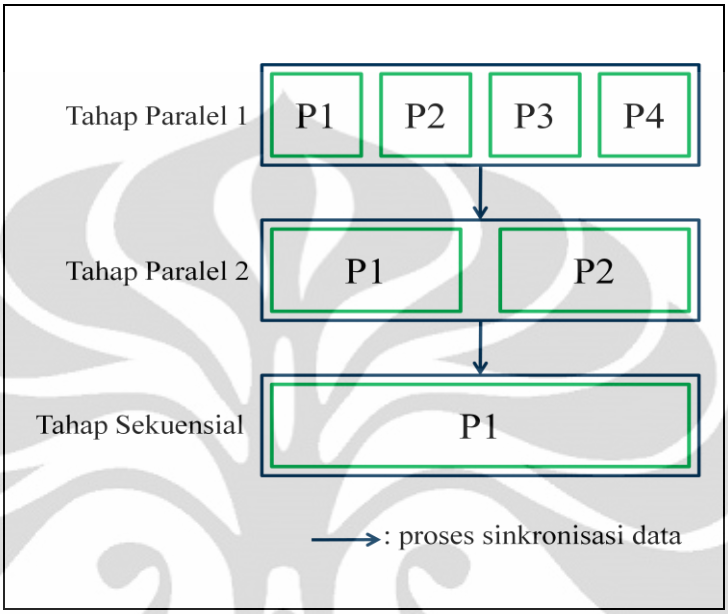



Gambar 2.3 Contoh *concurrency*

4. Sinkronisasi data

Pada algoritma paralel, biasanya terdiri dari beberapa tahap paralel dan tahap sekuensial. masing-masing prosesor mengeksekusi tugas-tugas yang berbeda. Untuk mendapatkan output yang diharapkan, antar tahap yang satu dengan yang lainnya membutuhkan proses sinkronisasi data, misalnya proses penjumlahan matriks, penggabungan elemen-elemen matriks yang diperoleh pada tahap sebelumnya, dan sebagainya. Sinkronisasi data adalah proses penyelarasan data antar prosesor pada suatu tahap paralel dengan tahap paralel lainnya atau dengan tahap sekuensial (lihat Gambar

2.4). Dalam membangun algoritma paralel yang baik, seorang *programmer* sebaiknya memperhatikan waktu komputasi yang diperlukan untuk proses sinkronisasi data.

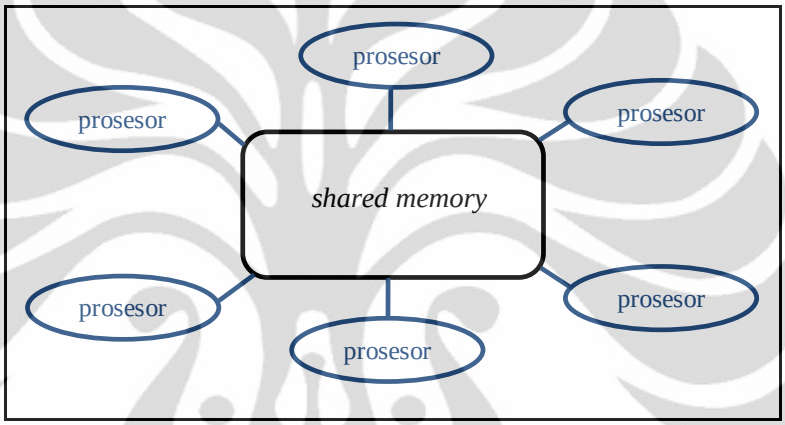



Gambar 2.4 Contoh proses sinkronisasi data

Komputasi paralel melibatkan dua atau lebih prosesor. Rancangan arsitektur prosesor paralel dapat dilihat dari rancangan arsitektur memorinya. Terdapat dua jenis arsitektur memori yang umum digunakan pada pemrograman paralel, yaitu *shared memory* dan *distributed memory* (Wilkinson & Allen, 1999). Namun pada skripsi ini hanya digunakan *shared memory*.

Pada arsitektur *shared memory* memungkinkan setiap prosesor dapat bekerja secara independen dan mengakses semua data yang tersimpan dalam *shared memory* (lihat Gambar 2.5). Namun pada suatu waktu tertentu, hanya satu prosesor yang dapat mengakses suatu lokasi pada *shared memory*. Sinkronisasi data dilakukan melalui pembacaan dan penulisan langsung ke *shared memory*. Kelebihan dari arsitektur *shared memory* adalah mudah digunakan karena semua data yang terlibat dalam pemrograman paralel tersimpan pada *shared memory*. Dengan demikian antar prosesor dapat berkomunikasi lebih cepat (Barney, 2010). Kekurangan dari arsitektur *shared memory* adalah keterbatasan kemampuan memori apabila jumlah prosesor bertambah. Ketika bertambahnya jumlah

prosesor, maka akan bertambah banyak pula *path* yang menghubungkan antara prosesor dengan *shared memory* (Barney, 2010). Dengan demikian akan terjadi akses yang tinggi pada *shared memory* sehingga berakibat pada semakin lamanya waktu komputasi. Selain itu, apabila ada perubahan isi pada suatu lokasi *shared memory* oleh satu prosesor, maka akan berdampak pada prosesor lainnya yang juga menggunakan lokasi *shared memory* tersebut. Oleh karena itu, seorang *programmer* harus cermat dalam mengakses dan memodifikasi isi suatu lokasi *shared memory*.




Gambar 2.5 *Shared memory*

### 2.2.1 Model-model Pemrograman Paralel

Dalam pemrograman paralel, pengolahan data yang besar dapat dilakukan dengan berbagai cara. Pada 1966, Michael J. Flynn menciptakan suatu sistem klasifikasi untuk komputasi paralel, yang dikenal dengan sebutan taksonomi Flynn (Barney, 2010). Flynn mengelompokkan komputasi paralel berdasarkan banyaknya instruksi yang dieksekusi dan banyaknya data yang digunakan oleh instruksi tersebut (Barney, 2010). Berikut ini adalah taksonomi Flynn:

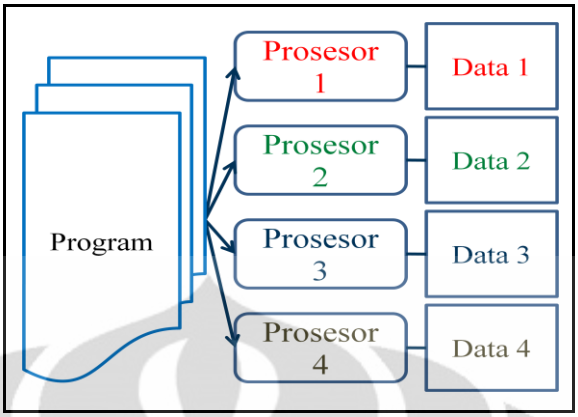
1. *Single Instruction Single Data* (SISD)
   Dalam taksonomi ini hanya digunakan satu prosesor. Setiap instruksi hanya dapat mengerjakan satu input data pada suatu waktu tertentu.

Taksonomi ini adalah yang paling sederhana dan diterapkan pada komputasi sekuensial.

2. *Single Instruction Multiple Data* (SIMD)
   Dalam taksonomi ini dapat digunakan dua atau lebih prosesor. Pada suatu waktu tertentu, setiap prosesor memiliki instruksi yang sama, namun input data yang digunakan pada tiap prosesor berbeda. Taksonomi ini diterapkan pada komputasi paralel.
3. *Multiple Instruction Single Data* (MISD)
   Dalam taksonomi ini dapat digunakan dua atau lebih prosesor. Pada suatu waktu tertentu, setiap prosesor memiliki input data yang sama, namun setiap prosesor memiliki instruksi yang berbeda terhadap input data tersebut. Taksonomi ini diterapkan pada komputasi paralel.
4. *Multiple Instruction Multiple Data* (MIMD)
   Dalam taksonomi ini dapat digunakan dua atau lebih prosesor. Pada suatu waktu tertentu, setiap prosesor memiliki input data yang berbeda dan juga memiliki instruksi yang berbeda terhadap input data tersebut. Taksonomi ini diterapkan pada komputasi paralel.

Pada skripsi ini akan digunakan tipe SIMD pada proses paralel. Implementasi SIMD pada pemrograman paralel dikenal dengan *Single Program Multiple Data* (SPMD). SPMD adalah salah satu model pemrograman paralel dimana setiap prosesor mengerjakan program yang sama tetapi data yang diproses berbeda (Wilkinson & Allen, 1999) (lihat Gambar 2.6).

MATLAB versi 2008a atau yang lebih baru dengan *Parallel Computing Toolbox* telah dapat melakukan SPMD dengan menggunakan perintah *spmd*. Pada *spmd*, setiap prosesor dapat menyimpan variabel dengan data yang berbeda pada masing-masing prosesor (Luszczek, 2009). Untuk kebutuhan sinkronisasi data, suatu prosesor membutuhkan komunikasi dengan prosesor yang lainnya dengan *message passing* antar prosesor dengan menggunakan perintah *labSend* dan *labReceive* (Luszczek, 2009).




Gambar 2.6 Contoh *Single Program Multiple Data* (SPMD)

Berdasarkan perangkat keras yang mendukung komputasi paralel terdapat sistem klasifikasi lain untuk komputasi paralel, yaitu *multicore computing, symmetric multiprocessing, distributed computing, cluster computing, massive parralel processing, grid computing* (Barney, 2010). *Multicore computing* adalah komputasi paralel dengan menggunakan satu mesin yang memiliki dua atau lebih prosesor (mesin *multicore*) (Barney, 2010). Pada skripsi ini hanya menggunakan tipe *multicore computing*. Untuk melakukan *multicore computing* digunakan mesin *multicore* yang saat ini sudah banyak diterapkan pada jenis prosesor misalnya Intel Core Duo, AMD Phenom II X2, Intel Core i5, AMD Phenom II X4, Intel Core i7 Extreme Edition 980X, ataupun AMD Phenom II X6.

### 2.2.2 Kinerja Algoritma Paralel

Kinerja suatu algoritma paralel biasa diukur dengan menggunakan parameter *speed up* dan efisiensi paralel. Berikut ini akan dibahas dua parameter kinerja paralel tersebut:

1. *Speed up*

   Secara umum *speed up* didefinisikan sebagai

$$S(n) = \frac{waktu\ komputasi\ menggunakan\ satu\ prosesor}{waktu\ komputasi\ menggunakan\ n\ prosesor} = \frac{t_s}{t_p}$$

(Wilkinson & Allen, 1999). Nilai $S(n)$ meningkat sebanding dengan banyaknya prosesor yang digunakan. Kinerja paralel yang baik apabila nilai $S(n)$ mendekati $n$ (Wilkinson & Allen, 1999).

2. Efisiensi

Secara umum efisiensi didefinisikan sebagai

$$E(n) = \frac{speed\ up}{banyaknya\ prosesor} = \frac{S(n)}{n}$$

(Wilkinson & Allen, 1999). Nilai $E(n)$ biasanya terletak antara 0 dan 1 atau dengan kata lain $0 \leq E(n) \leq 1$. Kinerja paralel yang baik apabila nilai $E(n)$ mendekati 1 (Wilkinson & Allen, 1999).

Pada subbab selanjutnya akan dibahas mengenai teori dasar mengenai Persamaan Diferensial Stokastik (PDS) diantaranya adalah proses *Wiener*, metode Euler-Maruyama, dan Sistem PDS berdimensi *N* dengan proses *Wiener* berdimensi *M*.

## 2.3 Persamaan Diferensial Stokastik (PDS)

Secara umum Persamaan Diferensial Stokastik (PDS) memiliki bentuk sebagai berikut:

$$dX(t) = f\big(X(t)\big)dt + g\big(X(t)\big)dW(t)\ ,\ \ X(0) = X_0\ ,\ \ 0 \leq t \leq T \qquad (2.1)$$

$f\big(X(t)\big)$ adalah koefisien *drift* dan bersifat deterministik, sedangkan $g\big(X(t)\big)$ adalah koefisien *diffusion* yang bersifat stokastik, dan $X(0) = X_0$ adalah nilai awal dan $W(t)$ adalah sebuah proses *Wiener*. Berikut ini akan dibahas mengenai proses *Wiener*.

### 2.3.1 Proses *Wiener*

Proses *Wiener* atau dikenal juga sebagai gerak *Brown*, pertama kali ditemukan oleh R. Brown pada tahun 1828. Menurut Higham D. J. (2001), Gerak

*Brown* pada $[0,T]$ adalah variabel random $W(t)$ yang bergantung secara kontinu pada $t \in [0,T]$ dan memenuhi 3 sifat berikut:

1. $W(0) = 0$.
2. Untuk $0 \leq s < t \leq T$, $W(t) - W(s)$ berdistribusi normal dengan mean nol dan variansi $t - s$ atau dengan kata lain:
$$W(t) - W(s) \sim \sqrt{t-s}\, N(0,1).$$
3. Untuk $0 \leq s < t < u < v \leq T$, $W(t) - W(s)$ dan $W(v) - W(u)$ saling bebas.

Untuk kebutuhan komputasi, maka interval waktu $[0,T]$ didiskretisasi pada tiap $t \in [0,T]$. Misalkan $\Delta t = T/L$ dengan $L$ bilangan bulat positif. $L$ menyatakan banyaknya titik diskretisasi pada $[0,T]$. Misalkan $t_j = j\Delta t$ dimana $j = 1,2,\dots,L$. Kondisi 1 menjelaskan $W(0) = 0$, sedangkan kondisi 2 dan 3 menjelaskan:

$$W(t_j) = W(t_{j-1}) + dW(t_j), \quad j = 1,2,\dots,L$$

dimana setiap $dW(t_j)$ merupakan variabel random berdistribusi $\sqrt{\Delta t}\, N(0,1)$. Gerak *Brown* bersifat *nowhere differentiable* dengan probabilitas 1 (Evans, 2003) sehingga untuk mencari solusi dari suatu PDS yang dipengaruhi oleh suatu proses *Wiener*, tidak dapat digunakan cara seperti pada integral biasa (misal jumlah *Riemann*). Untuk menyelesaikan PDS pada persamaan (2.1), salah satu metode numerik yang dapat digunakan adalah metode Euler-Maruyama yang akan dibahas pada subbab selanjutnya.

## 2.4 Metode Euler-Maruyama (EM)

Untuk menerapkan metode numerik dalam menyelesaikan PDS pada persamaan (1) pada $[0,T]$, pertama-tama diskretisasi interval $[0,T]$. Misalkan $\Delta t = T/L$ untuk $L$ bilangan bulat positif dan $L$ merupakan banyaknya titik diskretisasi pada $[0,T]$. Misalkan $t_j = j\Delta t$ dimana $j = 1,2,\dots,L$. Berdasarkan Higham D. J. (2001), aproksimasi solusi $X(t_j)$ dengan metode Euler-Maruyama memiliki bentuk sebagai berikut:

$$X(t_j) = X(t_{j-1}) + f\left(X(t_{j-1})\right)\Delta t + g\left(X(t_{j-1})\right)dW(t_j) \quad (2.2)$$

dimana $j = 1,2,\dots,L$. Untuk memahami darimana (2.2) berasal, maka perhatikan bentuk berikut ini:

$$X(t_j) = X(t_{j-1}) + \int_{t_{j-1}}^{t_j} f(X(s))ds + \int_{t_{j-1}}^{t_j} g(X(s))dW(s).$$

Selanjutnya akan dibahas untuk menghitung $\int_{t_{j-1}}^{t_j} f(X(s))ds$ dan $\int_{t_{j-1}}^{t_j} g(X(s))dW(s)$. Diberikan sebuah fungsi $f(X(s))$, maka $\int_{t_{j-1}}^{t_j} f(X(s))ds$ dapat diaproksimasi dengan menggunakan jumlah Riemann, yaitu:

$$\sum_{j=0}^{N-1} f(X(t_j))(t_{j+1} - t_j) \quad (2.3)$$

dimana $t_j = j\Delta t$. Nilai dari $\int_{t_{j-1}}^{t_j} f(X(s))ds$ didefinisikan sebagai limit $\Delta t \to 0$ dari (2.3).

Secara analog integral $\int_{t_{j-1}}^{t_j} g(X(s))dW(s)$ juga dapat diaproksimasi dengan cara seperti jumlah *Riemann*, yaitu:

$$\sum_{j=0}^{N-1} g(X(t_j))\left(W(t_{j+1}) - W(t_j)\right). \quad (2.4)$$

Pada kasus ini, fungsi $g(X(s))$ diintegralkan atas sebuah proses *Wiener*. Penyelesaian integral stokastik $\int_{t_{j-1}}^{t_j} g(X(s))dW(s)$ dengan menggunakan (2.4) dikenal sebagai integral Ito.

## 2.5 Sistem PDS

Secara umum Sistem PDS berdimensi *N* dengan proses *Wiener* berdimensi *M* memiliki bentuk sebagai berikut:

$$dX_i(t) = \sum_{j=1}^{N} a_{i,j}\, X_j(t)dt + \sum_{k=1}^{M}\sum_{j=1}^{N} b_{i,j}^{k}\, X_j(t)dW_k(t)\ , \boldsymbol{X}(0) = \boldsymbol{X_0}, 0 \le t \le T \tag{2.5}$$

dimana $i = 1,2,\dots,N$ dan $W_1(t), W_2(t), \dots, W_M(t)$ adalah proses *Wiener* yang saling bebas (Higham & Kloeden, 2002). Persamaan (2.5) dapat dinyatakan dalam bentuk matriks yaitu sebagai berikut:

$$d\begin{bmatrix} X_1(t) \\ X_2(t) \\ \vdots \\ X_N(t) \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,N} \\ a_{2,1} & a_{2,2} & & a_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N,1} & a_{N,2} & \cdots & a_{N,N} \end{bmatrix}\begin{bmatrix} X_1(t) \\ X_2(t) \\ \vdots \\ X_N(t) \end{bmatrix} dt + \begin{bmatrix} b_{1,1}^{1} & b_{1,2}^{1} & \cdots & b_{1,N}^{1} \\ b_{2,1}^{1} & b_{2,2}^{1} & & b_{2,N}^{1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{N,1}^{1} & b_{N,2}^{1} & \cdots & b_{N,N}^{1} \end{bmatrix}$$

$$\begin{bmatrix} X_1(t) \\ X_2(t) \\ \vdots \\ X_N(t) \end{bmatrix} dW_1(t) + \begin{bmatrix} b_{1,1}^{2} & b_{1,2}^{2} & \cdots & b_{1,N}^{2} \\ b_{2,1}^{2} & b_{2,2}^{2} & & b_{2,N}^{2} \\ \vdots & \vdots & \ddots & \vdots \\ b_{N,1}^{2} & b_{N,2}^{2} & \cdots & b_{N,N}^{2} \end{bmatrix}\begin{bmatrix} X_1(t) \\ X_2(t) \\ \vdots \\ X_N(t) \end{bmatrix} dW_2(t) + \cdots +$$

$$\begin{bmatrix} b_{1,1}^{M} & b_{1,2}^{M} & \cdots & b_{1,N}^{M} \\ b_{2,1}^{M} & b_{2,2}^{M} & & b_{2,N}^{M} \\ \vdots & \vdots & \ddots & \vdots \\ b_{N,1}^{M} & b_{N,2}^{M} & \cdots & b_{N,N}^{M} \end{bmatrix}\begin{bmatrix} X_1(t) \\ X_2(t) \\ \vdots \\ X_N(t) \end{bmatrix} dW_M(t)$$

Secara ringkas dapat dinyatakan dengan:

$$d\boldsymbol{X}(t) = A\boldsymbol{X}(t)dt + \sum_{k=1}^{M} B^k \boldsymbol{X}(t)dW_k(t)\ , \qquad \boldsymbol{X}(0) = \boldsymbol{X_0}$$

dimana

$$\boldsymbol{X} = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_N \end{bmatrix}, A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,N} \\ a_{2,1} & a_{2,2} & & a_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N,1} & a_{N,2} & \cdots & a_{N,N} \end{bmatrix}, \boldsymbol{X_0} = \begin{bmatrix} X_{1,0} \\ X_{2,0} \\ \vdots \\ X_{N,0} \end{bmatrix}, \text{dan}$$

$$B^k = \begin{bmatrix} b_{1,1}^{k} & b_{1,2}^{k} & \cdots & b_{1,N}^{k} \\ b_{2,1}^{k} & b_{2,2}^{k} & & b_{2,N}^{k} \\ \vdots & \vdots & \ddots & \vdots \\ b_{N,1}^{k} & b_{N,2}^{k} & \cdots & b_{N,N}^{k} \end{bmatrix}.$$

Selanjutnya penyelesaian masing-masing PDS dalam sistem PDS berdimensi *N* dengan proses *Wiener* berdimensi *M* sama seperti penyelesaian PDS dengan metode Euler-Maruyama yang telah dibahas sebelumnya.

# BAB 3
# ALGORITMA PARALEL UNTUK MODEL PDS

Pada bab ini akan dibahas algoritma paralel untuk model Persamaan Diferensial Stokastik (PDS). Secara garis besar, algoritma paralel untuk model PDS yang akan dibahas terdiri dari dua jenis. Algoritma paralel yang pertama adalah algoritma paralel secara *sample path* untuk PDS. Sedangkan algoritma paralel yang kedua adalah algoritma paralel untuk Sistem PDS berdimensi $N$ dengan proses *Wiener* berdimensi $M$. Pada akhir bab ini akan dibahas algoritma paralel untuk suatu model Persamaan Diferensial Stokastik (PDS) pada bidang keuangan.

## 3.1 Algoritma untuk Menyelesaikan PDS

Pada subbab ini akan dibangun algoritma untuk mengaproksimasi solusi dari PDS dengan metode Euler-Maruyama (EM). Misalkan diberikan PDS sebagai berikut:

$$dX(t) = f\big(X(t)\big)dt + g\big(X(t)\big)dW(t),\ X(0) = X_0\ , 0 \le t \le T \qquad (3.1)$$

dimana $W(t)$ adalah suatu proses *Wiener*. PDS pada persamaan (6) dapat diselesaikan dengan metode Euler-Maruyama (EM), yaitu:

$$X(t_j) = X(t_{j-1}) + f\left(X(t_{j-1})\right)\Delta t + g\left(X(t_{j-1})\right)dW(t_j)\ ,\ j = 1,2,\dots,L \qquad (3.2)$$

dengan $L$ titik diskretisasi pada $[0,T]$, $\Delta t = T/L$, dan $t_j = j\Delta t$, dimana $L$ bilangan bulat positif dan $dW(t_j) = W(t_j) - W(t_{j-1})$. Array $[W(t_0), W(t_1), W(t_2), \dots, W(t_L)]$ dikenal sebagai *discretized Brown path*. Sedangkan solusi $X(t)$ dimana $0 \le t \le T$ dikenal sebagai *sample path*. Karena $X(t)$ mengandung unsur stokastik, maka biasanya melibatkan lebih dari satu simulasi. Setiap simulasi menghasilkan *sample path* tersendiri.

Algoritma 3.1 adalah algoritma untuk menyelesaikan PDS pada persamaan (3.1) dengan metode EM pada persamaan (3.2). Algoritma ini bekerja secara sekuensial (lihat Tabel 3.1). Input dari algoritma ini adalah banyaknya simulasi

yaitu $S$, $f(X(t))$ yang merupakan koefisien *drift*, koefisien *diffusion* yang merupakan $g(X(t))$, $T$ yang merupakan batas atas interval $[0,T]$, nilai awal $Xzero$, dan banyaknya titik diskretisasi yaitu $L$.

Tabel 3.1 Algoritma untuk menyelesaikan PDS

**Algoritma 3.1 Euler-Maruyama method for SDE**

INPUT : $S$ is number of simulations, $f(X(t))$ and $g(X(t))$ are drift coefficient and diffusion coefficient respectively, $T$ is upper bound of interval $[0,T]$, $Xzero$ is initial value of SDE, $L$ is number of discretized point.

OUTPUT : $X$ is an $S$-by-$L$ array of Euler-Maruyama solution to the SDE:

$$dX(t) = f(X(t))dt + g(X(t))dW(t), X(0) = Xzero,\ 0 \le t \le T$$

1 **set** $\Delta t := T/L$

2 **set** $dW$ is an $S$-by-$L$ array of random number from normally distributed $\sqrt{\Delta t}\ N(0,1)$

3 **set** $X$ is an $S$-by-$(L+1)$ zeros array

4 **for** $i = 1$ to $S$ **do**

5 $X_{i,1} := Xzero$

6 **for** $j = 1$ to $L$ **do**

7 $X_{i,j+1} := X_{i,j} + f(X_{i,j}) * \Delta t + g(X_{i,j}) * dW_{i,j}$

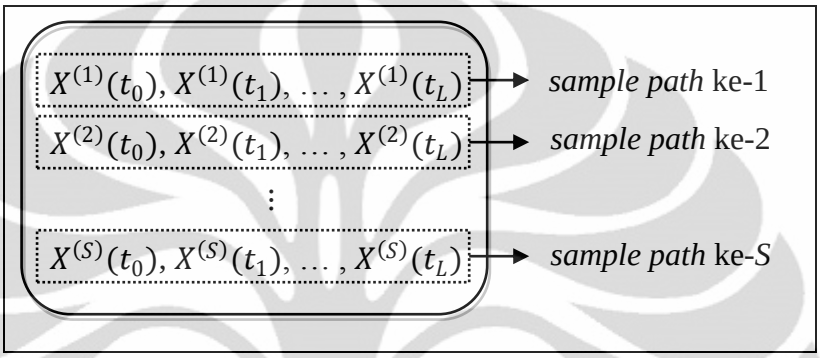
**end for**

**end for**

Langkah pertama adalah menghitung $\Delta t = T/L$. Titik diskretisasi $t_1$ adalah $\Delta t$, titik diskretisasi $t_2$ adalah $2 * \Delta t$, atau secara umum titik diskretisasi $t_j$ adalah $j * \Delta t$ dimana $j = 1,2,\dots,L$. Selanjutnya dihitung $dW(t_j) = W(t_j) - W(t_{j-1})$, $dW$ merupakan matriks berukuran $S\ x\ L$ berisi bilangan random berdistribusi $\sqrt{\Delta t}\ N(0,1)$. Dalam hal ini $dW_{i,j}$ adalah entri baris ke-$i$ kolom ke-$j$ dimana

$$dW_{i,j} = W_{i,j} - W_{i,j-1}\,,\ \ i = 1,2,\dots,S\,,\ \ j = 1,2,\dots,L. \qquad (3.3)$$

Selanjutnya akan dihitung solusi EM dari (3.1). $X_{i,j}$ adalah entri baris ke-$i$ kolom ke-$j$ dari $X$ yang merupakan solusi EM pada titik diskretisasi $t_{j-1}$ dari simulasi ke-$i$. Pada langkah 3, inisialisasi $X$ sebagai matriks berukuran $S\ x\ (L+1)$ yang semua entrinya adalah nol. Matriks $X$ akan digunakan untuk menyimpan solusi EM dengan $S$ simulasi.




Gambar 3.1 Visualisasi *S sample path*

Pada Gambar 3.1, $X^{(i)}(t_j)$ merupakan solusi EM pada titik diskretisasi ke-$j$ untuk sample path ke-$i$. Untuk menghitung *S sample path* solusi EM dari (3.1) dilakukan iterasi dari 1 sampai *S* (langkah 4 hingga langkah 7). Untuk mendapatkan *sample path* ke-$i$, pertama-tama inisialisasi $X_{i,1} = Xzero$. $X_{i,1}$ merupakan solusi EM pada titik diskretisasi $t_0$ dari simulasi ke-$i$. Untuk mendapatkan 1 *sample path* solusi EM dilakukan iterasi dari 1 sampai *L* (langkah 6 hingga langkah 7). Setelah dihitung $X_{i,j+1} := X_{i,j} + f(X_{i,j}) * \Delta t + g(X_{i,j}) * dW_{i,j}$ untuk $j = 1,2,\dots,L$, maka diperoleh matriks $X$ sebagai berikut:

$$X = \begin{bmatrix} X_{1,1} & X_{1,2} & \cdots & X_{1,L+1} \\ X_{2,1} & X_{2,2} & & X_{2,L+1} \\ \vdots & \vdots & \ddots & \vdots \\ X_{S,1} & X_{S,2} & \cdots & X_{S,L+1} \end{bmatrix}.$$

Dalam hal ini, entri baris ke-$i$ dari $X$ adalah solusi EM untuk PDS pada persamaan (3.1) dari simulasi ke-$i$. Sedangkan kolom ke-$j$ dari $X$ menyatakan titik diskretisasi $t_{j-1}$. Berarti $X_{i,1}$ adalah $X^{(i)}(t_0)$, $X_{i,2}$ adalah $X^{(i)}(t_1)$, …, $X_{i,L+1}$ adalah $X^{(i)}(t_L)$.

Pada bidang aplikasi tertentu, misalnya bidang keuangan, biasanya melibatkan nilai $S$ yang besar. Akibatnya waktu komputasi dari algoritma ini akan meningkat. Oleh karena itu, akan dibangun algoritma paralel untuk PDS sehingga diharapkan dapat mempercepat waktu komputasi. Pada subbab berikutnya akan dibahas algoritma paralel secara *sample path* untuk menyelesaikan PDS dengan metode Euler-Maruyama.

## 3.2 Algoritma Paralel secara *Sample Path* untuk PDS

Pada subbab ini akan dibangun algoritma paralel secara *sample path* untuk mengaproksimasi solusi dari PDS pada persamaan (3.1). Perhatikan kembali bentuk dari metode Euler-Maruyama, yaitu

$$X(t_j) = X(t_{j-1}) + f\left(X(t_{j-1})\right)\Delta t + g\left(X(t_{j-1})\right)dW(t_j), \qquad j = 1,2,\dots,L.$$

Perhatikan bahwa untuk menghitung $X(t_1)$ diperlukan nilai $X(t_0)$**,** untuk menghitung $X(t_2)$ diperlukan nilai $X(t_1)$**,** atau secara umum untuk menghitung $X(t_j)$ diperlukan nilai $X(t_{j-1})$. Berarti pada kasus ini mengandung *data dependence* sehingga untuk menghitung $X(t_1)$, $X(t_2)$,…, $X(t_L)$ tidak dapat dikerjakan secara bersamaan. Jika kembali ke Algoritma 3.1, maka iterasi pada langkah 6 hingga 7, yaitu:

**for** $j = 1$ to $L$ **do**
    $X_{i,j+1} := X_{i,j} + f(X_{i,j}) * \Delta t + g(X_{i,j}) * dW_{i,j}$
**end for**

tidak dapat dikerjakan secara paralel karena mengandung *data dependence*. Tetapi perhatikan iterasi pada langkah 4 sampai 7 pada Algoritma 3.1, yaitu:

**for** $i = 1$ to $S$ **do**
    $X_{i,1} := Xzero$
    **for** $j = 1$ to $L$ **do**
        $X_{i,j+1} := X_{i,j} + f(X_{i,j}) * \Delta t + g(X_{i,j}) * dW_{i,j}$
    **end for**
**end for**

Untuk $i = 1$ dilakukan penghitungan *sample path* pertama, untuk $i = 2$ dilakukan penghitungan *sample path* kedua, dan seterusnya hingga $i = S$ dilakukan penghitungan *sample path* ke-$S$. Pada saat mengerjakan $S$ *sample path* diperlukan nilai-nilai $f(X(t))$, $g(X(t))$, $\Delta t$, $Xzero$, $S$, $L$, dan $dW$. Untuk nilai $f(X(t))$, $g(X(t))$, $\Delta t$, $Xzero$, $S$, dan $L$ dapat diakses secara bersamaan karena nilai-nilai tersebut bisa disimpan pada *shared memory*. Sekarang akan ditinjau apakah $dW$ mengandung *data dependence* atau tidak. Pada saat mengerjakan *sample path* ke-1 diperlukan entri dari baris pertama matriks $dW$, sedangkan pada saat mengerjakan *sample path* ke-2 diperlukan entri dari baris kedua matriks $dW$, dan seterusnya hingga pada saat mengerjakan *sample path* ke-$S$ diperlukan entri dari baris ke-$S$ matriks $dW$. Karena baris pertama, baris kedua, hingga baris ke-$S$ dari $dW$ saling bebas, maka semua baris tersebut dapat diakses oleh semua prosesor secara bersamaan. Jadi, untuk mengerjakan *sample path* ke-2 tidak bergantung dengan *sample path* ke-1. Untuk mengerjakan *sample path* ke-3 tidak bergantung dengan *sample path* ke-2 Secara umum, untuk mengerjakan *sample path* ke-$i$ tidak bergantung dengan *sample path* ke-$(i-1)$ dimana $i = 2,3, \dots, S$. Dengan demikian untuk menghitung *sample path* ke-1 hingga *sample path* ke-$S$ dapat dikerjakan secara bersamaan. Karena yang dikerjakan secara paralel adalah ketika proses penghitungan *sample path,* maka algoritma paralel seperti ini dinamakan algoritma paralel secara *sample path* untuk PDS.

Berikut ini akan dibangun algoritma paralel secara *sample path* untuk PDS. Misalkan *p* prosesor ditugaskan untuk mengerjakan solusi EM untuk PDS pada persamaan (3.1) dengan $S$ *sample path*. Masalah yang dihadapi adalah menghitung $S$ *sample path* dimana $S$ *sample path* tersebut saling bebas sehingga dapat dikerjakan secara paralel. Ide paralel pada kasus ini adalah menugaskan setiap prosesor untuk mengerjakan sejumlah *sample path* yang sama banyak, namun pada kenyataannya, kondisi tersebut tidak selalu dapat terjadi. Berikut ini akan dibahas pembagian tugas untuk masing-masing prosesor, yaitu:

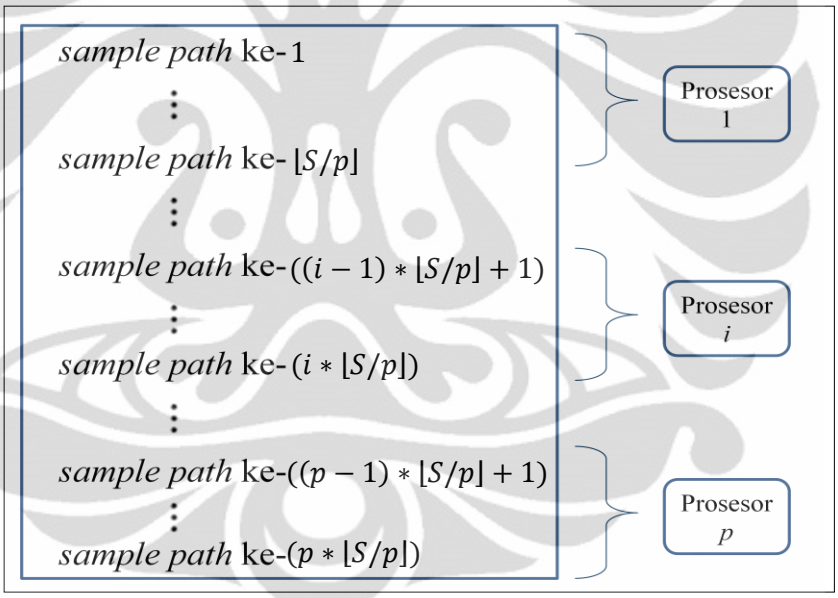
- Kasus $S \leq p$.

  Pada kasus ini, masing-masing prosesor ditugaskan untuk mengerjakan 1 *sample path*, maka dalam kasus ini kondisi *load balance* terpenuhi.

- Kasus $S > p$.

  Pada kasus ini, terdapat 2 subkasus yaitu kasus $p$ habis membagi $S$ dan $p$ tidak habis membagi $S$. Untuk kasus $p$ habis membagi $S$, masing-masing prosesor ditugaskan untuk mengerjakan $\lfloor S/p \rfloor$ *sample path*, maka dalam kasus ini kondisi *load balance* terpenuhi. Sedangkan untuk kasus $p$ tidak habis membagi $S$, maka kondisi *load balance* tidak terpenuhi karena prosesor ke-1 hingga prosesor ke-$mod(S,p)$ mengerjakan$\lfloor S/p \rfloor + 1$ *sample path*, tetapi prosesor ke-$mod(S,p) + 1$ hingga prosesor ke-$p$ mengerjakan $\lfloor S/p \rfloor$ *sample path*.

Agar memenuhi *concurrency*, maka masing-masing prosesor mengerjakan tugasnya pada saat yang bersamaan.

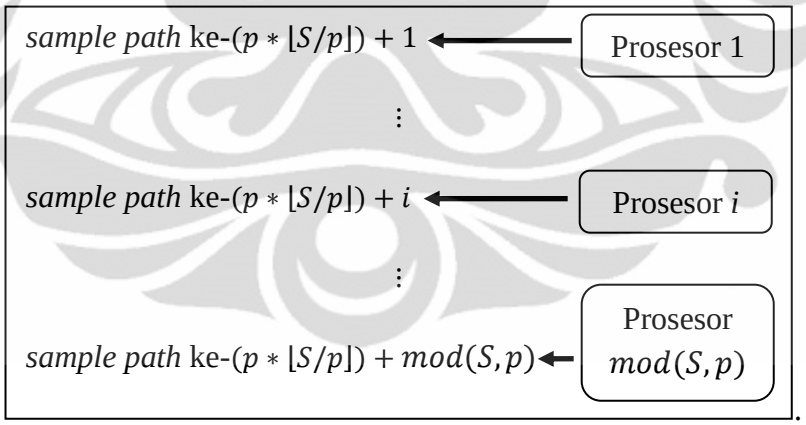



Gambar 3.2 Distribusi $S$ *sample path* pada $p$ prosesor untuk kasus $p$ habis membagi $S$

Untuk kasus $S \leq p$, pembagian tugasnya sangat mudah, yaitu prosesor ke-$i$ mengerjakan *sample path* ke-$i$ dimana $i = 1,2, \dots, p$, sehingga untuk mengerjakan $S$ *sample path* hanya diperlukan $S$ prosesor. Untuk kasus $S > p$ dimana $p$ habis membagi $S$, pembagian tugasnya adalah prosesor ke-$i$ mengerjakan *sample path* ke-$((i-1) * \lfloor S/p \rfloor + 1)$ hingga *sample path* ke-$(i * \lfloor S/p \rfloor)$ dimana $i = 1,2, \dots, p$ (lihat Gambar 3.2). Sedangkan untuk kasus $S > p$ dimana $p$ tidak habis membagi $S$, pembagian tugasnya sama seperti kasus $p$

habis membagi $S$, namun perhatikan bahwa hanya *sample path* ke-1 hingga *sample path* ke-$(p * \lfloor S/p \rfloor)$ yang telah dihitung. Berarti sebanyak $mod(S,p)$ *sample path* belum dihitung, yaitu *sample path* ke-$(p * \lfloor S/p \rfloor) + 1$ hingga *sample path* ke-$(p * \lfloor S/p \rfloor) + mod(S,p)$ dimana *sample path* ke-$(p * \lfloor S/p \rfloor) + mod(S,p)$ merupakan *sample path* ke-$S$. Oleh karena itu, masing-masing prosesor $i$ dimana $i = 1,2, \dots, mod(S,p)$ ditugaskan untuk mengerjakan 1 *sample path* (lihat Gambar 3.3).

Jadi, untuk kasus $p$ habis membagi $S$, masing-masing prosesor ditugaskan untuk mengerjakan $\lfloor S/p \rfloor$ *sample path*, maka dalam kasus ini kondisi *load balance* terpenuhi. Sedangkan untuk kasus $p$ tidak habis membagi $S$, maka kondisi *load balance* tidak terpenuhi karena prosesor ke-1 hingga prosesor ke-$mod(S,p)$ mengerjakan$\lfloor S/p \rfloor + 1$ *sample path*, tetapi prosesor ke-$mod(S,p) + 1$ hingga prosesor ke-$p$ mengerjakan $\lfloor S/p \rfloor$ *sample path*. Sebagai contoh, misalkan 4 prosesor ditugaskan untuk mengerjakan 19 *sample path*. Karena $\lfloor 19/4 \rfloor = 4$ dan $mod(19,4) = 3$, maka prosesor 1, prosesor 2, dan prosesor 3 mengerjakan $(4 + 1)$ *sample path* sedangkan prosesor 4 mengerjakan 4 *sample path*.




Gambar 3.3 Distribusi *mod* (*S*,*p*) *sample path* pada *mod* (*S*,*p*) prosesor untuk kasus *p* tidak habis membagi *S*.

Berikut ini akan dibahas Algoritma 3.2 yaitu algoritma paralel secara *sample path* untuk PDS (lihat Tabel 3.2). Pada algoritma ini akan digunakan *p* prosesor. Input dari algoritma ini adalah banyaknya simulasi yaitu $S$, $f(X(t))$ yang merupakan koefisien *drift*, koefisien *diffusion* yang merupakan $g(X(t))$, $T$

yang merupakan batas atas interval $[0, T]$, nilai awal $Xzero$, banyaknya titik diskretisasi yaitu $L$, dan banyaknya prosesor yaitu $p$ dimana $S \geq p$.

Tabel 3.2 Algoritma paralel secara *sample path* untuk PDS

**Algoritma 3.2 Euler-Maruyama method for SDE in parallel**

INPUT : $S$ is a number of simulations, $f(X(t))$ and $g(X(t))$ are drift coefficient and diffusion coefficient respectively, $T$ is upper bound of interval $[0, T]$, $Xzero$ is initial value of SDE, $L$ is a number of discretized point, $p$ is a number of processors where $S \geq p$.

OUTPUT : $X$ is an $S$-by-$L$ array of Euler-Maruyama solution to the SDE:

$$dX(t) = f(X(t))dt + g(X(t))dW(t), \;\; X(0) = X_0, \;\; 0 \leq t \leq T$$

1. **set** $\Delta t := T/L$
2. **set** $dW$ is an $S$-by-$L$ array of random number from normally distributed $\sqrt{\Delta t}\, N(0,1)$
3. **set** $X$ is an $S$-by-$(L+1)$ zeros array
4. Each processor $i$ where $i = 1,2, \dots, p$ **do in parallel**
5. **if** $mod(S, p) \neq 0$ **do**
6. Each processor $e$ where $e = 1,2, \dots, mod(S, p)$ **do in parallel**
7. **set** $X\_local$ is an $(\lfloor S/p \rfloor + 1)$-by-$(L+1)$ zeros array
8. **set** $Winc\_end$ is an 1-by-$L$ array whose entries are $(p * \lfloor S/p \rfloor + e)$th row of $dW$
9. $X\_local_{\lfloor S/p \rfloor + 1,1} := Xzero$
10. **for** $j = 1$ to $L$ **do**
11. $X\_local_{\lfloor S/p \rfloor + 1, j+1} := X\_local_{\lfloor S/p \rfloor + 1, j} + f(X\_local_{\lfloor S/p \rfloor + 1, j}) * \Delta t + f(X\_local_{\lfloor S/p \rfloor + 1, j}) * Winc\_end_j$

**end for**

12. **set** $X_{p * \lfloor S/p \rfloor + e, :} = X\_local_{\lfloor S/p \rfloor + 1, :}$

**end do**

13. **set** $X\_local$ is an $\lfloor S/p \rfloor$-by-$(L+1)$ zeros array for processor $f$ where

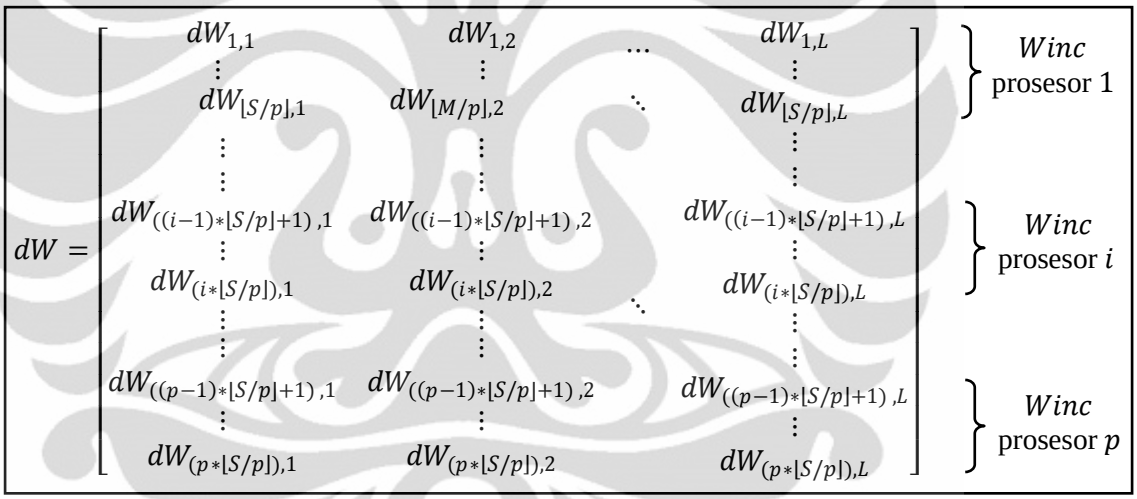
$f = mod(S,p)+1, mod(S,p)+2, \ldots, p$

14. **else**
15. **set** $X\_local$ is an $\lfloor S/p \rfloor$-by-$(L+1)$ zeros array for each processor
    **end if**
16. **set** $Winc$ is an $\lfloor S/p \rfloor$-by-$L$ array whose entries are
    $((i-1)*\lfloor S/p \rfloor + 1)$th row until $(i*\lfloor S/p \rfloor)$th row of $dW$
17. **for** $h = 1$ to $\lfloor S/p \rfloor$ **do**
18. $X\_local_{h,1} := Xzero$
19. **for** $j = 1$ to $L$ **do**
20. $X\_local_{h,j+1} := X\_local_{h,j} + f\left(X\_local_{h,j}\right) * \Delta t + g\left(X\_local_{h,j}\right) * Winc_{h,j}$
    **end for**
    **end for**
21. **set** $X_{(i-1)*\lfloor S/p \rfloor+1:i*\lfloor S/p \rfloor,:} = X\_local_{1:\lfloor S/p \rfloor,:}$
    **end do**

Langkah 1 hingga langkah 3 sama seperti pada Algoritma 3.1. Nilai-nilai $S, f\left(X(t)\right), g(X(t)), T, \Delta t, Xzero, L$ dan $dW$ disimpan pada *shared memory* sehingga dapat diakses oleh semua prosesor. Langkah selanjutnya merupakan proses paralel yang dimulai pada langkah 4 dengan menggunakan $p$ prosesor. Setiap prosesor $i$ dimana $i = 1,2, \ldots, p$ memiliki variabel lokal yaitu $X\_local$ untuk menyimpan solusi EM. Langkah 5 hingga 13 dilakukan apabila $p$ tidak habis membagi $S$. Langkah 6 hingga 12 dilakukan oleh prosesor $e$ dimana $e = 1,2, \ldots, mod(S,p)$. Pada langkah 7, inisialisasi $X\_local$ sebagai matriks berukuran $(\lfloor S/p \rfloor + 1\ x\ (L+1)$ yang semua entrinya adalah nol. Lalu langkah 8, $Winc\_end$ menyimpan nilai $dW$ yang diperlukan oleh prosesor $e$ untuk mengerjakan 1 *sample path*. Langkah 9 hingga 11 merupakan proses penghitungan *sample path* ke-$(p*\lfloor S/p \rfloor + 1)$ hingga *sample path* ke-$S$ secara bersamaan dan hasilnya disimpan dalam matriks $X\_local$ prosesor $e$ pada baris ke-$(\lfloor S/p \rfloor + 1)$. Pada langkah 12, meng-*update* entri baris ke-$(p*\lfloor S/p \rfloor + e)$ dari $X$ dengan entri baris ke-$(\lfloor S/p \rfloor + 1)$ dari $X\_local$. Kemudian pada langkah 13, inisialisasi $X\_local$ pada prosesor $f$ dimana $f = mod(S,p)+1, mod(S,p)+$

$2, \dots, p$ sebagai matriks berukuran $(\lfloor S/p \rfloor \; x \; (L+1)$ yang semua entrinya adalah nol. Langkah 15 dilakukan apabila $p$ habis membagi $S$, yaitu inisialisasi $X\_local$ pada prosesor $i$ dimana $i = 1,2, \dots, p$ sebagai matriks berukuran $(\lfloor S/p \rfloor \; x \; (L+1)$ yang semua entrinya adalah nol.

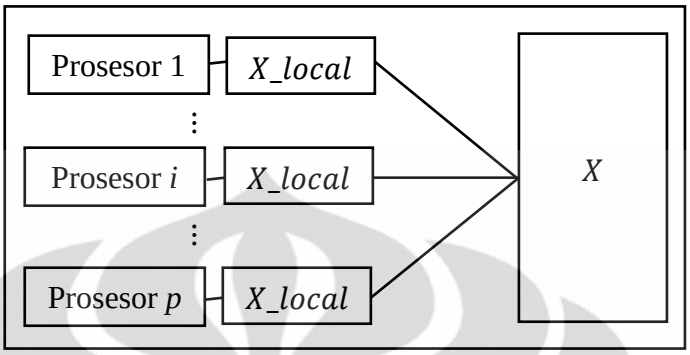
Kemudian langkah 16 adalah membuat variabel lokal $Winc$ pada prosesor $i$ dimana $i = 1,2, \dots, p$ yang digunakan untuk menyimpan nilai baris tertentu dari $dW$ yang diperlukan oleh masing-masing prosesor untuk menghitung $\lfloor S/p \rfloor$ *sample path*. $Winc$ adalah matriks berukuran $\lfloor S/p \rfloor \; x \; L$ dan entri dari $Winc$ adalah entri baris ke-$((i-1) * \lfloor S/p \rfloor + 1)$ sampai baris ke-$(i * \lfloor S/p \rfloor)$ dari $dW$ dimana $i = 1,2, \dots, p$. Jadi, setiap prosesor $i$ memiliki variabel local $Winc$ yang nilainya berbeda-beda (lihat Gambar 3.4).




Gambar 3.4 Distribusi $dW$ pada $p$ prosesor

Langkah berikutnya adalah setiap prosesor $i$ menghitung $\lfloor S/p \rfloor$ *sample path* secara bersamaan. Setiap prosesor $i$ melakukan proses penghitungan *sample path* ke-$((i-1) * \lfloor S/p \rfloor + 1)$ hingga *sample path* ke-$(i * \lfloor S/p \rfloor)$ dan disimpan dalam matriks $X\_local$ pada baris ke-1 hingga baris ke-$\lfloor S/p \rfloor$. Setelah mendapatkan nilai $X\_local$ pada masing-masing prosesor $i$, maka pada langkah 21 meng-*update* entri baris ke-$((i-1) * \lfloor S/p \rfloor + 1)$ hingga baris ke-$(i * \lfloor S/p \rfloor)$ dari $X$ dengan entri baris ke-1 hingga baris ke-$\lfloor S/p \rfloor$ dari $X\_local$ pada masing-masing prosesor $i$. Kemudian proses paralel diakhiri dan telah diperoleh matriks $X$ yang

berisi solusi EM untuk *S sample path.* Skema algoritma paralel secara *sample path* untuk PDS dapat dilihat pada Gambar 3.5.




Gambar 3.5 Skema algoritma paralel secara *sample path* untuk PDS

Dengan demikian diharapkan waktu komputasi menjadi lebih cepat dalam menyelesaikan PDS pada persamaan (6) apabila menggunakan Algoritma 3.2 yang merupakan algoritma paralel secara *sample path* untuk PDS. Implementasi dari algoritma ini akan diberikan pada Bab 4.

## 3.3 Algoritma untuk Menyelesaikan Sistem PDS

Setelah membangun algoritma paralel secara *sample path* untuk menyelesaikan PDS, maka selanjutnya akan dibangun algoritma paralel untuk menyelesaikan sistem PDS. Pada skripsi ini, sistem PDS yang digunakan adalah sistem PDS berdimensi *N* dengan proses *Wiener* berdimensi *M*. Untuk membangun algoritma paralel untuk sistem PDS, terlebih dahulu akan dibahas algoritma untuk menyelesaikan sistem PDS berdimensi *N* dengan proses *Wiener* berdimensi *M* yang. Pada bab sebelumnya telah diketahui bahwa sistem PDS berdimensi *N* dengan proses *Wiener* berdimensi *M* memiliki bentuk sebagai berikut:

$$dX_i(t) = \sum_{j=1}^{N} a_{i,j}\, X_j(t)dt + \sum_{k=1}^{M}\sum_{j=1}^{N} b_{i,j}^{k}\, X_j(t)dW_k(t), \;\; \boldsymbol{X}(0) = \boldsymbol{X_0}, 0 \le t \le T \tag{3.4}$$

dimana $i = 1,2,\dots,N$ dan $W_1(t), W_2(t), \dots, W_M(t)$ adalah proses *Wiener* yang saling bebas. Apabila persamaan (3.4) dijabarkan dan dinyatakan dalam bentuk matriks, maka akan menjadi seperti berikut:

$$d\begin{bmatrix} X_1(t) \\ X_2(t) \\ \vdots \\ X_N(t) \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,N} \\ a_{2,1} & a_{2,2} & & a_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N,1} & a_{N,2} & \cdots & a_{N,N} \end{bmatrix}\begin{bmatrix} X_1(t) \\ X_2(t) \\ \vdots \\ X_N(t) \end{bmatrix} dt + \begin{bmatrix} b_{1,1}^1 & b_{1,2}^1 & \cdots & b_{1,N}^1 \\ b_{2,1}^1 & b_{2,2}^1 & & b_{2,N}^1 \\ \vdots & \vdots & \ddots & \vdots \\ b_{N,1}^1 & b_{N,2}^1 & \cdots & b_{N,N}^1 \end{bmatrix}$$

$$\begin{bmatrix} X_1(t) \\ X_2(t) \\ \vdots \\ X_N(t) \end{bmatrix} dW_1(t) + \begin{bmatrix} b_{1,1}^2 & b_{1,2}^2 & \cdots & b_{1,N}^2 \\ b_{2,1}^2 & b_{2,2}^2 & & b_{2,N}^2 \\ \vdots & \vdots & \ddots & \vdots \\ b_{N,1}^2 & b_{N,2}^2 & \cdots & b_{N,N}^2 \end{bmatrix}\begin{bmatrix} X_1(t) \\ X_2(t) \\ \vdots \\ X_N(t) \end{bmatrix} dW_2(t) + \cdots +$$

$$\begin{bmatrix} b_{1,1}^M & b_{1,2}^M & \cdots & b_{1,N}^M \\ b_{2,1}^M & b_{2,2}^M & & b_{2,N}^M \\ \vdots & \vdots & \ddots & \vdots \\ b_{N,1}^M & b_{N,2}^M & \cdots & b_{N,N}^M \end{bmatrix}\begin{bmatrix} X_1(t) \\ X_2(t) \\ \vdots \\ X_N(t) \end{bmatrix} dW_M(t)$$

Secara ringkas dapat dinyatakan dengan:

$$d\boldsymbol{X}(t) = A\boldsymbol{X}(t)dt + \sum_{k=1}^{M} B^k \boldsymbol{X}(t) dW_k(t)\ , \qquad \boldsymbol{X}(0) = \boldsymbol{X_0} \tag{3.5}$$

dimana

$$\boldsymbol{X} = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_N \end{bmatrix}, A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,N} \\ a_{2,1} & a_{2,2} & & a_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N,1} & a_{N,2} & \cdots & a_{N,N} \end{bmatrix}, \boldsymbol{X_0} = \begin{bmatrix} X_{1,0} \\ X_{2,0} \\ \vdots \\ X_{N,0} \end{bmatrix}, \text{dan}$$

$$B^k = \begin{bmatrix} b_{1,1}^k & b_{1,2}^k & \cdots & b_{1,N}^k \\ b_{2,1}^k & b_{2,2}^k & & b_{2,N}^k \\ \vdots & \vdots & \ddots & \vdots \\ b_{N,1}^k & b_{N,2}^k & \cdots & b_{N,N}^k \end{bmatrix}.$$

Dalam sistem PDS berdimensi *N* dengan proses *Wiener* berdimensi *M* terdapat *N* PDS. Setiap PDS dalam sistem akan diselesaikan dengan metode EM. Pertama-tama menentukan banyaknya titik diskretisasi pada $[0, T]$, yaitu misalkan $L$. Lalu hitung $\Delta t = T/L$ dimana $L$ bilangan bulat positif. Misalkan $t_j = j\Delta t$, $dW_k(t_j) = W_k(t_j) - W_k(t_{j-1})$, dan $j = 1,2,\dots,L$, maka sistem PDS dapat diselesaikan dengan metode EM, yaitu

$$\boldsymbol{X}(t_j) = \boldsymbol{X}(t_{j-1}) + A\boldsymbol{X}(t_{j-1})\Delta t + \sum_{k=1}^{M} B^k \boldsymbol{X}(t_{j-1}) dW_k(t_j), \quad \boldsymbol{X}(0) = \boldsymbol{X_0} \tag{3.6}$$

Selanjutnya akan dibahas mengenai algoritma untuk menyelesaikan menyelesaikan sistem PDS berdimensi $N$ dengan proses *Wiener* berdimensi $M$ pada persamaan (3.5) dengan metode EM pada persamaan (3.6). Algoritma ini bekerja secara sekuensial (lihat Tabel 3.3). Input dari algoritma ini adalah banyaknya PDS pada sistem yaitu $N$, banyaknya proses *Wiener* yaitu $M$, $T$ yang merupakan batas atas interval $[0, T]$, nilai awal dari masing-masing PDS yaitu $X1zero, X2zero, \dots, XNzero$, banyaknya titik diskretisasi yaitu $L$, $A$ yang merupakan matriks koefisien *drift*, dan $B^k$ yang merupakan koefisien *diffusion*.

Tabel 3.3 Algoritma untuk menyelesaikan Sistem PDS

**Algoritma 3.3 Euler-Maruyama method for SDE system**

INPUT : $N$ is a number of SDEs, $M$ is a number of *Wiener* process, $T$ is upper bound of interval $[0, T]$, $X1zero, X2zero, \dots, XNzero$ is initial value of SDEs respectively, $L$ is number of discretized point, $A$ is drift coefficient matrix and $B^k$ for $k = 1,2, \dots, M$ is diffusion coefficient matrix where

$$A := \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,N} \\ a_{2,1} & a_{2,2} & & a_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N,1} & a_{N,2} & \cdots & a_{N,N} \end{bmatrix} \text{ and } B^k := \begin{bmatrix} b_{1,1}^k & b_{1,2}^k & \cdots & b_{1,N}^k \\ b_{2,1}^k & b_{2,2}^k & & b_{2,N}^k \\ \vdots & \vdots & \ddots & \vdots \\ b_{N,1}^k & b_{N,2}^k & \cdots & b_{N,N}^k \end{bmatrix}$$

respectively.

OUTPUT : $Xsys$ is an $N$-by-$(L+1)$ array where $i$th row is the Euler-Maruyama solution of $X_i$ to the SDE system:

$$d\boldsymbol{X}(t) = A\boldsymbol{X}(t)dt + \sum_{k=1}^{M} B^k \boldsymbol{X}(t) dW_k(t) \ , \boldsymbol{X}(0) = \boldsymbol{X_0}, 0 \le t \le T$$

1 $\Delta t := T/L$

2 **set** $dW\_1$ is a 1-by-$L$ array of random number from normally distributed $\sqrt{\Delta t}\, N(0,1)$

$dW\_2$ is a 1-by-$L$ array of random number from normally distributed $\sqrt{\Delta t}\, N(0,1)$

$\vdots$

$dW\_M$ is a 1-by-$L$ array of random number from normally distributed $\sqrt{\Delta t}\, N(0,1)$

3 **set** $Xsys$ is an $N$-by-$(L+1)$ zeros array

4 **set** $Xsys_{:,1} := \begin{bmatrix} X1zero \\ X2zero \\ \vdots \\ XNzero \end{bmatrix}$

5 **set** $B\_temp$ is an $N$-by-$M$ zeros array

6 **for** $j = 1$ to $L$ **do**

7 **for** k = 1 to $M$ **do**

8 **set** $B\_temp_{:,k} := B^k * Xsys_{:,j}$

**end for**

9 $Xsys_{:,j+1} := Xsys_{:,j} + A * Xsys_{:,j} * \Delta t + B\_temp * \begin{bmatrix} dW\_1_j \\ dW\_2_j \\ \vdots \\ dW\_M_j \end{bmatrix}$

**end for**

Langkah pertama adalah menghitung $\Delta t = T/L$. Titik diskretisasi $t_1$ adalah $\Delta t$, titik diskretisasi $t_2$ adalah $2 * \Delta t$, atau secara umum titik diskretisasi $t_j$ adalah $j * \Delta t$ dimana $j = 1,2, \dots, L$. Selanjutnya dihitung $dW_k(t_j) = W_k(t_j) - W_k(t_{j-1})$ dimana $k = 1, \dots, M,\ j = 1, \dots, L$. $dW\_1, dW\_2, \dots, dW\_N$ merupakan array berukuran $1\ x\ L$ berisi bilangan random berdistribusi $\sqrt{\Delta t}\, N(0,1)$. Dalam hal ini $dW\_1_j$ adalah entri ke-$j$ dari $dW\_1$ adalah nilai dari $dW_1(t_j) = W_1(t_j) - W_1(t_{j-1})$ dimana $j = 1, \dots, L$. $dW\_2_j$ adalah entri ke-$j$ dari $dW\_2$ adalah nilai dari $dW_2(t_j) = W_2(t_j) - W_2(t_{j-1})$ dimana $j = 1, \dots, L$. Demikian seterusnya hingga $dW\_M_j$ adalah entri ke-$j$ adalah entri ke-$j$ dari $dW\_M$ adalah nilai dari $dW_M(t_j) = W_M(t_j) - W_M(t_{j-1})$ dimana $j = 1, \dots, L$.

Proses berikutnya adalah menghitung solusi EM untuk sistem PDS. Pada langkah 3, inisialisasi $Xsys$ sebagai matriks berukuran $N\ x\ (L+1)$ yang semua entrinya adalah nol. Baris ke-$i$ dari $Xsys$ dimana $i = 1, \dots, N$ akan digunakan untuk menyimpan solusi EM dari PDS ke-$i$ dalam sistem yaitu $X_i$.
Pada langkah 4, $Xsys_{:,1}$ adalah kolom ke-1 dari $Xsys$ berisi nilai awal untuk masing-masing PDS yaitu $[X1zero; X2zero; \dots; XNzero]$. Untuk menghitung solusi EM dari masing-masing PDS pada tiap titik diskretisasi $t_j$ dimana $j = 1, \dots, L$, maka dilakukan iterasi dari 1 sampai $L$ (langkah 6 hingga langkah 9). Di dalam proses penghitungan solusi EM pada titik diskretisasi $t_j$, dilakukan proses iterasi dari $k = 1$ hingga $k = M$ untuk menghitung $B\_temp_{:,k} = B^k * Xsys_{:,j}$. $B\_temp_{:,k}$ adalah kolom ke-$k$ dari $B\_temp$ dan

$$B\_temp_{:,k} = \begin{bmatrix} b_{1,1}^k & b_{1,2}^k & \cdots & b_{1,N}^k \\ b_{2,1}^k & b_{2,2}^k & & b_{2,N}^k \\ \vdots & \vdots & \ddots & \vdots \\ b_{N,1}^k & b_{N,2}^k & \cdots & b_{N,N}^k \end{bmatrix} * Xsys_{:,j}.$$

Setelah itu dihitung $Xsys_{:,j+1} := Xsys_{:,j} + A * Xsys_{:,j} * \Delta t + B\_temp * \begin{bmatrix} dW\_1_j \\ dW\_2_j \\ \vdots \\ dW\_M_j \end{bmatrix}$ untuk $j = 1,2, \dots, L$, maka diperoleh matriks $Xsys$ sebagai berikut:

$$Xsys = \begin{bmatrix} Xsys_{1,1} & Xsys_{1,2} & \cdots & Xsys_{1,L+1} \\ X_{sys}2_{,1} & Xsys_{2,2} & & Xsys_{2,L+1} \\ \vdots & \vdots & \ddots & \vdots \\ Xsys_{N,1} & Xsys_{N,2} & \cdots & Xsys_{N,L+1} \end{bmatrix}.$$

Dalam hal ini, entri baris ke-$i$ dari $Xsys$ adalah solusi EM untuk PDS ke-$i$ dalam sistem yaitu $X_i$. Sedangkan kolom ke-$j$ dari $Xsys$ menyatakan titik diskretisasi $t_{j-1}$. Jadi, entri baris ke-$i$ kolom ke-$j$ dari $Xsys$ adalah solusi EM untuk $X_i$ pada titik diskretisasi $t_{j-1}$ dimana $i = 1,2, \dots, N$ dan $j = 1,2, \dots, L$.

Dalam berbagai bidang aplikasi seringkali dijumpai masalah sistem PDS melibatkan sistem PDS berukuran besar (melibatkan nilai $N$ yang besar). Apabila masalah sistem PDS tersebut dikerjakan secara sekuensial maka berdampak pada lamanya waktu komputasi yang diperlukan. Oleh karena itu, pada subbab selanjutnya akan dibangun algoritma paralel untuk sistem PDS sehingga diharapkan dapat mempercepat waktu komputasi.

## 3.4 Algoritma Paralel untuk Sistem PDS

Berikut ini akan dibangun algoritma paralel untuk mengaproksimasi solusi dari sistem PDS pada persamaan (3.5) dengan metode EM pada persamaan (3.6). Sebelum menbangun algoritma paralel untuk sistem PDS, perhatikan kembali bentuk dari sistem PDS berdimensi *N* dengan proses *Wiener* berdimensi *M*, yaitu

$$d\boldsymbol{X}(t) = A\boldsymbol{X}(t)dt + \sum_{k=1}^{M} B^k \boldsymbol{X}(t) dW_k(t)\ , \qquad \boldsymbol{X}(0) = \boldsymbol{X_0}$$

yang apabila dinyatakan dalam bentuk matriks adalah sebagai berikut:

$$d\begin{bmatrix} X_1(t) \\ X_2(t) \\ \vdots \\ X_N(t) \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,N} \\ a_{2,1} & a_{2,2} & & a_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N,1} & a_{N,2} & \cdots & a_{N,N} \end{bmatrix} \begin{bmatrix} X_1(t) \\ X_2(t) \\ \vdots \\ X_N(t) \end{bmatrix} dt + \begin{bmatrix} b_{1,1}^1 & b_{1,2}^1 & \cdots & b_{1,N}^1 \\ b_{2,1}^1 & b_{2,2}^1 & & b_{2,N}^1 \\ \vdots & \vdots & \ddots & \vdots \\ b_{N,1}^1 & b_{N,2}^1 & \cdots & b_{N,N}^1 \end{bmatrix}$$

$$\begin{bmatrix} X_1(t) \\ X_2(t) \\ \vdots \\ X_N(t) \end{bmatrix} dW_1(t) + \begin{bmatrix} b_{1,1}^2 & b_{1,2}^2 & \cdots & b_{1,N}^2 \\ b_{2,1}^2 & b_{2,2}^2 & & b_{2,N}^2 \\ \vdots & \vdots & \ddots & \vdots \\ b_{N,1}^2 & b_{N,2}^2 & \cdots & b_{N,N}^2 \end{bmatrix} \begin{bmatrix} X_1(t) \\ X_2(t) \\ \vdots \\ X_N(t) \end{bmatrix} dW_2(t) + \cdots +$$

$$\begin{bmatrix} b_{1,1}^M & b_{1,2}^M & \cdots & b_{1,N}^M \\ b_{2,1}^M & b_{2,2}^M & & b_{2,N}^M \\ \vdots & \vdots & \ddots & \vdots \\ b_{N,1}^M & b_{N,2}^M & \cdots & b_{N,N}^M \end{bmatrix} \begin{bmatrix} X_1(t) \\ X_2(t) \\ \vdots \\ X_N(t) \end{bmatrix} dW_M(t).$$

Sistem PDS dapat diselesaikan dengan metode EM sebagai berikut:

$$\boldsymbol{X}(t_j) = \boldsymbol{X}(t_{j-1}) + A\boldsymbol{X}(t_{j-1})\Delta t + \sum_{k=1}^{M} B^k \boldsymbol{X}(t_{j-1}) dW_k(t_j), \qquad \boldsymbol{X}(0) = \boldsymbol{X_0}$$

yang apabila dijabarkan menjadi:

$$\begin{bmatrix} X_1(t_j) \\ X_2(t_j) \\ \vdots \\ X_N(t_j) \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,N} \\ a_{2,1} & a_{2,2} & & a_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N,1} & a_{N,2} & \cdots & a_{N,N} \end{bmatrix} \begin{bmatrix} X_1(t_{j-1}) \\ X_2(t_{j-1}) \\ \vdots \\ X_N(t_{j-1}) \end{bmatrix} \Delta t + \begin{bmatrix} b_{1,1}^1 & b_{1,2}^1 & \cdots & b_{1,N}^1 \\ b_{2,1}^1 & b_{2,2}^1 & & b_{2,N}^1 \\ \vdots & \vdots & \ddots & \vdots \\ b_{N,1}^1 & b_{N,2}^1 & \cdots & b_{N,N}^1 \end{bmatrix}$$

$$\begin{bmatrix} X_1(t_{j-1}) \\ X_2(t_{j-1}) \\ \vdots \\ X_N(t_{j-1}) \end{bmatrix} dW_1(t_j) + \begin{bmatrix} b_{1,1}^2 & b_{1,2}^2 & \cdots & b_{1,N}^2 \\ b_{2,1}^2 & b_{2,2}^2 & & b_{2,N}^2 \\ \vdots & \vdots & \ddots & \vdots \\ b_{N,1}^2 & b_{N,2}^2 & \cdots & b_{N,N}^2 \end{bmatrix} \begin{bmatrix} X_1(t_{j-1}) \\ X_2(t_{j-1}) \\ \vdots \\ X_N(t_{j-1}) \end{bmatrix} dW_2(t_j)$$

$$+ \cdots +$$

$$\begin{bmatrix} b_{1,1}^{M} & b_{1,2}^{M} & \cdots & b_{1,N}^{M} \\ b_{2,1}^{M} & b_{2,2}^{M} & & b_{2,N}^{M} \\ \vdots & \vdots & \ddots & \vdots \\ b_{N,1}^{M} & b_{N,2}^{M} & \cdots & b_{N,N}^{M} \end{bmatrix} \begin{bmatrix} X_1(t_{j-1}) \\ X_2(t_{j-1}) \\ \vdots \\ X_N(t_{j-1}) \end{bmatrix} dW_M(t_j)$$

dimana $\Delta t = T/L$ dan $L$ bilangan bulat positif, $t_j = j\Delta t$, $dW_k(t_j) = W_k(t_j) - W_k(t_{j-1})$, dan $j = 1,2,\dots,L$. Perhatikan bahwa untuk menyelesaikan $X_1(t_j)$ diperlukan nilai $X_1(t_{j-1}), X_2(t_{j-1}), \dots, X_N(t_{j-1})$. Begitu pula untuk menyelesaikan $X_2(t_j)$ diperlukan nilai $X_1(t_{j-1}), X_2(t_{j-1}), \dots, X_N(t_{j-1})$. Secara umum untuk menyelesaikan $X_i(t_j)$ dimana $i = 1,2,\dots,N$ diperlukan nilai $X_1(t_{j-1}), X_2(t_{j-1}), \dots, X_N(t_{j-1})$. Jadi, pada kasus ini memiliki *data dependence* sehingga proses penghitungan $X_1(t_j), X_2(t_j), \dots, X_N(t_j)$ tidak dapat dikerjakan secara bersamaan. Dengan demikian dapat disimpulkan bahwa jika matriks $A$ dan $B^k$ merupakan matriks penuh, maka algoritma paralel tidak efektif diterapkan pada sistem PDS.

Dalam berbagai bidang aplikasi yang melibatkan sistem PDS, seringkali dijumpai matriks suku deterministik $A$ dan matriks suku stokastik $B^k$ merupakan matriks penuh sehingga algoritma paralel tidak efektif untuk diterapkan. Akan tetapi, menurut Bai dan Ward (2006), dengan menggunakan transformasi *Householder*, maka suatu matriks penuh dapat ditransformasi ke bentuk matriks tridiagonal. Oleh karena itu, matriks penuh $A$ dan $B^k$ yang terdapat pada sistem PDS dapat ditransformasi ke bentuk matriks tridiagonal. Pada skripsi ini, transformasi *Householder* tidak dibahas, tetapi diasumsikan bahwa telah diterapkan transformasi *Householder* pada matriks penuh $A$ dan $B^k$ sehingga $A$ dan $B^k$ merupakan matriks tridiagonal.

Berikut ini akan dibangun algoritma paralel untuk sistem PDS dimana matriks suku deterministik $A$ dan matriks suku stokastik $B^k$ telah ditransformasi menjadi matriks tridiagonal. Pertama-tama perhatikan sistem PDS pada kasus ini, yaitu:

$$d\boldsymbol{X}(t) = A\boldsymbol{X}(t)dt + \sum_{k=1}^{M} B^k \boldsymbol{X}(t) dW_k(t) \ , \qquad \boldsymbol{X}(0) = \boldsymbol{X_0}$$

dimana $\boldsymbol{X} = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_N \end{bmatrix}, \boldsymbol{X_0} = \begin{bmatrix} X_{1,0} \\ X_{2,0} \\ \vdots \\ X_{N,0} \end{bmatrix}$

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & & & & & & \\ a_{2,1} & a_{2,2} & a_{2,3} & & & & 0 & \\ & a_{3,2} & a_{3,3} & a_{3,4} & & & & \\ & & a_{4,3} & a_{4,4} & a_{4,5} & & & \\ & & & & \ddots & & & \\ & 0 & & & & a_{N-1,N-2} & a_{N-1,N-1} & a_{N-1,N} \\ & & & & & & a_{N,N-1} & a_{N,N} \end{bmatrix}$$

dan

$$B^k = \begin{bmatrix} b_{1,1}^k & b_{1,2}^k & & & & & & \\ b_{2,1}^k & b_{2,2}^k & b_{2,3}^k & & & & 0 & \\ & b_{3,2}^k & b_{3,3}^k & b_{3,4}^k & & & & \\ & & b_{4,3}^k & b_{4,4}^k & b_{4,5}^k & & & \\ & & & & \ddots & & & \\ & 0 & & & & b_{N-1,N-2}^k & b_{N-1,N-1}^k & b_{N-1,N}^k \\ & & & & & & b_{N,N-1}^k & b_{N,N}^k \end{bmatrix}$$

Untuk memudahkan penghitungan, maka $N = 2^n$ dimana $n$ adalah bilangan asli.

Sistem PDS diatas dapat diselesaikan dengan metode EM sebagai berikut

$$\boldsymbol{X}(t_j) = \boldsymbol{X}(t_{j-1}) + A\boldsymbol{X}(t_{j-1})\Delta t + \sum_{k=1}^{M} B^k \boldsymbol{X}(t_{j-1}) dW_k(t_j), \quad \boldsymbol{X}(0) = \boldsymbol{X_0}$$

yang apabila dijabarkan menjadi

$$\begin{bmatrix} X_1(t_j) \\ X_2(t_j) \\ \vdots \\ X_N(t_j) \end{bmatrix} = A \begin{bmatrix} X_1(t_{j-1}) \\ X_2(t_{j-1}) \\ \vdots \\ X_N(t_{j-1}) \end{bmatrix} dt + B^1 \begin{bmatrix} X_1(t_{j-1}) \\ X_2(t_{j-1}) \\ \vdots \\ X_N(t_{j-1}) \end{bmatrix} dW_1(t_j) + \cdots +$$

$$B^M \begin{bmatrix} X_1(t_{j-1}) \\ X_2(t_{j-1}) \\ \vdots \\ X_N(t_{j-1}) \end{bmatrix} dW_M(t_j).$$

Perhatikan bahwa:

1. Untuk menyelesaikan $X_1(t_j)$ diperlukan nilai $X_1(t_{j-1})$ dan $X_2(t_{j-1})$.
2. Untuk menyelesaikan $X_2(t_j)$ diperlukan nilai $X_1(t_{j-1})$, $X_2(t_{j-1})$, dan $X_3(t_{j-1})$.

   Untuk menyelesaikan $X_3(t_j)$ diperlukan nilai $X_2(t_{j-1})$, $X_3(t_{j-1})$, dan $X_4(t_{j-1})$.

   ⋮

   Untuk menyelesaikan $X_{N-1}(t_j)$ diperlukan nilai $X_{N-2}(t_{j-1})$, $X_{N-1}(t_{j-1})$, dan $X_N(t_{j-1})$.
3. Untuk menyelesaikan $X_N(t_j)$ diperlukan nilai $X_{N-1}(t_{j-1})$, dan $X_N(t_{j-1})$

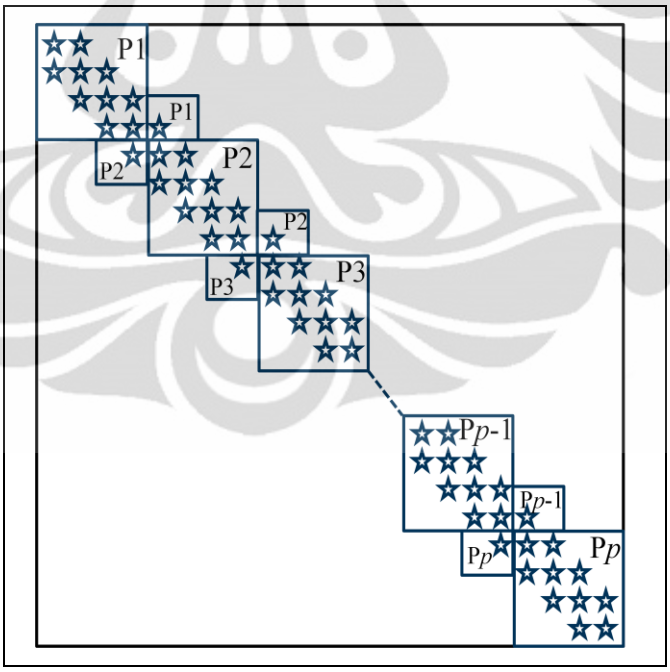
Secara umum, untuk menyelesaikan $X_i(t_j)$ diperlukan suku $X_{i-1}(t_{j-1})$, $X_i(t_{j-1})$, dan $X_{i+1}(t_{j-1})$ untuk $i \neq 1$ dan $i \neq N$. Misalkan tersedia $p$ prosesor untuk menyelesaikan masalah ini dimana $p = 2^n$ dimana $n$ adalah bilangan asli dan $p$ habis membagi $N$. Setiap prosesor $i$ dimana $i = 1,2, \dots, p$ ditugaskan untuk menghitung $X_{(i-1)*N/p+1}(t_j)$ hingga $X_{i*N/p}(t_j)$ dimana $j = 1,2, \dots, L$. Ide paralel pada kasus ini adalah sebagai berikut:

1. Elemen baris ke-$((i-1)*N/p+1)$ hingga baris ke-$(i*N/p)$ dan kolom ke-$((i-1)*N/p+1)$ hingga kolom ke-$(i*N/p)$ dari matriks $A$ dan $B^k$ dikirimkan ke prosesor $i$ dimana $i = 1, \dots, p$.
2. Berdasarkan elemen-elemen matriks-matriks tersebut, setiap prosesor $i$ menghitung $X_{(i-1)*N/p+1}(t_j)$ hingga $X_{i*N/p}(t_j)$. Namun, masalah muncul pada prosesor $i$ dimana $i = 1, \dots, p-1$ ketika menghitung $X_{i*N/p}(t_j)$ karena membutuhkan $X_{i*N/p-1}(t_{j-1})$, $X_{i*N/p}(t_{j-1})$, dan $X_{i*N/p+1}(t_{j-1})$. Sedangkan $X_{i*N/p+1}(t_{j-1})$ dikerjakan oleh prosesor $(i+1)$. Untuk mengatasi masalah tersebut, maka elemen baris ke-$(i*N/p)$ dan kolom ke-$(i*N/p+1)$ dari matriks $A$ dan $B^k$ dikirimkan ke prosesor $i$ dimana $i = 1, \dots, p-1$. Pada setiap titik diskretisasi, prosesor $(i+1)$

mengirimkan nilai $X_{i*N/p+1}(t_{j-1})$ ke prosesor $i$ dimana $i = 1, \dots, p-1$ sehingga prosesor $i$ dapat menghitung $X_{i*N/p}(t_j)$.

3. Masalah juga muncul pada prosesor $i$ dimana $i = 2, \dots, p$ ketika menghitung $X_{(i-1)*N/p+1}(t_j)$ karena membutuhkan $X_{(i-1)*N/p}(t_{j-1})$, $X_{(i-1)*N/p+1}(t_{j-1})$, dan $X_{(i-1)*N/p+2}(t_{j-1})$. Sedangkan $X_{(i-1)*N/p}(t_{j-1})$ dikerjakan oleh prosesor $(i-1)$. Untuk mengatasi masalah tersebut, maka elemen baris ke-$((i-1)*N/p+1)$ dan kolom ke-$((i-1)*N/p)$ dari matriks $A$ dan $B^k$ dikirimkan ke prosesor $i$ dimana $i = 2, \dots, p$. Pada setiap titik diskretisasi, prosesor $(i-1)$ mengirimkan nilai $X_{(i-1)*N/p}(t_{j-1})$ ke prosesor $i$ dimana $i = 2, \dots, p$ sehingga prosesor $i$ dapat menghitung $X_{(i-1)*N/p+1}(t_j)$.

Berdasarkan ide paralel yang telah dibahas, maka distribusi matriks $A$ dan $B^k$ pada $p$ prosesor adalah seperti pada Gambar 3.6.




Gambar 3.6 Distribusi matriks tridiagonal $A$ dan $B^k$ pada $p$ prosesor

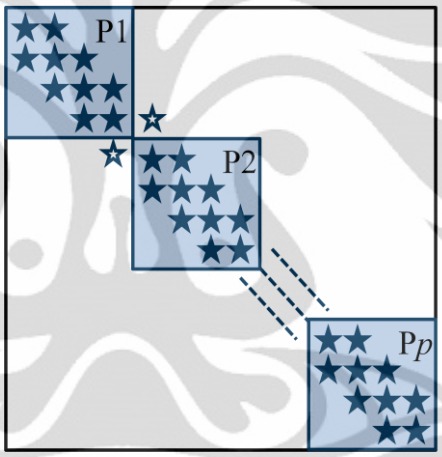
Berikut ini akan dibangun algoritma paralel untuk sistem PDS. Algoritma paralel untuk sistem PDS akan dibagi menjadi dua tahap paralel, yaitu:

1. Tahap Paralel 1

   Pada tahap ini, setiap prosesor $i$ dimana $i = 1,2,\dots,p$ menghitung $X_{(i-1)*N/p+1}(t_j)$ hingga $X_{i*N/p}(t_j)$ berdasarkan elemen baris ke-$((i-1)*N/p+1)$ hingga baris ke-$(i*N/p)$ dan kolom ke-$((i-1)*N/p+1)$ hingga kolom ke-$(i*N/p)$ dari matriks $A$ dan $B^k$. Jadi, setelah tahap paralel 1 dilakukan, maka didapat $\boldsymbol{X}(t_j)$, yaitu

$$\boldsymbol{X}(t_j) = \boldsymbol{X}(t_{j-1}) + A\boldsymbol{X}(t_{j-1})\Delta t + \sum_{k=1}^{M} B^k \boldsymbol{X}(t_{j-1}) dW_k(t_j), \qquad \boldsymbol{X}(0) = \boldsymbol{X_0}$$

   dimana matriks $A$ dan $B^k$ adalah matriks blok diagonal dengan banyaknya blok diagonal adalah $p$ dan ukuran blok diagonal sebesar $N/p$ (lihat Gambar 3.7).




Gambar 3.7 Distribusi matriks $A$ dan $B^k$ pada tahap paralel 1

2. Tahap Paralel 2

   Pada tahap ini, prosesor $i$ dimana $i = 1,\dots,p-1$ meng-*update* nilai $X_{i*N/p}(t_j)$ karena pada tahap paralel 1 $X_{i*N/p}(t_j)$ hanya dihitung berdasarkan $X_{i*N/p-1}(t_{j-1})$ dan $X_{i*N/p}(t_{j-1})$, sedangkan seharusnya untuk menghitung $X_{i*N/p}(t_{j-1})$ diperlukan nilai $X_{i*N/p-1}(t_{j-1})$, $X_{i*N/p}(t_{j-1})$, dan $X_{i*N/p+1}(t_{j-1})$. Nilai $X_{i*N/p+1}(t_{j-1})$ dikerjakan oleh prosesor $(i+1)$. Oleh karena itu, pada setiap titik diskretisasi $t_j$ prosesor $(i+1)$ mengirimkan nilai $X_{i*N/p+1}(t_{j-1})$ ke prosesor $i$ dimana $i = 1,\dots,p-1$ sehingga prosesor $i$ dapat meng-*update* $X_{i*N/p}(t_j)$ yang telah diperoleh pada tahap paralel 1, yaitu

$$X_{i*N/p}(t_j) = X_{i*N/p}(t_j) + A_{(i*N/p),(i*N/p+1)} * X_{i*N/p+1}(t_{j-1}) + \sum_{k=1}^{M} B^k_{(i*N/p),(i*N/p+1)} * X_{i*N/p+1}(t_{j-1})$$

dimana $i = 1, \dots, p-1$.

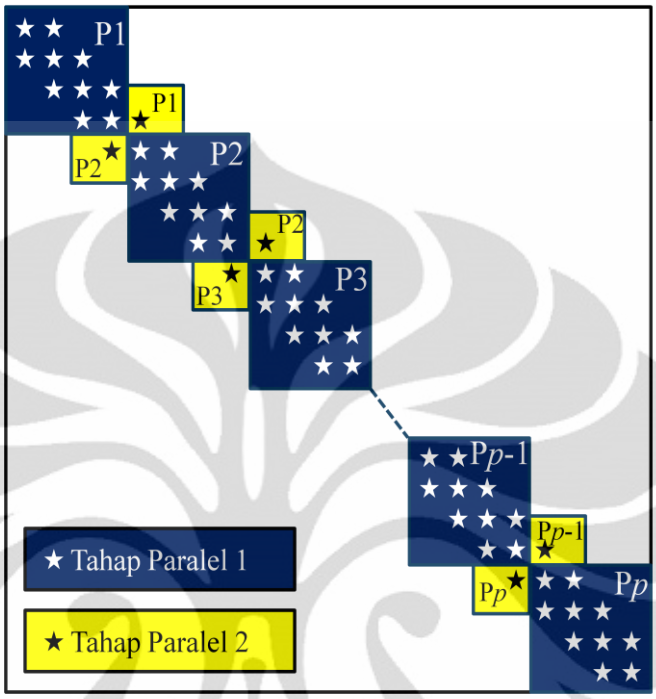
Selain itu, pada tahap paralel 2 ini prosesor $i$ dimana $i = 2, \dots, p$ juga meng-*update* nilai $X_{(i-1)*N/p+1}(t_j)$ karena pada tahap paralel 1 $X_{(i-1)*N/p+1}(t_j)$ hanya dihitung berdasarkan $X_{(i-1)*N/p+1}(t_{j-1})$ dan $X_{(i-1)*N/p+2}(t_{j-1})$, sedangkan seharusnya untuk menghitung $X_{(i-1)*N/p+1}(t_j)$ diperlukan nilai $X_{(i-1)*N/p}(t_{j-1})$, $X_{(i-1)*N/p+1}(t_{j-1})$, dan $X_{(i-1)*N/p+2}(t_{j-1})$. Nilai $X_{(i-1)*N/p}(t_{j-1})$ dikerjakan oleh prosesor $(i-1)$. Oleh karena itu, pada setiap titik diskretisasi $t_j$ prosesor $(i-1)$ mengirimkan nilai $X_{(i-1)*N/p}(t_{j-1})$ ke prosesor $i$ dimana $i = 2, \dots, p$ sehingga prosesor $i$ dapat meng-*update* $X_{(i-1)*N/p+1}(t_j)$ yang telah diperoleh pada tahap paralel 1, yaitu

$$X_{(i-1)*N/p+1}(t_j) = X_{(i-1)*N/p+1}(t_j) + A_{((i-1)*N/p+1),((i-1)*N/p)} * X_{(i-1)*N/p}(t_{j-1}) + \sum_{k=1}^{M} B^k_{((i-1)*N/p+1),((i-1)*N/p)} * X_{(i-1)*N/p}(t_{j-1})$$

dimana $i = 2, \dots, p$. Pada tahap paralel 2, setiap prosesor $i$ dimana $i = 1,2, \dots, p$ melakukan tugasnya secara bersamaan.

Jadi, distribusi matriks $A$ dan $B^k$ yang digunakan untuk tahap paralel 1 dan tahap paralel 2 adalah seperti pada Gambar 3.8. Berikut ini adalah Algoritma 3.4 yaitu algoritma paralel untuk sistem PDS pada kasus matriks $A$ dan $B^k$ adalah matriks tridiagonal (lihat Tabel 3.4). Algoritma 3.4 bertujuan untuk mengaproksimasi solusi dari sistem PDS pada persamaan (3.5) dengan metode Euler-Maruyama pada persamaan (3.6) dengan asumsi bahwa matriks $A$ dan $B^k$ telah ditransformasi menjadi matriks tridiagonal dengan transformasi *Householder*. Input dari algoritma ini adalah banyaknya PDS pada sistem yaitu $N = 2^n$, banyaknya proses *Wiener* yaitu $M, T$ yang merupakan batas atas interval $[0, T]$, nilai awal dari masing-masing PDS yaitu $X1zero, X2zero, \dots, XNzero$, banyaknya titik diskretisasi yaitu $L$, $A$ yang merupakan matriks koefisien *drift*, $B^k$

yang merupakan koefisien *diffusion*, dan $p = 2^n$ yang merupakan banyaknya prosesor dimana $p|N$.




Gambar 3.8 Distribusi matriks $A$ dan $B^k$ pada tahap paralel 1 dan tahap paralel 2

Langkah pertama pada Algoritma 3.4 adalah menghitung $\Delta t = T/L$. Titik diskretisasi $t_1$ adalah $\Delta t$, titik diskretisasi $t_2$ adalah $2 * \Delta t$, atau secara umum titik diskretisasi $t_j$ adalah $j * \Delta t$ dimana $j = 1,2, \dots, L$. Selanjutnya dihitung $dW_k(t_j) = W_k(t_j) - W_k(t_{j-1})$ dimana $k = 1, \dots, M,\ j = 1, \dots, L$. $dW\_1, dW\_2, \dots, dW\_N$ merupakan array berukuran $1\ x\ L$ berisi bilangan random berdistribusi $\sqrt{\Delta t}\ N(0,1)$. Dalam hal ini $dW\_1_j$ adalah entri ke-$j$ dari $dW\_1$ adalah nilai dari $dW_1(t_j) = W_1(t_j) - W_1(t_{j-1})$ dimana $j = 1, \dots, L$. $dW\_2_j$ adalah entri ke-$j$ dari $dW\_2$ adalah nilai dari $dW_2(t_j) = W_2(t_j) - W_2(t_{j-1})$ dimana $j = 1, \dots, L$. Demikian seterusnya hingga $dW\_M_j$ adalah entri ke-$j$ adalah entri ke-$j$ dari $dW\_M$ adalah nilai dari $dW_M(t_j) = W_M(t_j) - W_M(t_{j-1})$ dimana $j = 1, \dots, L$.

Pada langkah 3, inisialisasi $Xsys$ sebagai matriks berukuran $N\ x\ (L + 1)$ yang semua entrinya adalah nol. Baris ke-$i$ dari $Xsys$ dimana $i = 1, \dots, N$ akan digunakan untuk menyimpan solusi EM dari PDS ke-$i$ dalam sistem yaitu $X_i$.

Pada langkah 4, $Xsys_{:,1}$ adalah kolom ke-1 dari $Xsys$ berisi nilai awal untuk masing-masing PDS yaitu $[X1zero; X2zero; \dots; XNzero]$.

Tabel 3.4 Algoritma paralel untuk sistem PDS dimana matriks suku deterministik $A$ dan matriks suku stokastik $B^k$ telah ditransformasi menjadi matriks tridiagonal

**Algoritma 3.4 Euler-Maruyama method for SDE system in parallel where $A$ and $B^k$ are tridiagonal matrix**

INPUT : $N = 2^n$ is a number of SDEs, $M$ is a number of *Wiener* process, $T$ is upper bound of interval $[0, T]$, $X1zero, X2zero, \dots, XNzero$ is initial value of SDEs respectively, $L$ is number of discretized point, $A$ is drift coefficient matrix and $B^k$ for $k = 1,2, \dots, M$ is diffusion coefficient matrix where $A$ and $B^k$ are tridiagonal matrix respectively, $p$ is a number of processors where $p|N$.

OUTPUT : $Xsys$ is an $N$-by-$L$ where $i$th row is the Euler-Maruyama solution of $X_i$ to the SDE system:

$$d\boldsymbol{X}(t) = A\boldsymbol{X}(t)dt + \sum_{k=1}^{M} B^k \boldsymbol{X}(t) dW_k(t) \ , \boldsymbol{X}(0) = \boldsymbol{X_0}, 0 \le t \le T$$

1 $\Delta t := T/L$

2 **set** $dW\_1$ is a 1-by-$L$ array of random number from normally distributed $\sqrt{\Delta t}\, N(0,1)$

$dW\_2$ is a 1-by-$L$ array of random number from normally distributed $\sqrt{\Delta t}\, N(0,1)$

$\vdots$

$dW\_M$ is a 1-by-$L$ array of random number from normally distributed $\sqrt{\Delta t}\, N(0,1)$

3 **set** $Xsys$ is an $N$-by-$(L+1)$ zeros array

4 **set** $Xsys_{:,1} := \begin{bmatrix} X1zero \\ X2zero \\ \vdots \\ XNzero \end{bmatrix}$

5 Each processor $i$ where $i = 1,2, \dots, p$ **do in parallel**

6 **set** $A1 := 0$

7 **set** $A2 := 0$

8 **set** $B1$ is an 1-by-$M$ zeros array

9 **set** $B2$ is an 1-by-$M$ zeros array

10 **if** $i \neq p$ processor $i$ **do**

11 $A1 := A_{i*N/p,i*N/p+1} * Xsys_{i*N/p+1,1}$

12 **for** $k$ = 1 to $M$ **do**

13 $B1_{1,k} := B^{k}_{i*N/p,i*N/p+1} * Xsys_{i*N/p+1,1}$

**end for**

**end if**

14 **if** $i \neq 1$ processor $i$ **do**

15 $A2 := A_{(i-1)*N/p+1,(i-1)*N/p} * Xsys_{(i-1)*N/p,1}$

16 **for** $k$ = 1 to $M$ **do**

17 $B2_{1,k} := B^{k}_{(i-1)*N/p+1,(i-1)*N/p} * Xsys_{(i-1)*N/p,1}$

**end for**

**end if**

18 **set** $Xsys\_local$ is an $(N/p)$-by-$(L+1)$ zeros array

19 **set** $A\_local := A_{(i-1)*N/p+1:i*N/p,(i-1)*N/p+1:i*N/p}$

20 **set** $Xsys\_local_{:,1} := Xsys_{(i-1)*N/p+1:i*N/p,1}$

21 **set** $Btemp$ is an $(N/p)$-by-$M$ zeros array

22 **for** $j = 1$ to $L$ **do**

23 **for** $k = 1$ to $M$ **do**

24 **set** $Btemp_{:,m} := B^{k}_{(i-1)*N/p+1:i*N/p,(i-1)*N/p+1:i*N/p} * Xsys\_local_{:,j}$

**end for**

25 $Atemp := A\_local * Xsys\_local_{:,j}$

26 $Atemp_{N/p} := Atemp_{N/p} + X1$

27 $Atemp_{1} := Atemp_{1} + X2$

28 $Btemp_{N/p,:} := Btemp_{N/p,:} + Y1$

29 $Btemp_{1,:} := Btemp_{1,:} + Y2$

30 $Xsys\_local_{:,j+1} := Xsys\_local_{:,j} + Atemp * \Delta t + Btemp * \begin{bmatrix} dW\_1_j \\ dW\_2_j \\ \vdots \\ dW\_M_j \end{bmatrix}$

31 **if** $i \neq p$ processor $i$ **do**

32 **labSend** $Xsys\_local_{N/p,j+1}$ **to** prosesor $(i+1)$

33 **labReceive** $Xsys\_local_{1,j+1}$ **from** prosesor $(i+1)$

34 $A1 := A_{i*N/p,i*N/p+1} * Xsys\_local_{1,j+1}$

35 **for** $k$ = 1 to $M$ **do**

36 $B1_{1,k} := B^k_{i*N/p,i*N/p+1} * Xsys\_local_{1,j+1}$

**end for**

**end if**

37 **if** $i \neq 1$ processor $i$ **do**

38 **labSend** $Xsys\_local_{1,j+1}$ **to** prosesor $(i-1)$

39 **labReceive** $Xsys\_local_{N/p,j+1}$ **from** prosesor $(i-1)$

40 $A2 := A_{(i-1)*N/p+1,(i-1)*N/p} * Xsys\_local_{N/p,j+1}$

41 **for** $k$ = 1 to $M$ **do**

42 $B2_{1,k} := B^k_{(i-1)*N/p+1,(i-1)*N/p} * Xsys\_local_{N/p,j+1}$

**end for**

**end if**

**end for**

**end do**

Langkah selanjutnya merupakan proses paralel yang dimulai pada langkah 5 dengan menggunakan $p$ prosesor. Sebelum menghitung solusi EM dari masing-masing PDS pada tiap titik diskretisasi $t_j$ dimana $j = 1, \dots, L$, maka dilakukan inisialisasi $A1$, $A2$, $B1$, $B2$ terlebih dahulu. Langkah 10 hingga 13 dilakukan oleh prosesor $i$ dimana $i = 1, \dots, p-1$. Pada langkah ini dihitung $A1 = A_{i*N/p,i*N/p+1} * Xsys_{i*N/p+1,1}$ yang merupakan bagian dari proses tahap paralel 2 yaitu menghitung $A_{(i*N/p),(i*N/p+1)} * X_{i*N/p+1}(t_{j-1})$ pada $t_1$. Sedangkan $B1_{1,k} = B^k_{i*N/p,i*N/p+1} * Xsys_{i*N/p+1,1}$ untuk $k = 1, \dots, M$ merupakan bagian dari

proses tahap paralel 2 yaitu menghitung $\sum_{k=1}^{M} B^{k}_{(i*N/p),(i*N/p+1)} * X_{i*N/p+1}(t_{j-1})$ pada $t_1$. Sedangkan langkah 14 hingga 17 dilakukan oleh prosesor $i$ dimana $i = 2, \dots, p$. Pada langkah ini dihitung $A2 = A_{(i-1)*N/p+1,(i-1)*N/p} * Xsys_{(i-1)*N/p,1}$ yang merupakan bagian dari proses tahap paralel 2 yaitu menghitung $A_{((i-1)*N/p+1),((i-1)*N/p)} * X_{(i-1)*N/p}(t_{j-1})$ pada $t_1$. Sedangkan $B2_{1,k} = B^{k}_{(i-1)*N/p+1,(i-1)*N/p} * Xsys_{(i-1)*N/p,1}$ untuk $k = 1, \dots, M$ merupakan bagian dari proses tahap paralel 2 yaitu menghitung $\sum_{k=1}^{M} B^{k}_{((i-1)*N/p+1),((i-1)*N/p)} * X_{(i-1)*N/p}(t_{j-1})$ pada $t_1$.

Proses berikutnya yaitu langkah 18 hingga 24 merupakan tahap paralel 1 dalam menghitung solusi EM dari masing-masing PDS pada tiap titik diskretisasi $t_j$ dimana $j = 1, \dots, L$. $Xsys\_local$ adalah matriks berukuran $(N/p)\ x\ (L+1)$ yang akan digunakan untuk menyimpan solusi EM untuk sistem PDS pada tahap paralel 1 untuk tiap prosesor. Setiap prosesor $i$ menghitung $X_{(i-1)*N/p+1}(t_j)$ hingga $X_{i*N/p}(t_j)$ dan disimpan pada $Xsys\_local$. $A\_local$ adalah matriks yang berisi elemen baris ke-$((i-1)*N/p+1)$ hingga baris ke-$(i*N/p)$ dan kolom ke-$((i-1)*N/p+1)$ hingga kolom ke-$(i*N/p)$ dari matriks $A$. Langkah 23 dan 24 merupakan proses penghitungan $\sum_{k=1}^{M} B^{k}\boldsymbol{X}(t_{j-1})$ pada tahap paralel 1. Hasil dari penghitungan ini disimpan pada $Btemp$. Langkah 25 merupakan proses penghitungan $A\boldsymbol{X}(t_{j-1})$ pada tahap paralel 1. Hasil dari penghitungan ini disimpan pada $Atemp$. Langkah 26 hingga 30 merupakan sinkronisasi dari proses penghitungan tahap paralel 1 dan paralel 2 sehingga didapat nilai $\boldsymbol{X}(t_j)$ sesuai yang diharapkan.

Langkah 31 hingga 36 merupakan tahap paralel 2 yang dikerjakan oleh prosesor $i$ dimana $i = 1, \dots, p-1$. Pada langkah ini, prosesor $i$ mengirimkan $Xsys\_local_{N/p,j+1}$ yang telah didapat ke prosesor $(i+1)$ untuk digunakan dalam meng-*update* nilai $A2$ dan $B2$. Pada saat yang sama, prosesor $i$ dimana $i = 1, \dots, p-1$ menerima $Xsys\_local_{1,j+1}$ dari prosesor $(i+1)$ dan kemudian meng-*update* nilai $A1$ dan $B1$. Sedangkan Langkah 37 hingga 42 merupakan tahap paralel 2 yang dikerjakan oleh prosesor $i$ dimana $i = 2, \dots, p$. Pada langkah ini, prosesor $i$ mengirimkan $Xsys\_local_{1,j+1}$ yang telah didapat ke prosesor $(i-1)$

untuk digunakan dalam meng-*update* nilai $A1$ dan $B1$. Pada saat yang sama, prosesor $i$ dimana $i = 2, \dots, p$ menerima $Xsys\_local_{N/p,j+1}$ dari prosesor $(i-1)$ dan kemudian meng-*update* nilai $A2$ dan $B2$.

Dengan menerapkan Algoritma 3.4, diharapkan akan mempercepat waktu komputasi dalam mengaproksimasi solusi sistem PDS dengan metode EM. Implementasi dari algoritma ini akan diberikan pada Bab 4.

## 3.5 Algoritma Paralel untuk Suatu Model PDS pada Bidang Keuangan

Model-model Persamaan Diferensial Stokastik (PDS) memiliki peranan yang sangat penting di berbagai bidang diantaranya adalah bidang keuangan. Salah satu model PDS pada bidang keuangan yang akan dibahas pada skripsi ini adalah model harga opsi *call* Asia dengan volatilitas stokastik. Menurut Kanniainen dan Piche (2009), harga *underlying asset* $U$ di bawah kondisi *martingale* (lihat (Elliot & Madan, 1998)) mengikuti persamaan diferensial stokastik

$$d\ln U(t) = \left(r - \frac{1}{2}v(t)\right)dt + \sqrt{v(t)}dW(t)\,, \quad U(0) = U_0 > 0 \tag{3.7}$$

dimana $r$ adalah suku bunga bebas resiko, $v$ adalah *return variance* yang bersifat stokastik, dan $W$ adalah proses *Wiener*. *Return variance* diasumsikan sebagai proses *mean-reverting* (lihat (Fouque, Papanicolaou, & Sircar, 2000))

$$dv(t) = \kappa\big(\bar{v} - v(t)\big)dt + \psi\sqrt{v(t)}dW_v(t)\,, \quad v(0) = v_0 > 0 \tag{3.8}$$

dimana $\bar{v}$ adalah *reference variance*, $\kappa$ adalah kecepatan pengembalian, $\psi$ adalah volatilitas, dan $dW_v$ adalah gerak *Brown* standar dimana $dW\,dW_v = \rho dt$ dan $\rho \in [-1,1]$. Harga opsi *call* Asia (lihat (Kanniainen & Piche, 2009)) adalah

$$e^{-r(T-t)}\mathbb{E}_t[\overline{U} - E]^+ \tag{3.9}$$

dimana $t_0 < t$ adalah waktu kontrak opsi, $T > t$ adalah waktu jatuh tempo opsi, $E$ adalah *the strike price*, $\mathbb{E}_t$ adalah ekspektasi bersyarat pada saat $t$ di bawah kondisi *martingale*, dan

$$\overline{U} = \frac{1}{L}\sum_{i=1}^{L} U(t_i)$$

(3.10)

dimana $t_0 \le t_1 < t_2 < \cdots < t_L \le T$, $\bar{U}$ adalah harga *arithmatic-average* dari *underlying stock*. Solusi EM dari PDS pada persamaan (3.7) adalah

$$\ln U(t_{i+1}) = \ln U(t_i) + \left(r - \frac{1}{2}v(t_i)\right)\Delta t + \sqrt{v(t_i)\Delta t} * W(t_i) \qquad (3.11)$$

dimana barisan $\{W(t_i)\}$ adalah bilangan random berdistribusi $N(0,1)$. Sedangkan solusi EM dari PDS pada persamaan (3.8) adalah

$$v(t_{i+1}) = v(t_i) + \kappa\big(\bar{v} - v(t_i)\big)\Delta t + \psi\sqrt{v(t_i)\Delta t} * W_v(t_i) \qquad (3.12)$$

dimana $W_v(t_i) = \rho * W(t_i) + \sqrt{1-\rho^2} * Y(t_i)$ dimana barisan $\{Y(t_i)\}$ adalah bilangan random berdistribusi $N(0,1)$ yang tidak berkorelasi dengan $\{W(t_i)\}$ (Kanniainen & Piche, 2009). Jika dilakukan simulasi sebanyak $H$ *sample path* dari *underlying stock* dengan menggunakan (3.11) dan *return variance* dengan menggunakan (3.12), maka harga opsi pada saat $t$ dihitung berdasarkan persamaan (3.9) (lihat (Kanniainen & Piche, 2009)) yaitu

$$\frac{e^{-r(T-t)}}{H}\sum_{k=1}^{H}\left[\frac{1}{L}\sum_{i=1}^{L}U(t_i)^{(k)} - E\right]^{+} \qquad (3.13)$$

Jika hanya digunakan satu prosesor untuk menghitung harga opsi *call* Asia dengan melakukan simulasi sebanyak $10^6$ *sample path*, maka dibutuhkan waktu sedikitnya 7 jam (Kanniainen & Piche, 2009). Oleh karena itu, diperlukan komputasi paralel untuk mempercepat proses komputasi dalam mendapatkan harga opsi.

Pada skripsi ini, akan dibangun algoritma paralel untuk menghitung $H$ *sample path* untuk mendapatkan harga opsi *call* Asia. Algoritma paralel secara *sample path* untuk PDS yang telah dibahas pada Subbab 3.2 dapat diterapkan pada model ini. Berikut ini adalah algoritma paralel secara *sample path* untuk menghitung $H$ *sample path* untuk mendapatkan harga opsi *call* Asia. Pada algoritma ini akan digunakan *p* prosesor. Input dari algoritma ini adalah banyaknya simulasi yaitu $H$, $r$ adalah suku bunga bebas resiko, $\kappa$ adalah kecepatan pengembalian, $v$ adalah *return variance*, $T$ adalah batas atas interval

$[0, T]$, $t_L = T$ adalah waktu jatuh tempo opsi, $\bar{v}$ adalah *reference variance*, $\psi$ adalah volatilitas, dan banyaknya prosesor yaitu $p$ dimana $H \geq p$.

Tabel 3.5 Algoritma paralel secara *sample path* untuk menghitung harga opsi pada model harga opsi *call* Asia

**Algoritma 3.5.1 Euler-Maruyama method for Asian *call* options in parallel across simulations**

INPUT : $H$ is a number of simulations, $r$ is the constant instantaneous risk-free interest rate, $T$ is upper bound of interval $[0, T]$, $t_L = T$ is maturity date of the option, $p$ is a number of processors where $H \geq p$, and $v$ is return variance which is assumed to be amean-reverting process

$$dv(t) = \kappa\big(\bar{v} - v(t)\big)dt + \psi\sqrt{v(t)}dW_v(t)\,, \;\; v(0) = v_0 > 0$$

where $\bar{v}$ is reference variance, $\kappa$ is speed of reversion, and $\psi$ is volatility of the volatility.

OUTPUT : $price$ is an arithmatic Asian *call* option price

1. **set** $\Delta t := T/L$
2. **set** $W$ is an $H$-by-$L$ array of random number from normally distributed $N(0,1)$
3. **set** $Y$ is an $H$-by-$L$ array of random number from normally distributed $N(0,1)$
4. **set** $Z$ is an $H$-by-$L$ array of random number from normally distributed $\rho * W + \sqrt{1-\rho^2} * Y$
5. **set** $U$ is an $H$-by-$(L+1)$ zeros array
6. Each processor $i$ where $i = 1,2, \dots, p$ **do in parallel**
7. **if** $mod(H,p) \neq 0$ **do**
8. Each processor $e$ where $e = 1,2, \dots, mod(H,p)$ **do in parallel**
9. **set** $X\_local$ is an $(\lfloor H/p \rfloor + 1)$-by-$(L+1)$ zeros array
10. **set** $W\_end$ is an 1-by-$L$ array whose entries are
11. $(p * \lfloor H/p \rfloor + e)$th row of $W$

12. **set** $Z\_end$ is an 1-by-$L$ array whose entries are
    $(p * \lfloor H/p \rfloor + e)$th row of $Z$
13. **set** $v := v_0$
14. **for** $j = 1$ to $L$ **do**
15. $X\_local_{\lfloor H/p \rfloor+1, j+1} := X\_local_{\lfloor H/p \rfloor+1, j} + (r - 0.5 * v) * \Delta t + sqrt(v * \Delta t) * W\_end_j$
16. $v := max(0, v + \kappa * (v_0 - v) * \Delta t + \psi * sqrt(v * \Delta t) * Z\_end_j)$
    **end for**
17. **set** $U_{p*\lfloor H/p \rfloor+e,:} := U_0 e^{X\_local_{\lfloor H/p \rfloor+1,:}}$
    **end do**
18. **set** $X\_local$ is an $\lfloor H/p \rfloor$-by-$(L+1)$ zeros array for processor $f$ where
    $f = mod(H, p) + 1, mod(H, p) + 2, \ldots, p$
19. **else**
20. **set** $X\_local$ is an $\lfloor H/p \rfloor$-by-$(L+1)$ zeros array for each processor
    **end if**
21. **set** $W\_local$ is an $\lfloor H/p \rfloor$-by-$L$ array whose entries are
    $((i-1) * \lfloor H/p \rfloor + 1)$th row until $(i * \lfloor H/p \rfloor)$th row of $W$
22. **set** $Z\_local$ is an $\lfloor H/p \rfloor$-by-$L$ array whose entries are
    $((i-1) * \lfloor H/p \rfloor + 1)$th row until $(i * \lfloor H/p \rfloor)$th row of $Z$
23. **for** $h = 1$ to $\lfloor H/p \rfloor$ **do**
24. **set** $v := v_0$
25. **for** $j = 1$ to $L$ **do**
26. $X\_local_{h,j+1} := X\_local_{h,j} + (r - 0.5 * v) * \Delta t + sqrt(v * \Delta t) * W\_local_{h,j}$
27. $v := max(0, v + \kappa * (v_0 - v) * \Delta t + \psi * sqrt(v * \Delta t) * Z\_local_{h,j})$
    **end for**
    **end for**
28. **set** $U_{(i-1)*\lfloor H/p \rfloor+1:i*\lfloor H/p \rfloor,:} := U_0 e^{X\_local_{1:\lfloor H/p \rfloor,:}}$
    **end do**
29. **set** $\overline{U} := sum(U, 2)/L$
30. **set** $price := e^{-r*T} * sum(max(\overline{U} - repmat(E, H, 1), 0))/H$

Langkah 1 adalah inisialisasi $\Delta t = T/L$. Kemudian inisialisasi $W$ dan $Y$ yang merupakan matriks berukuran $H\ x\ L$ bilangan random berdistribusi $N(0,1)$. Pada langkah 4 inisialisasi $Z$ yang merupakan matriks berukuran $H\ x\ L$ bilangan random berdistribusi $\rho * W + \sqrt{1-\rho^2} * Y$. Selanjutnya inisialisasi $U$ sebagai matriks berukuran $H$-by-$(L+1)$ yang semua entrinya adalah nol. Matriks $U$ disimpan pada *shared memory* dan digunakan untuk menyimpan nilai harga *underlying stock* $U$ dimana baris ke-$h$ kolom ke-$l$ adalah nilai $U(t_{l-1})$ dari s*ample path* ke-$h$.

Langkah selanjutnya merupakan proses paralel yang dimulai pada langkah 6 dengan menggunakan $p$ prosesor. Langkah 7 hingga 18 dilakukan apabila $p$ tidak habis membagi $H$. Langkah 8 hingga 17 dilakukan oleh prosesor $e$ dimana $e = 1,2,\dots,mod(H,p)$. Setiap prosesor $e$ memiliki variabel lokal yaitu $X\_local$ berukuran $(\lfloor H/p \rfloor + 1)$-by-$(L+1)$ dimana kolom ke-$(l+1)$ adalah nilai

$$X(t_{l+1}) = X(t_l) + \left(r - \frac{1}{2}v(t_l)\right)\Delta t + \sqrt{v(t_l)\Delta t} * W(t_l)$$

dimana $l = 0,1,\dots,L-1$. Berdasarkan $X\_local$ dapat dihitung $S$. Lalu langkah 10, $W\_end$ menyimpan nilai $W$ yang diperlukan oleh prosesor $e$ untuk mengerjakan 1 *sample path*. Pada langkah berikutnya, $Z\_end$ menyimpan nilai $Z$ yang diperlukan oleh prosesor $e$ untuk mengerjakan 1 *sample path*. Pada langkah 14 hingga 17, masing-masing prosesor $e$ melakukan proses penghitungan $X(t_{l+1})$ dimana $l = 0,1,\dots,L-1$ yang berkorelasi dengan *sample path* ke-$(p * \lfloor H/p \rfloor + 1)$ hingga *sample path* ke-$H$ secara bersamaan dan hasilnya disimpan pada baris terakhir $X\_local$ masing-masing prosesor $e$. Pada langkah 17, menghitung entri baris ke-$(p * \lfloor H/p \rfloor + e)$ dari $S$ dimana

$$U_{p*\lfloor H/p \rfloor + e,:} = U_0 e^{X\_local_{\lfloor H/p \rfloor + 1,:}}$$

dan $X\_local_{\lfloor H/p \rfloor + 1,:}$ adalah entri baris ke-$(\lfloor H/p \rfloor + 1)$ dari $X\_local$ pada masing-masing prosesor $e$. Kemudian pada langkah 18, inisialisasi $X\_local$ pada prosesor $f$ dimana $f = mod(H,p)+1, mod(H,p)+2,\dots,p$ sebagai matriks berukuran $(\lfloor H/p \rfloor\ x\ (L+1)$ yang semua entrinya adalah nol. Langkah 20 dilakukan apabila $p$ habis membagi $H$, yaitu inisialisasi $X\_local$ pada prosesor $i$ dimana $i = 1,2,\dots,p$ sebagai matriks berukuran $(\lfloor H/p \rfloor\ x\ (L+1)$ yang semua entrinya adalah nol.

Kemudian langkah 21 adalah membuat variabel lokal $W\_local$ pada prosesor $i$ dimana $i = 1,2, \dots, p$ yang digunakan untuk menyimpan nilai baris tertentu dari $W$ yang diperlukan oleh masing-masing prosesor untuk menghitung $\lfloor H/p \rfloor$ *sample path*. $W\_local$ adalah matriks berukuran $\lfloor H/p \rfloor \; x \; L$ dan entri dari $W\_local$ adalah entri baris ke-$((i-1) * \lfloor H/p \rfloor + 1)$ sampai baris ke-$(i * \lfloor H/p \rfloor)$ dari $W$ dimana $i = 1,2, \dots, p$. Pada langkah 22 adalah membuat variabel lokal $Z\_local$ pada prosesor $i$ dimana $i = 1,2, \dots, p$ yang digunakan untuk menyimpan nilai baris tertentu dari $Z$ yang diperlukan oleh masing-masing prosesor untuk menghitung $\lfloor H/p \rfloor$ *sample path*. $Z\_local$ adalah matriks berukuran $\lfloor H/p \rfloor \; x \; L$ dan entri dari $Z\_local$ adalah entri baris ke-$((i-1) * \lfloor H/p \rfloor + 1)$ sampai baris ke-$(i * \lfloor H/p \rfloor)$ dari $Z$ dimana $i = 1,2, \dots, p$. Jadi, setiap prosesor $i$ memiliki variabel local $W\_local$ dan $Z\_local$ yang nilainya berbeda-beda.

Langkah berikutnya adalah setiap prosesor $i$ menghitung $\lfloor H/p \rfloor$ *sample path* secara bersamaan. Setiap prosesor $i$ melakukan proses penghitungan $X(t_{l+1})$ dimana $l = 0,1, \dots, L-1$ yang berkorelasi dengan *sample path* ke-$((i-1) * \lfloor H/p \rfloor + 1)$ hingga *sample path* ke-$(i * \lfloor H/p \rfloor)$ dan disimpan dalam matriks $X\_local$ pada baris ke-1 hingga baris ke-$\lfloor H/p \rfloor$. Setelah mendapatkan nilai $X\_local$ pada masing-masing prosesor $i$, maka pada langkah 28 menghitung entri baris ke-$((i-1) * \lfloor H/p \rfloor + 1)$ hingga baris ke-$(i * \lfloor H/p \rfloor)$ dari $U$ dimana

$$U_{(i-1)*\lfloor H/p \rfloor + 1 : i*\lfloor H/p \rfloor, :} = U_0 e^{X\_local_{1:\lfloor H/p \rfloor, :}}$$

dan $X\_local_{1:\lfloor H/p \rfloor, :}$ adalah entri baris ke-1 hingga baris ke-$\lfloor H/p \rfloor$ dari $X\_local$ pada masing-masing prosesor $i$. Proses paralel diakhiri.

Langkah selanjutnya adalah menghitung $\overline{U} = sum(U, 2)/L$ yang merupakan nilai dari

$$\frac{1}{L} \sum_{i=1}^{L} U(t_i)^{(k)}$$

dimana $k = 1,2, \dots, H$. Pada langkah 30, harga opsi dihitung berdasarkan persamaan (3.13).

Dengan demikian diharapkan waktu komputasi menjadi lebih cepat dalam menghitung harga opsi *call* Asia. Implementasi dari Algoritma 3.5.1 akan diberikan pada Bab 4.

Masalah lain yang muncul adalah ketika menghitung harga opsi secara harian atau bahkan per jam. Artinya apabila waktu jatuh tempo opsi semakin lama, maka melibatkan nilai $L$ yang besar untuk menghitung

$$\overline{U} = \frac{1}{L}\sum_{i=1}^{L} U(t_i)$$

dimana $t_0 \le t_1 < t_2 < \cdots < t_L \le T$. Berdasarkan (3.13) nilai $\overline{U}$ diperlukan dalam mendapatkan harga opsi.

Berikut ini akan dibangun algoritma paralel secara titik diskretisasi untuk menghitung $\overline{U}$. Pertama-tama perhatikan kembali model harga *underlying asset U* di bawah kondisi *martingale* mengikuti persamaan diferensial stokastik:

$$d\ln U(t) = \left(r - \frac{1}{2}v(t)\right)dt + \sqrt{v(t)}dW(t)\,,\;\; U(0) = U_0 > 0$$

memiliki solusi EM berdasarkan persamaan (3.11) yaitu

$$\ln U(t_{i+1}) = \ln U(t_i) + \left(r - \frac{1}{2}v(t_i)\right)\Delta t + \sqrt{v(t_i)\Delta t} * W(t_i).$$

Terlihat bahwa untuk menghitung $U(t_{i+1})$ bergantung pada nilai $U(t_i)$ dan $v(t_i)$. Akan tetapi jika (3.11) ditulis dalam bentuk:

$$\ln U(t_{i+1}) - \ln U(t_i) = \left(r - \frac{1}{2}v(t_i)\right)\Delta t + \sqrt{v(t_i)\Delta t} * W(t_i)$$

dan dengan memisalkan

$$\Delta\ln U(t_{i+1}) = \ln U(t_{i+1}) - \ln U(t_i)$$

maka diperoleh

$$\Delta\ln U(t_{i+1}) = \left(r - \frac{1}{2}v(t_i)\right)\Delta t + \sqrt{v(t_i)\Delta t} * W(t_i)$$

sehingga untuk mendapatkan nilai $\Delta\ln U(t_{i+1})$ hanya bergantung dengan nilai $v(t_i)$. Jika $v(t_i)$ dihitung terlebih dahulu untuk setiap $t_i$ dimana $i = 1,2,\ldots,L$, maka untuk menghitung $\Delta\ln U(t_{i+1})$ dapat dikerjakan secara bersamaan. Setelah diperoleh nilai $\Delta\ln U(t_{i+1})$ dimana $i = 0,1,\ldots,L-1$, maka $\ln U(t_{i+1})$ dapat dihitung dengan $\ln U(t_{i+1}) = \ln U(t_i) + \Delta\ln U(t_{i+1})$ dimana $i = 0,1,\ldots,L-1$. Hal inilah yang menjadi ide paralel untuk menghitung $\overline{U}$ dimana proses penghitungan $\Delta\ln U(t_{i+1})$ dikerjakan secara paralel.

Algoritma 3.6 adalah algoritma paralel secara titik diskretisasi untuk menghitung $\bar{U}$. Pada algoritma ini akan digunakan *p* prosesor. Input dari algoritma ini adalah $r$ yang merupakan suku bunga bebas resiko, $\kappa$ adalah kecepatan pengembalian, $v$ adalah *return variance*, $T$ adalah batas atas interval $[0, T]$, $t_L = T$ adalah waktu jatuh tempo opsi, $\bar{v}$ adalah *reference variance*, $\psi$ adalah volatilitas, dan banyaknya prosesor yaitu *p* dimana $L \geq p$.

Tabel 3.6 Algoritma paralel secara titik diskretisasi untuk menghitung $\bar{U}$ pada model harga opsi *call* Asia

**Algoritma 3.5.2 Euler-Maruyama method for Asian *call* options in parallel across time**

INPUT : $r$ is the constant instantaneous risk-free interest rate, $T$ is upper bound of interval $[0, T]$, $t_L = T$ is maturity date of the option, $p$ is a number of processors where $L \geq p$, and $v$ is return variance which is assumed to be amean-reverting process

$$dv(t) = \kappa\big(\bar{v} - v(t)\big)dt + \psi\sqrt{v(t)}dW_v(t)\,, \quad v(0) = v_0 > 0$$

where $\bar{v}$ is reference variance, $\kappa$ is speed of reversion, and $\psi$ is volatility of the volatility.

OUTPUT : $\bar{U} = \frac{1}{L}\sum_{i=1}^{L} S_i$ where $U$ is an 1-by-$L$ array of Euler-Maruyama solution to the:

$$d\ln U(t) = \left(r - \frac{1}{2}v(t)\right)dt + \sqrt{v(t)}dW(t)\,, \quad U(0) = U_0 > 0$$

and $U_i$ is $i$th entry of $U$.

1. **set** $\Delta t := T/L$
2. **set** $W$ is an 1-by-$L$ array of random number from normally distributed $N(0,1)$
3. **set** $Y$ is an 1-by-$L$ array of random number from normally distributed $N(0,1)$
4. **set** $Z$ is an 1-by-$L$ array of random number from normally distributed $\rho * W + \sqrt{1-\rho^2} * Y$

5. **set** $U$ is an 1-by-$(L+1)$ zeros array
6. **set** $dX$ is an 1-by-$(L+1)$ zeros array
7. **set** $v$ is an 1-by-$L$ zeros array
8. **set** $v(1) := v_0$
9. **for** $j = 1$ to $(L-1)$ **do**
10. $v(j+1) = max\big(0, v(j) + \kappa * (v_0 - v(j))\big) * \Delta t + \psi * sqrt(v(j) * \Delta t) * Z\_end_j)$

    **end for**
11. Each processor $i$ where $i = 1,2, \dots, p$ **do in parallel**
12. **if** $mod(L,p) \neq 0$ **do**
13. Each processor $e$ where $e = 1,2, \dots, mod(L,p)$ **do in parallel**
14. **set** $dX\_local$ is an 1-by-$(\lfloor L/p \rfloor + 1)$ zeros array
15. **set** $W\_end$ is an entry $(p * \lfloor L/p \rfloor + e)$th column of $W$
16. **set** $v\_end$ is an entry $(p * \lfloor L/p \rfloor + e)$th column of $v$
17. $dX\_local_{\lfloor L/p \rfloor + 1} := (r - 0.5 * v\_end) * \Delta t + sqrt(v\_end * \Delta t) * W\_end$
18. **set** $dX_{p * \lfloor L/p \rfloor + 1 + e} = dX\_local_{\lfloor L/p \rfloor + 1}$

    **end do**
19. **set** $dX\_local$ is an 1-by-$\lfloor L/p \rfloor$ zeros array for processor $f$ where $f = mod(L,p) + 1, mod(L,p) + 2, \dots, p$
20. **else**
21. **set** $dX\_local$ is an 1-by-$\lfloor L/p \rfloor$ zeros array for each processor

    **end if**
22. **set** $W\_local$ is an 1-by-$\lfloor L/p \rfloor$- array whose entries are $((i-1) * \lfloor L/p \rfloor + 1)$th column until $(i * \lfloor L/p \rfloor)$th column of $W$
23. **set** $v\_local$ is an 1-by-$\lfloor L/p \rfloor$ array whose entries are $((i-1) * \lfloor L/p \rfloor + 1)$th column until $(i * \lfloor L/p \rfloor)$th column of $v$
24. **for** $j = 1$ to $\lfloor L/p \rfloor$ **do**
25. $dX\_local_j := \big(r - 0.5 * v\_local_j\big) * \Delta t + sqrt\big(v\_local_j * \Delta t\big) * W\_local_j$

    **end for**

26. **set** $dX_{(i-1)*\lfloor L/p \rfloor+2:i*\lfloor L/p \rfloor+1} = dX\_local_{1:\lfloor L/p \rfloor}$

    **end do**

27. **set** $X := cumsum(dX)$

28. **set** $U := U_0 e^X$

29. **set** $\overline{U} := sum(U, 2)/L$

Langkah 1 adalah inisialisasi $\Delta t = T/L$. Kemudian inisialisasi $W$ dan $Y$ yang merupakan matriks berukuran $1\ x\ L$ bilangan random berdistribusi $N(0,1)$. Pada langkah 4 inisialisasi $Z$ yang merupakan matriks berukuran $1\ x\ L$ bilangan random berdistribusi $\rho * W + \sqrt{1-\rho^2} * Y$. Selanjutnya inisialisasi $U$ sebagai matriks berukuran 1-by-$(L+1)$ yang semua entrinya adalah nol. Matriks $U$ disimpan pada *shared memory* dan digunakan untuk menyimpan nilai harga *underlying stock* $U$ dimana kolom ke-$l$ adalah nilai $U(t_{l-1})$. Pada langkah 6 inisialisasi $dX$ sebagai matriks berukuran 1-by-$(L+1)$ yang semua entrinya adalah nol. Langkah 7 hingga 10 merupakan proses penghitungan $v(t_h)$ untuk setiap $t_h$ dimana $h = 0,1, \dots, L-1$, dan nilai $v(t_h)$ disimpan pada matriks $v$ kolom ke-$(h+1)$.

Langkah selanjutnya merupakan proses paralel yang dimulai pada langkah 11 dengan menggunakan $p$ prosesor. Langkah 12 hingga 19 dilakukan apabila $p$ tidak habis membagi $L$. Langkah 13 hingga 18 dilakukan oleh prosesor $e$ dimana $e = 1,2, \dots, mod(L,p)$. Setiap prosesor $e$ memiliki variabel lokal yaitu $dX\_local$ berukuran $1\ x\ (\lfloor L/p \rfloor + 1)$. Lalu pada langkah 10, prosesor $e$ membuat variabel $W\_end$ yang digunakan untuk menyimpan entri kolom ke-$(p * \lfloor L/p \rfloor + e)$ dari $W$. Pada langkah berikutnya, prosesor $e$ membuat variabel $v\_end$ yang digunakan untuk menyimpan entri kolom ke-$(p * \lfloor L/p \rfloor + e)$ dari $v$. Pada langkah 17 masing-masing prosesor $e$ menghitung entri baris ke-$(\lfloor L/p \rfloor + 1)$ dari $dX\_local$ dimana

$dX\_local_{\lfloor L/p \rfloor+1} := (r - 0.5 * v\_end) * \Delta t + sqrt(v\_end * \Delta t) * W\_end.$

Kemudian pada langkah 18, meng-*update* nilai kolom ke- $(p * \lfloor L/p \rfloor + 1 + e)$ dengan nilai $dX\_local_{\lfloor L/p \rfloor+1}$. Langkah selanjutnya inisialisasi $dX\_local$ pada prosesor $f$ dimana $f = mod(L,p)+1, mod(L,p)+2, \dots, p$ sebagai matriks berukuran $(1\ x\ \lfloor L/p \rfloor)$ yang semua entrinya adalah nol. Langkah 21 dilakukan

apabila $p$ habis membagi $L$, yaitu inisialisasi $dX\_local$ pada prosesor $i$ dimana $i = 1,2, \dots, p$ sebagai matriks berukuran $(1\ x\ \lfloor L/p \rfloor)$ yang semua entrinya adalah nol.

Kemudian langkah 22 adalah membuat variabel lokal $W\_local$ pada prosesor $i$ dimana $i = 1,2, \dots, p$ yang digunakan untuk menyimpan nilai baris tertentu dari $W$ yang diperlukan oleh masing-masing prosesor untuk menghitung $\lfloor L/p \rfloor$ titik diskretisasi. $W\_local$ adalah matriks berukuran $1\ x\ \lfloor L/p \rfloor$ dan entri dari $W\_local$ adalah entri baris ke-$((i-1) * \lfloor L/p \rfloor + 1)$ sampai baris ke-$(i * \lfloor L/p \rfloor)$ dari $W$ dimana $i = 1,2, \dots, p$. Pada langkah 23 adalah membuat variabel lokal $v\_local$ pada prosesor $i$ dimana $i = 1,2, \dots, p$ yang digunakan untuk menyimpan nilai baris tertentu dari $v$ yang diperlukan oleh masing-masing prosesor untuk menghitung $\lfloor L/p \rfloor$ titik diskretisasi, $v\_local$ adalah matriks berukuran $1\ x\ \lfloor L/p \rfloor$ dan entri dari $v\_local$ adalah entri baris ke-$((i-1) * \lfloor L/p \rfloor + 1)$ sampai baris ke-$(i * \lfloor L/p \rfloor)$ dari $v$ dimana $i = 1,2, \dots, p$. Jadi, setiap prosesor $i$ memiliki variabel local $W\_local$ dan $v\_local$ yang nilainya berbeda-beda.

Langkah berikutnya adalah setiap prosesor $i$ menghitung $\lfloor L/p \rfloor$ titik diskretisasi secara bersamaan. Setiap prosesor $i$ melakukan proses penghitungan titik diskretisasi ke-$((i-1) * \lfloor L/p \rfloor + 1)$ hingga titik diskretisasi ke-$(i * \lfloor L/p \rfloor)$ dan disimpan dalam matriks $dX\_local$ pada baris ke-1 hingga baris ke-$\lfloor L/p \rfloor$. Setelah mendapatkan nilai $dX\_local$ pada masing-masing prosesor $i$, maka pada langkah 26 meng-*update* entri kolom ke-$((i-1) * \lfloor L/p \rfloor + 2)$ hingga kolom ke-$(i * \lfloor L/p \rfloor + 1)$ dari $dX$ dengan baris ke-1 hingga baris ke-$\lfloor L/p \rfloor$ dari $dX\_local$ dan proses paralel diakhiri.

Langkah selanjutnya adalah menghitung $X = cumsum(dX)$, $cumsum$ adalah $cummulative\ sum$ artinya entri ke-$c$ dari $X$ adalah $dX_1 + \cdots + dX_c$. Pada langkah 28 dihitung harga *underlying stock* $U$. Langkah terakhir adalah menghitung $\overline{U} = sum(U)/L$ yang merupakan nilai dari

$$\frac{1}{L}\sum_{i=1}^{L} U(t_i).$$

Dengan demikian diharapkan waktu komputasi menjadi lebih cepat dalam menghitung $\overline{U}$ dan tentunya akan mempercepat dalam menghitung harga opsi *call* Asia dengan menggunakan persamaan (3.13). Implementasi dari algoritma 3.5.2 akan diberikan pada Bab 4.

# BAB 4
# IMPLEMENTASI ALGORITMA PARALEL UNTUK MODEL PDS

Pada Bab 3 telah dijelaskan mengenai algoritma paralel Euler-Maruyama (EM) secara *sample path* untuk PDS, algoritma paralel untuk sistem PDS dimana matriks koefisien suku deterministik $A$ dan matriks koefisien suku stokastik $B^k$ merupakan matriks tridiagonal, algoritma paralel secara *sample path* untuk menghitung $H$ *sample path* untuk mendapatkan harga opsi *call* Asia, dan algoritma paralel secara titik diskretisasi untuk menghitung $\overline{U}$ pada model harga opsi *call* Asia. Selanjutnya pada bab ini akan diberikan implementasi dan simulasi dari algoritma-algoritma tersebut dalam bentuk program. Program tersebut akan berjalan pada mesin *multicore* (*laptop*) dengan spesifikasi sebagai berikut:

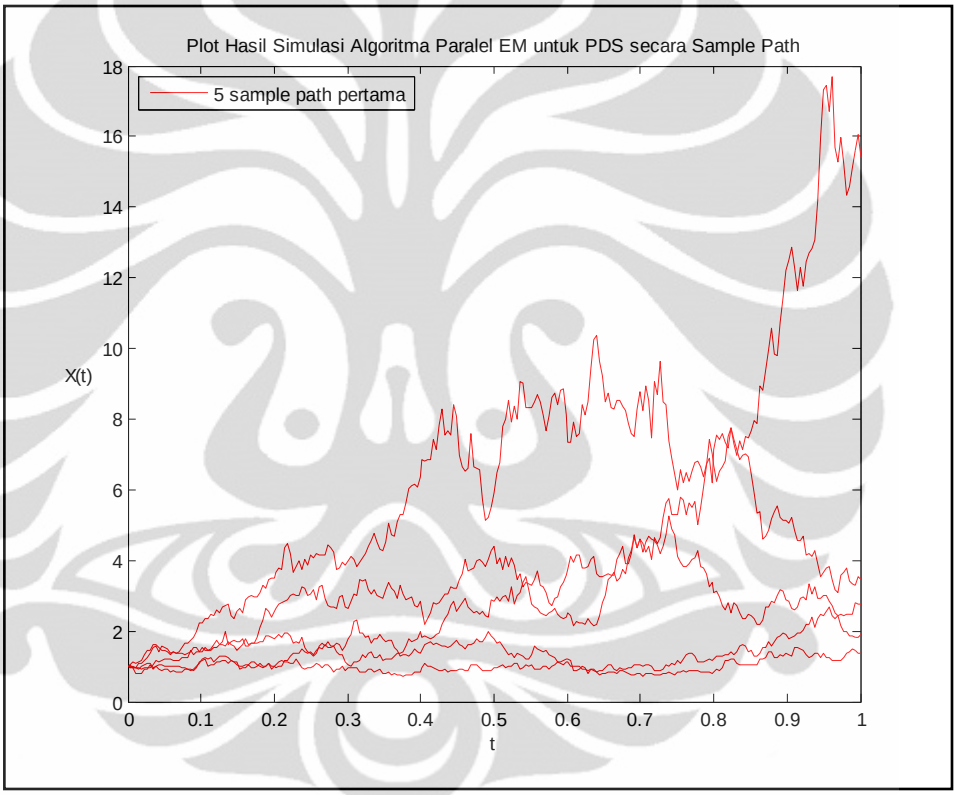
- Prosesor : *Intel(R) Core(TM) i5 M460@* 2.53 GHz
- Memori : 2048MB RAM
- Sistem Operasi : *Windows 7 Professional* 32-bit
- Perangkat Lunak : MATLAB versi R2009a dengan *Parallel Computing Toolbox* (versi *trial*)

Secara umum, simulasi ini dapat dijalankan dengan $p$ prosesor dimana $p$ adalah bilangan bulat, tetapi karena keterbatasan perangkat lunak yang digunakan, maka simulasi hanya dapat dilakukan dengan maksimum 4 prosesor. Dari hasil simulasi tersebut akan dikaji kinerja paralel dari algoritma-algoritma tersebut dengan menghitung *speed up* dan efisiensi paralel.

## 4.1 Implementasi Algoritma Paralel secara *Sample Path* untuk PDS

Pada Subbab 3.2 telah diberikan Algoritma 3.2 yaitu algoritma paralel secara *sample path* untuk PDS. Program MATLAB pada Lampiran 1 merupakan implementasi dari Algoritma 3.2. Input dari program ini adalah $p$ (banyak prosesor) dan $S$ (banyak simulasi). Program ini mengambil nilai $f(X(t)) = \lambda X(t)$ dan $g(X(t)) = \mu X(t)$ dengan $\lambda$ dan $\mu$ adalah konstanta real. Program ini bertujuan

untuk menguji sejauh mana kinerja paralel dari Algoritma 3.2. Pada program ini dilakukan $S$ simulasi dengan $\lambda = 2$ dan $\mu = 1$ dengan $L = 2^8$, $T = 1$ dan $X(0) = 1$. Output dari program ini adalah $S$ *sample path* yang dikerjakan oleh $1,2,\ldots,p$ prosesor. Jika dimasukkan input $p = 4$, maka program akan menghitung $S$ *sample path* dengan menggunakan 1 prosesor, 2 prosesor, 3 prosesor, dan 4 prosesor dan menghasilkan output yang sama. Dengan mengambil nilai $S$ = 1000 maka program akan menghasilkan output seperti terlihat pada Gambar 4.1 dengan menggunakan 1 prosesor, 2 prosesor, 3 prosesor, dan 4 prosesor.




Gambar 4.1 Plot 5 *sample path* pertama hasil simulasi algoritma Paralel secara *sample path* untuk PDS

Program ini juga akan menghasilkan *running time* program untuk 1 prosesor, 2 prosesor, 3 prosesor, dan 4 prosesor. Tabel 4.1 adalah hasil *running time* yang telah didapat untuk beberapa nilai $S$ tertentu. Berdasarkan hasil *running time* pada Tabel 4.1, maka diperoleh *speed up* algoritma paralel secara *sample path* untuk PDS (lihat Tabel 4.2). Dan juga diperoleh efisiensi algoritma paralel secara *sample path* untuk PDS (lihat Tabel 4.3).

Tabel 4.1 Hasil *running time* algoritma paralel secara *sample path* untuk PDS

| Banyak *sample path* (*S*) | *Running Time* (dalam sekon) | | | |
|---|---|---|---|---|
| | 1 Prosesor | 2 Prosesor | 3 Prosesor | 4 Prosesor |
| 1,000 | 0.676712 | 0.380117 | 0.303609 | 0.294459 |
| 5,000 | 2.507737 | 1.236540 | 0.981806 | 0.939102 |
| 10,000 | 4.220312 | 2.389176 | 1.852649 | 1.672050 |
| 15,000 | 5.979968 | 3.294437 | 2.649422 | 2.415978 |
| 20,000 | 7.900786 | 4.347245 | 3.475586 | 3.136930 |
| 25,000 | 9.844665 | 5.367693 | 4.314641 | 3.817435 |

Tabel 4.2 *Speed up* algoritma paralel secara *sample path* untuk PDS

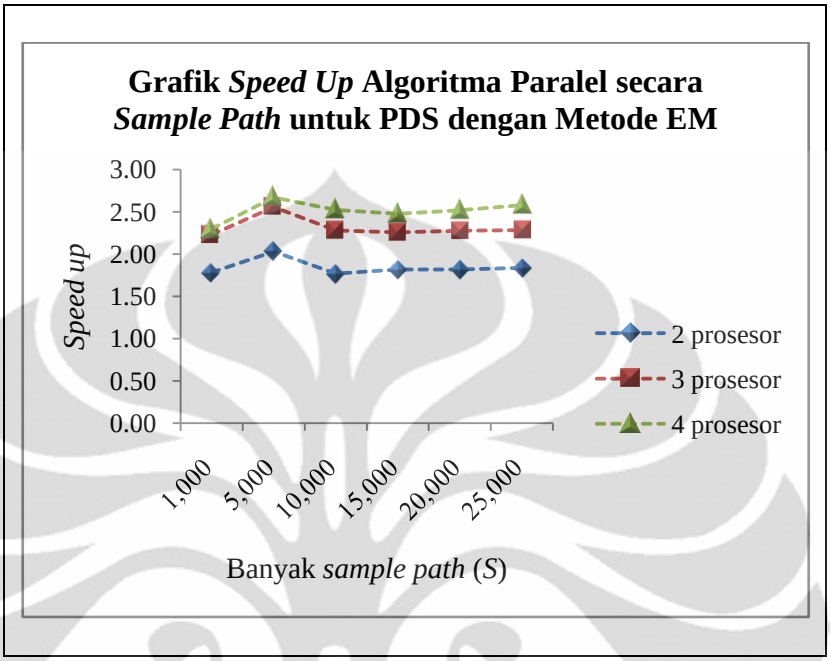
| Banyak *sample path* (*S*) | *Speed Up* | | |
|---|---|---|---|
| | 2 Prosesor | 3 Prosesor | 4 Prosesor |
| 1,000 | 1.78 | 2.23 | 2.30 |
| 5,000 | 2.03 | 2.55 | 2.67 |
| 10,000 | 1.77 | 2.28 | 2.52 |
| 15,000 | 1.82 | 2.26 | 2.48 |
| 20,000 | 1.82 | 2.27 | 2.52 |
| 25,000 | 1.83 | 2.28 | 2.58 |

Tabel 4.3 Efisiensi algoritma paralel secara *sample path* untuk PDS

| Banyak *sample path* (*S*) | Efisiensi | | |
|---|---|---|---|
| | 2 Prosesor | 3 Prosesor | 4 Prosesor |
| 1,000 | 0.89 | 0.74 | 0.57 |
| 5,000 | 1.01 | 0.85 | 0.67 |
| 10,000 | 0.88 | 0.76 | 0.63 |
| 15,000 | 0.91 | 0.75 | 0.62 |
| 20,000 | 0.91 | 0.76 | 0.63 |
| 25,000 | 0.92 | 0.76 | 0.64 |

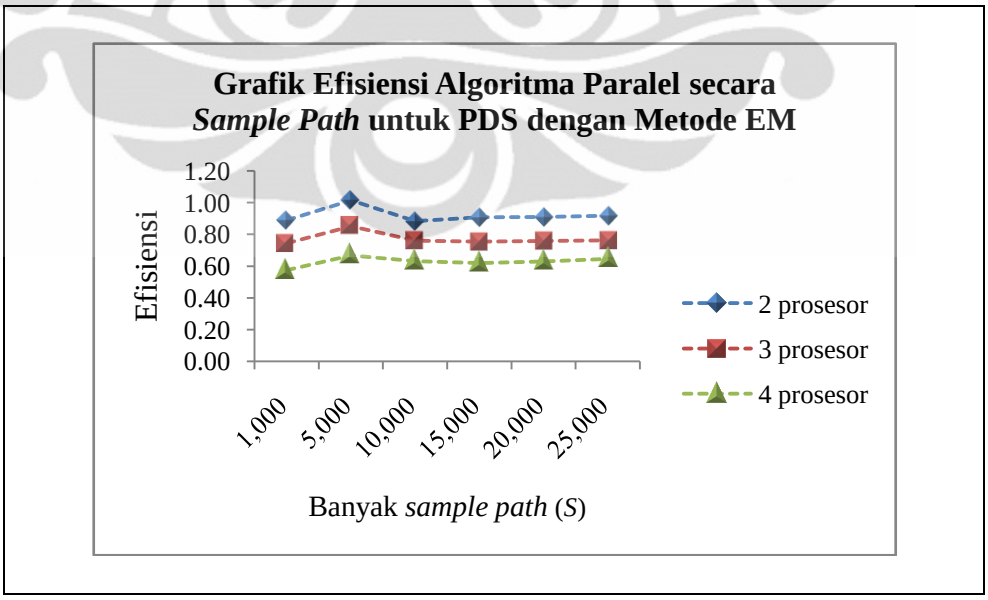
Grafik *speed up* algoritma paralel secara *sample path* untuk PDS dapat dilihat pada Gambar 4.2. Berdasarkan Gambar 4.2 terlihat bahwa semakin banyak simulasi atau semakin banyak *sample path* (*S*) yang dihitung maka nilai *speed up* relatif konstan untuk $S$=1000 hingga $S$=25000. Terlihat pula algoritma paralel secara *sample path* untuk PDS lebih efektif dengan menggunakan 4 prosesor

karena menghasilkan *speed up* yang lebih baik daripada menggunakan 2 atau 3 prosesor (lihat Gambar 4.2).




Gambar 4.2 Grafik *speed up* algoritma paralel secara *sample path* untuk PDS

Grafik efisiensi dari algoritma paralel untuk PDS secara *sample path* dapat dilihat pada Gambar 4.3.




Gambar 4.3 Grafik efisiensi algoritma paralel secara *sample path* untuk PDS

Berdasarkan Gambar 4.3 terlihat bahwa semakin banyak simulasi atau semakin banyak *sample path* (*S*) yang dihitung maka nilai efisiensi relatif konstan untuk *S*=1000 hingga *S*=25000. Terlihat pula algoritma paralel secara *sample path* untuk PDS lebih efisien dengan menggunakan 2 prosesor karena menghasilkan efisiensi yang lebih baik daripada menggunakan 3 atau 4 prosesor (lihat Gambar 4.3).

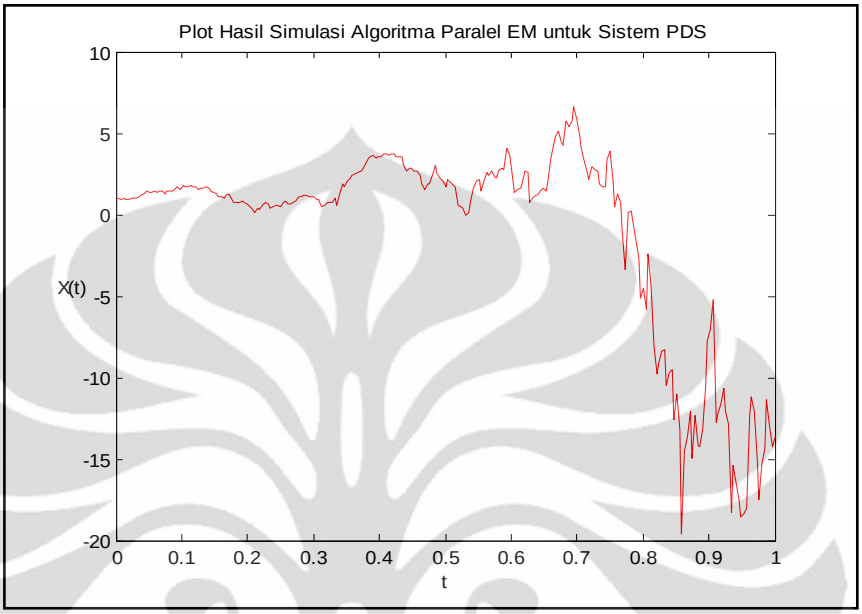
## 4.2 Implementasi Algoritma Paralel untuk Sistem PDS

Pada Subbab 3.4 telah diberikan Algoritma 3.4 yaitu algoritma paralel untuk sistem PDS dimana matriks suku deterministik $A$ dan matriks suku stokastik $B^k$ telah ditransformasi menjadi matriks tridiagonal. Program MATLAB pada Lampiran 2 merupakan implementasi dari Algoritma 3.4. Input dari program ini adalah *p* (banyak prosesor), *N* (banyak PDS dalam sistem), dan *W* (banyak proses *Wiener* yang saling bebas). Matriks tridiagonal $A$ dan $B^k$ pada program ini memiliki elemen-elemen yang berasal dari bilangan random $N(0,1)$. Dalam membuat matriks tridiagonal $A$ dan $B^k$ digunakan fungsi *sparse* yang ada di MATLAB, artinya hanya elemen tak nol dari matriks tridiagonal $A$ dan $B^k$ yang disimpan sehingga menghemat memori dan mempercepat proses komputasi.

Pada program ini, input *p* dan *N* berupa bilangan bulat $2^n$ dimana $n \epsilon N$ dan $p \mid N$. Sedangkan input *W* dapat berupa bilangan bulat positif atau array $1 \; x \; n$ dimana $n \epsilon N$ yang berisi bilangan bulat positif. Program ini bertujuan untuk menguji sejauh mana kinerja paralel dari Algoritma 3.4. Pada program ini diambil nilai $L = 2^8, \; T = 1$ dan $X(0) = 1$. Output dari program ini adalah solusi EM untuk sistem PDS yang dikerjakan oleh $1,2, \ldots, p$ prosesor.

Jika dimasukkan input $p = 4$, maka program akan menghitung solusi EM dengan menggunakan 1 prosesor, 2 prosesor, 3 prosesor, dan 4 prosesor dan menghasilkan output yang sama. Dengan mengambil nilai $N$ = 16 dan $W$ = 2 maka program akan menghasilkan output yang sama dengan menggunakan 1 prosesor, 2 prosesor, 3 prosesor, dan 4 prosesor. Gambar 4.4 menampilkan 1 *sample path* untuk suatu nilai $X_j(t)$ dimana $j$ diambil secara acak dimana $X_j(t)$ adalah solusi EM dari PDS ke-$j$ pada sistem PDS.

Program ini juga akan menghasilkan *running time* program untuk 1 prosesor, 2 prosesor, 3 prosesor, dan 4 prosesor. Tabel 4.4 adalah hasil *running time* yang telah didapat untuk beberapa nilai $N$ dan $W$ tertentu.




Gambar 4.4 Plot solusi dari $X_j(t)$ untuk sistem PDS hasil simulasi algoritma paralel untuk sistem PDS dimana $j$ diambil secara acak

Berdasarkan Tabel 4.4, maka diperoleh *speed up* algoritma paralel untuk sistem PDS (lihat Tabel 4.5) dan efisiensi algoritma paralel untuk sistem PDS (lihat Tabel 4.6).

Tabel 4.4 Hasil *running time* algoritma paralel untuk sistem PDS dimana matriks suku deterministik $A$ dan matriks suku stokastik $B^k$ telah ditransformasi menjadi matriks tridiagonal

| **Banyak PDS ($N$)** | **Banyak Proses *Wiener* ($M$)** | ***Running Time* (dalam sekon)** | | |
|---|---|---|---|---|
| | | **1 Prosesor** | **2 Prosesor** | **4 Prosesor** |
| **16** | **16** | 0.999070439 | 0.963907187 | 1.317836871 |
| **64** | **16** | 0.94779891 | 0.67624575 | 1.20392683 |
| | **64** | 1.072975794 | 1.353626989 | 2.284757228 |
| **256** | **16** | 1.0952144 | 0.8207656 | 1.30255393 |
| | **64** | 3.152852787 | 2.247084891 | 3.288586095 |
| | **128** | 5.361929013 | 4.290473176 | 5.222870925 |
| | **256** | 13.23229296 | 8.443380668 | 10.27486879 |
| **512** | **16** | 1.39451407 | 0.96315119 | 1.46962628 |
| | **64** | 4.06879455 | 3.33990561 | 4.11466663 |

| | | | | |
|---|---|---|---|---|
| | **128** | 11.88893774 | 7.057924932 | 7.773093549 |
| | **256** | 28.59745558 | 15.02385495 | 13.63763538 |
| **1,024** | **16** | 2.786206273 | 1.922805118 | 2.315326165 |
| | **64** | 10.54576284 | 5.633615659 | 5.118462683 |
| | **128** | 26.40572616 | 13.42513469 | 10.33285167 |
| | **256** | 62.93404859 | 35.73092901 | 25.18459036 |
| **2,048** | **16** | 4.681343382 | 2.844516158 | 2.813571588 |
| | **64** | 23.56121617 | 10.66916807 | 8.8704954 |
| | **128** | 54.22106222 | 27.74563419 | 19.73821223 |
| | **256** | 127.4884992 | 72.96843343 | 54.47435806 |
| **4,096** | **16** | 6.86761951 | 3.79495154 | 3.63022294 |
| | **64** | 46.34842524 | 23.30727705 | 17.15101515 |
| | **128** | 109.0970755 | 56.27096675 | 41.70272898 |
| | **256** | 257.9106823 | 145.6212741 | 113.9367388 |

Tabel 4.5 *Speed up* algoritma paralel untuk sistem PDS dimana matriks suku deterministik $A$ dan matriks suku stokastik $B^k$ telah ditransformasi menjadi matriks tridiagonal

| **Banyak PDS (*N*)** | **Banyak Proses *Wiener* (*M*)** | ***Speed up*** | |
|---|---|---|---|
| | | **2 Prosesor** | **4 Prosesor** |
| **16** | **16** | 1.04 | 0.76 |
| **64** | **16** | 1.40 | 0.79 |
| | **64** | 0.79 | 0.47 |
| **256** | **16** | 1.33 | 0.84 |
| | **64** | 1.40 | 0.96 |
| | **128** | 1.25 | 1.03 |
| | **256** | 1.57 | 1.29 |
| **512** | **16** | 1.45 | 0.95 |
| | **64** | 1.22 | 0.99 |
| | **128** | 1.68 | 1.53 |
| | **256** | 1.90 | 2.10 |
| **1,024** | **16** | 1.45 | 1.20 |
| | **64** | 1.87 | 2.06 |
| | **128** | 1.97 | 2.56 |
| | **256** | 1.76 | 2.50 |
| **2,048** | **16** | 1.65 | 1.66 |
| | **64** | 2.21 | 2.66 |
| | **128** | 1.95 | 2.75 |
| | **256** | 1.75 | 2.34 |
| **4,096** | **16** | 1.81 | 1.89 |

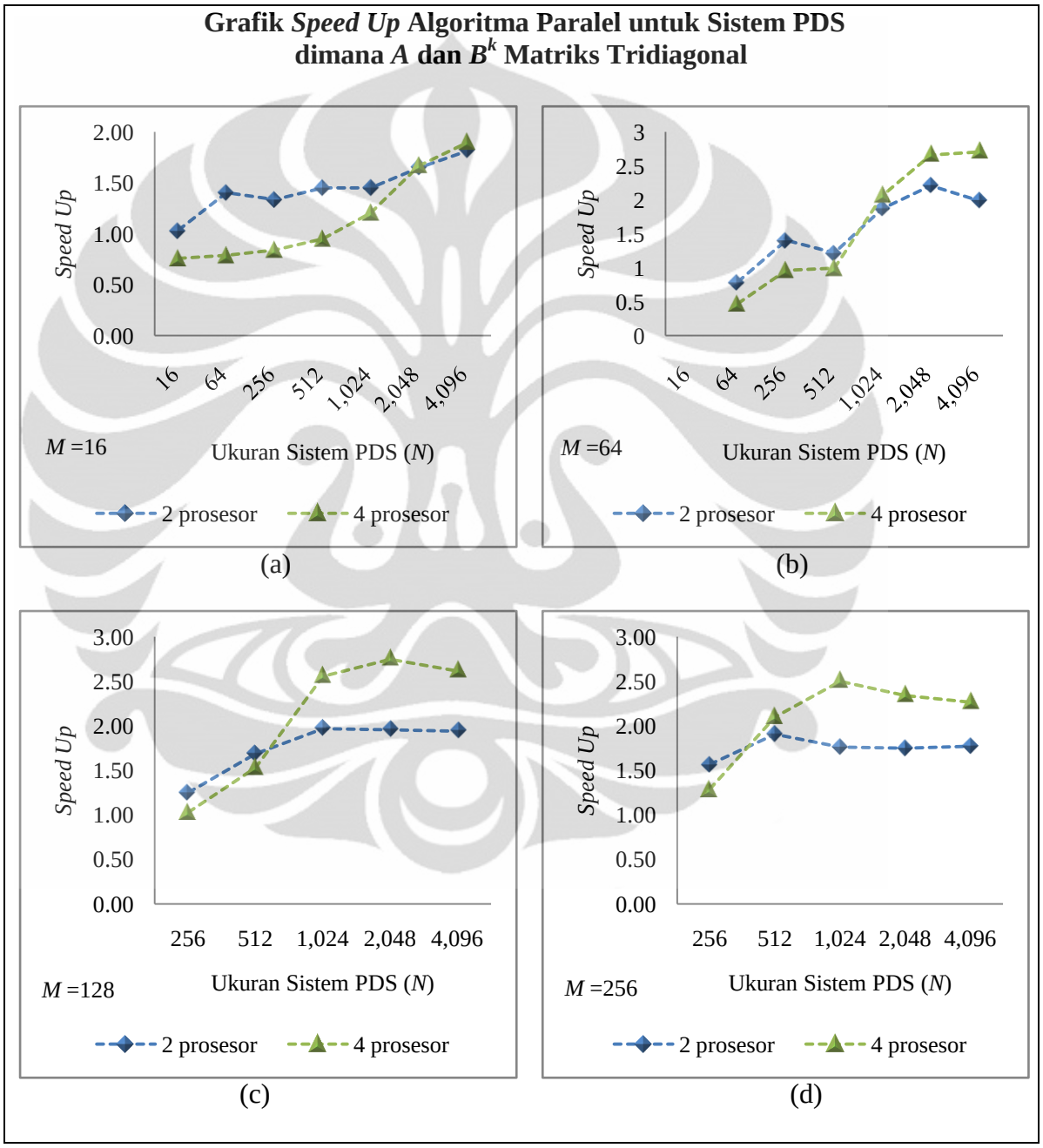
| | | | |
|---|---|---|---|
| | **64** | 1.99 | 2.70 |
| | **128** | 1.94 | 2.62 |
| | **256** | 1.77 | 2.26 |

Tabel 4.6 Efisiensi algoritma paralel untuk sistem PDS dimana matriks suku deterministik $A$ dan matriks suku stokastik $B^k$ telah ditransformasi menjadi matriks tridiagonal

| **Banyak PDS (*N*)** | **Banyak Proses *Wiener* (*M*)** | **Efisiensi** | |
|---|---|---|---|
| | | **2 Prosesor** | **4 Prosesor** |
| **16** | **16** | 0.68 | 0.22 |
| **64** | **16** | 0.69 | 0.23 |
| | **64** | 0.40 | 0.12 |
| **256** | **16** | 0.76 | 0.24 |
| | **64** | 0.70 | 0.24 |
| | **128** | 0.62 | 0.26 |
| | **256** | 0.78 | 0.32 |
| **512** | **16** | 0.72 | 0.24 |
| | **64** | 0.61 | 0.25 |
| | **128** | 0.84 | 0.38 |
| | **256** | 0.95 | 0.52 |
| **1,024** | **16** | 0.72 | 0.30 |
| | **64** | 0.94 | 0.52 |
| | **128** | 0.98 | 0.64 |
| | **256** | 0.88 | 0.62 |
| **2,048** | **16** | 0.82 | 0.42 |
| | **64** | 1.10 | 0.66 |
| | **128** | 0.98 | 0.69 |
| | **256** | 0.87 | 0.59 |
| **4,096** | **16** | 0.90 | 0.47 |
| | **64** | 0.99 | 0.68 |
| | **128** | 0.97 | 0.65 |
| | **256** | 0.89 | 0.57 |

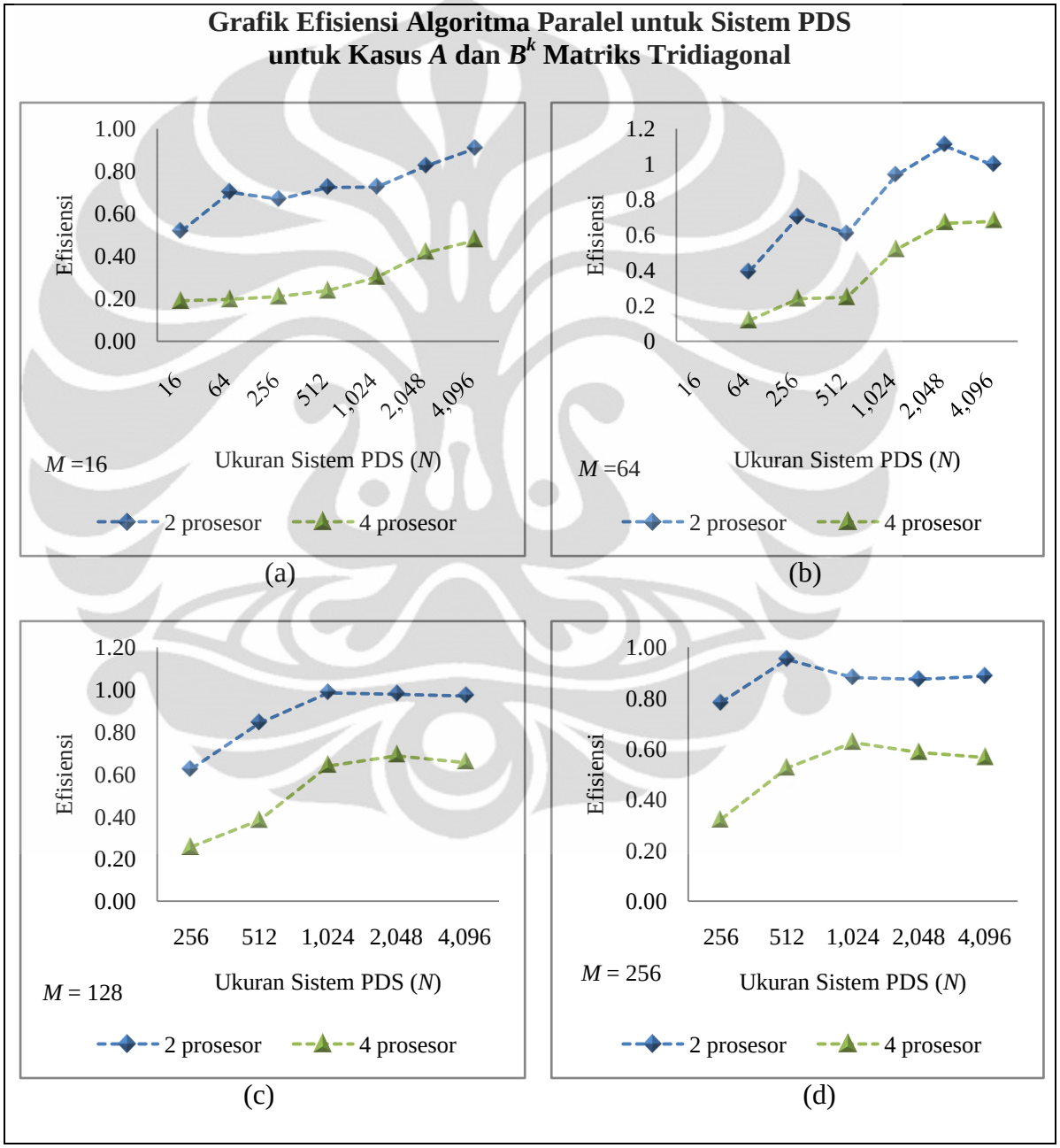
Dari hasil *speed up* dan efisiensi algoritma paralel untuk sistem PDS, akan dibuat grafik *speed up* dan efisiensi yang dikelompokkan berdasarkan banyak proses *Wiener* (*M*). Jadi, terdapat masing-masing 4 grafik *speed up* dan efisiensi dengan $M = 16$, $M = 64$, $M = 128$, dan $M = 256$ (lihat Gambar 4.5). Berdasarkan Gambar 4.5 terlihat bahwa semakin besar ukuran sistem PDS (*N*) maka semakin besar nilai *speed up*. Untuk $M = 16$ diperoleh *speed up* dengan 2 prosesor lebih

besar daripada 4 prosesor hingga $N$ = 1024 (lihat Gambar 4.5a). Untuk $M$ = 64 diperoleh *speed up* dengan 2 prosesor lebih besar daripada 4 prosesor hingga $N$ = 512, sedangkan untuk $N$ = 1024, 2048, dan 4096 lebih efektif apabila menggunakan 4 prosesor (lihat Gambar 4.5b).




Gambar 4.5 Grafik *speed up* algoritma paralel untuk sistem PDS dimana matriks suku deterministik $A$ dan matriks suku stokastik $B^k$ telah ditransformasi menjadi matriks tridiagonal

Untuk $M$ = 128 diperoleh *speed up* dengan 2 prosesor lebih besar daripada 4 prosesor hingga $N$ = 512, sedangkan untuk $N$ = 1024, 2048, dan 4096 jauh lebih efektif apabila menggunakan 4 prosesor (lihat Gambar 4.5c). Untuk $M$ = 256 diperoleh *speed up* dengan 2 prosesor lebih besar daripada 4 prosesor hingga $N$ = 256 saja, sedangkan untuk $N$ = 512,1024, 2048, dan 4096 lebih efektif apabila menggunakan 4 prosesor (lihat Gambar 4.5d).




Gambar 4.6 Grafik efisiensi algoritma paralel untuk sistem PDS dimana matriks suku deterministik $A$ dan matriks suku stokastik $B^k$ telah ditransformasi menjadi matriks tridiagonal

Grafik efisiensi algoritma paralel untuk sistem PDS dapat dilihat pada Gambar 4.6. Berdasarkan Gambar 4.6, terlihat bahwa untuk semua nilai *N* dan *M*, efisiensi dengan menggunakan 2 prosesor lebih baik daripada efisiensi dengan menggunakan 4 prosesor.

Jadi, dapat disimpulkan bahwa untuk *N* yang relatif kecil (kurang dari 512) dan nilai *M* yang relatif kecil (kurang dari 16), maka lebih efektif jika menggunakan 2 prosesor, sedangkan untuk nilai *N* dan *M* yang relatif lebih besar, maka lebih efektif jika menggunakan 4 prosesor.

## 4.3 Implementasi Algoritma Paralel secara *Sample Path* untuk Menghitung Harga Opsi pada Model Harga Opsi *Call* Asia

Pada Subbab 3.5 telah diberikan Algoritma 3.5.1 yaitu algoritma paralel secara *sample path* untuk menghitung harga opsi pada model harga opsi *call* Asia. Program MATLAB pada Lampiran 3 merupakan implementasi dari Algoritma 3.5.1. Input dari program ini adalah *p* (banyak prosesor) dan *H* (banyak simulasi). Program ini bertujuan untuk menguji sejauh mana kinerja paralel dari Algoritma 3.5.1. Pada program ini melakukan simulasi sebanyak *H sample path* dengan $U_0$ = 45, $v_0$ = 0.35, $r$ = 0.06, $E$ = 70, $\rho$ =−0.5, $\kappa$ = 0.1, $\psi$ = 0.5. Pada program ini diambil waktu jatuh tempo opsi adalah 360 hari. Output dari program ini adalah *H sample path* yang dikerjakan oleh $1,2,\ldots,p$ prosesor. Jika dimasukkan input $p = 4$, maka program akan menghitung *H sample path* dengan menggunakan 1 prosesor, 2 prosesor, 3 prosesor, dan 4 prosesor dan menghasilkan output yang sama. Dengan melakukan simulasi sebanyak 5000 *sample path* dengan menggunakan 1 prosesor, 2 prosesor, 3 prosesor, dan 4 prosesor, maka program ini menghasilkan harga opsi 0.7907.

Program ini juga akan menghasilkan *running time* program untuk 1 prosesor, 2 prosesor, 3 prosesor, dan 4 prosesor. Tabel 4.7 adalah hasil *running time* yang telah didapat untuk beberapa nilai *H* tertentu. Berdasarkan hasil *running time* pada Tabel 4.7, maka diperoleh *speed up* algoritma paralel secara *sample path* untuk menghitung harga opsi pada model harga opsi *call* Asia (lihat Tabel 4.8) dan efisiensi algoritma paralel secara *sample path* untuk menghitung harga opsi pada model harga opsi *call* Asia (lihat Tabel 4.9).

Tabel 4.7 Hasil *running time* algoritma paralel secara *sample path* untuk menghitung harga opsi pada model harga opsi *call* Asia

| **Banyak *Sample Path* (*H*)** | ***Running Time* (dalam sekon)** | | | |
|---|---|---|---|---|
| | **1 Prosesor** | **2 Prosesor** | **3 Prosesor** | **4 Prosesor** |
| **100** | 0.588059382 | 0.37819193 | 0.347225288 | 0.328018273 |
| **500** | 1.129014002 | 0.742608705 | 0.716048618 | 0.699550717 |
| **1000** | 1.923098275 | 1.195250037 | 1.046010684 | 0.95551152 |
| **5000** | 7.670213207 | 4.899217308 | 4.338812419 | 3.985213316 |
| **10000** | 15.7543185 | 9.550121678 | 8.289444541 | 7.315445706 |

Grafik *speed up* algoritma paralel secara *sample path* untuk menghitung harga opsi pada model harga opsi *call* Asia dapat dilihat pada Gambar 4.7. Berdasarkan Gambar 4.7 terlihat bahwa semakin banyak *sample path* (*H*) yang dihitung maka nilai *speed up* relatif konstan untuk *H*=100 hingga *H*=10000. *Speed up* tertinggi diperoleh ketika *H*=10000 sebesar 2.15 (lihat Gambar 4.7).

Tabel 4.8 *Speed up* algoritma paralel secara *sample path* untuk menghitung harga opsi pada model harga opsi *call* Asia

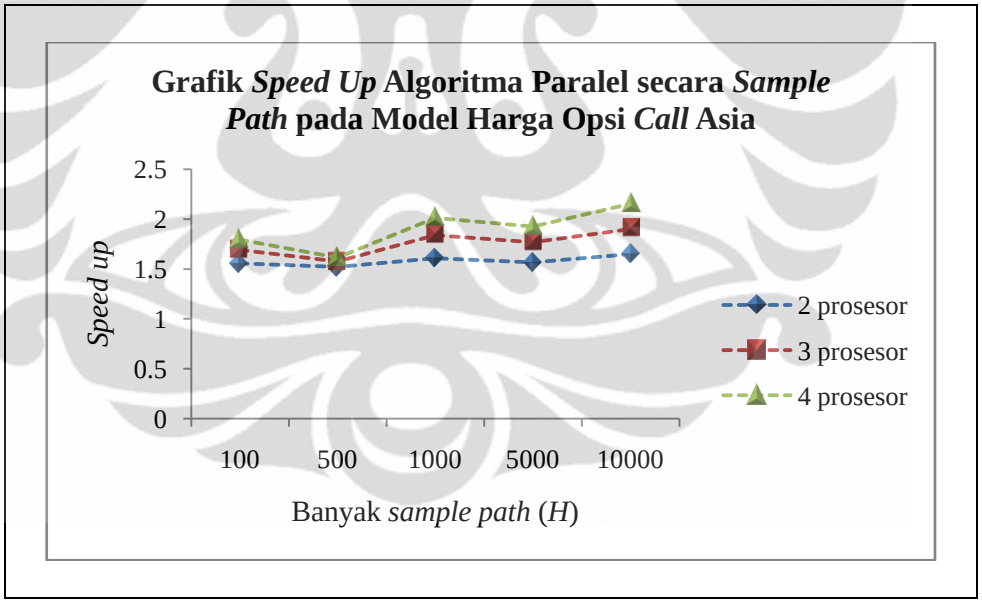
| **Banyak *Sample Path* (*H*)** | ***Speed Up*** | | |
|---|---|---|---|
| | **2 Prosesor** | **3 Prosesor** | **4 Prosesor** |
| **100** | 1.55 | 1.69 | 1.79 |
| **500** | 1.52 | 1.58 | 1.62 |
| **1000** | 1.61 | 1.84 | 2.01 |
| **5000** | 1.57 | 1.77 | 1.92 |
| **10000** | 1.65 | 1.90 | 2.15 |

Tabel 4.9 Efisiensi algoritma paralel secara *sample path* untuk menghitung harga opsi pada model harga opsi *call* Asia

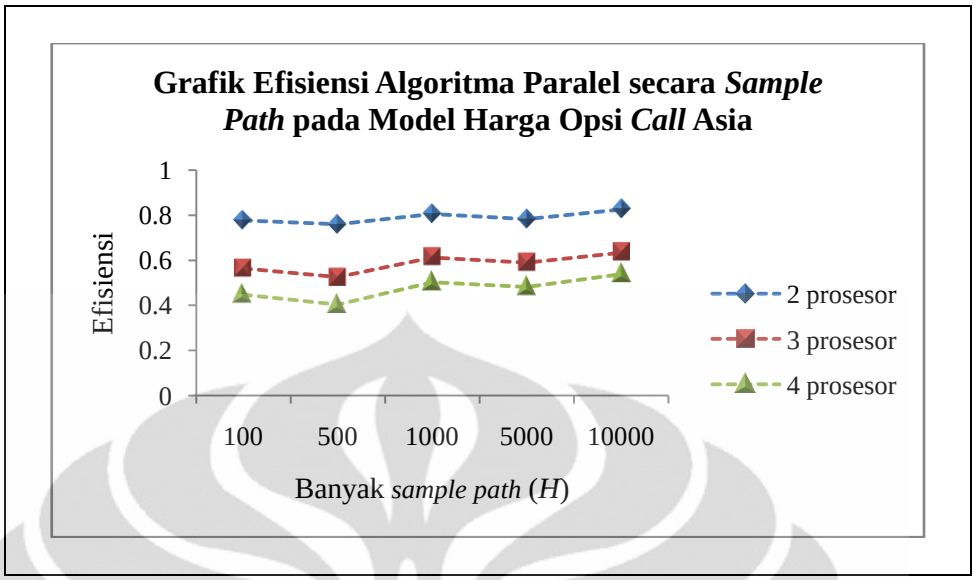
| **Banyak *Sample Path* (*H*)** | **Efisiensi** | | |
|---|---|---|---|
| | **2 Prosesor** | **3 Prosesor** | **4 Prosesor** |
| **100** | 0.78 | 0.56 | 0.45 |
| **500** | 0.76 | 0.53 | 0.40 |
| **1000** | 0.80 | 0.61 | 0.50 |
| **5000** | 0.78 | 0.59 | 0.48 |
| **10000** | 0.82 | 0.63 | 0.54 |

Terlihat bahwa algoritma paralel secara *sample path* untuk menghitung harga opsi pada model harga opsi *call* Asia lebih efektif dengan menggunakan 4 prosesor karena menghasilkan *speed up* yang lebih baik daripada menggunakan 2 atau 3 prosesor (lihat Gambar 4.7).

Grafik efisiensi dari algoritma paralel secara *sample path* untuk menghitung harga opsi pada model harga opsi *call* Asia dapat dilihat pada Gambar 4.8. Berdasarkan Gambar 4.8 terlihat bahwa semakin banyak *sample path* ($H$) yang dihitung maka nilai efisiensi relatif konstan untuk $H$=100 hingga $H$=10000. Terlihat pula bahwa algoritma paralel secara *sample path* untuk menghitung harga opsi pada model harga opsi *call* Asia lebih efisien dengan menggunakan 2 prosesor karena menghasilkan efisiensi yang lebih baik daripada menggunakan 3 atau 4 prosesor (lihat Gambar 4.8).




Gambar 4.7 Grafik *speed up* algoritma paralel secara *sample path* untuk menghitung harga opsi pada model harga opsi *call* Asia




Gambar 4.8 Grafik efisiensi algoritma paralel secara *sample path* untuk menghitung harga opsi pada model harga opsi call Asia

## 4.4 Implementasi Algoritma Paralel secara Titik Diskretisasi untuk Menghitung $\overline{U}$ pada Model Harga Opsi *Call* Asia

Pada Subbab 3.5 telah diberikan Algoritma 3.5.2 yaitu algoritma paralel secara titik diskretisasi untuk menghitung $\overline{U}$ pada model harga opsi *call* Asia. Program MATLAB pada Lampiran 4 merupakan implementasikan dari Algoritma 3.5.2. Input dari program ini adalah $p$ (banyak prosesor) dan $L$ (waktu jatuh tempo). Program ini bertujuan untuk menguji sejauh mana kinerja paralel dari Algoritma 3.5.2. Pada program ini dilakukan 1 simulasi untuk menghitung $\overline{U}$ dengan $U_0 = 45$, $v_0 = 0.35$, $r = 0.06$, $E = 70$, $\rho = -0.5$, $\kappa = 0.1$, $\psi = 0.5$. Pada simulasi ini waktu jatuh tempo opsi dihitung per jam. Dalam hal ini waktu jatuh tempo opsi 1 tahun setara dengan 360*24 jam (titik diskretisasi). Output dari program ini adalah $\overline{U}$ yang dikerjakan oleh $1,2,\ldots,p$ prosesor. Jika dimasukkan input $p = 4$, maka program akan menghitung $\overline{U}$ dengan menggunakan 1 prosesor, 2 prosesor, 3 prosesor, dan 4 prosesor dan menghasilkan output yang sama. Karena hanya digunakan 4 prosesor dan waktu jatuh tempo ($L$) habis dibagi 2, 3, dan 4, maka program MATLAB (lihat Lampiran 4) hanya untuk kasus $p$ habis membagi $L$. Dengan demikian diharapkan akan diperoleh nilai *speed up* yang maksimum karena *load balance* terpenuhi.

Dengan melakukan simulasi untuk waktu jatuh tempo 1 tahun dengan menggunakan 1 prosesor, 2 prosesor, 3 prosesor, dan 4 prosesor, maka program ini menghasilkan $\bar{U} = 39.4876$. Program ini juga akan menghasilkan *running time* program untuk 1 prosesor, 2 prosesor, 3 prosesor, dan 4 prosesor. Tabel 4.10 adalah hasil *running time* yang telah didapat untuk beberapa nilai $L$ tertentu. Berdasarkan hasil *running time* pada Tabel 4.10, maka diperoleh *speed up* algoritma paralel untuk untuk menghitung $\bar{U}$ pada model harga opsi *call* Asia secara titik diskretisasi (lihat Tabel 4.11) dan efisiensi algoritma paralel untuk untuk menghitung $\bar{U}$ pada model harga opsi *call* Asia secara titik diskretisasi (lihat Tabel 4.12).

Tabel 4.10 Hasil *running time* algoritma paralel secara titik diskretisasi untuk menghitung $\bar{U}$ pada model harga opsi *call* Asia

| **Waktu Jatuh Tempo ($L$)** | ***Running Time* (dalam sekon)** | | | |
|---|---|---|---|---|
| | **1 Prosesor** | **2 Prosesor** | **3 Prosesor** | **4 Prosesor** |
| **1 tahun** | 0.46683032 | 0.30297345 | 0.27721173 | 0.23163794 |
| **5 tahun** | 0.79109679 | 0.57969437 | 0.52627982 | 0.47577673 |
| **10 tahun** | 1.12944933 | 0.90710838 | 0.81278272 | 0.77526808 |
| **15 tahun** | 1.56719564 | 1.32357599 | 1.26006677 | 1.12672332 |
| **20 tahun** | 2.14688245 | 1.80381244 | 1.71371747 | 1.50431816 |

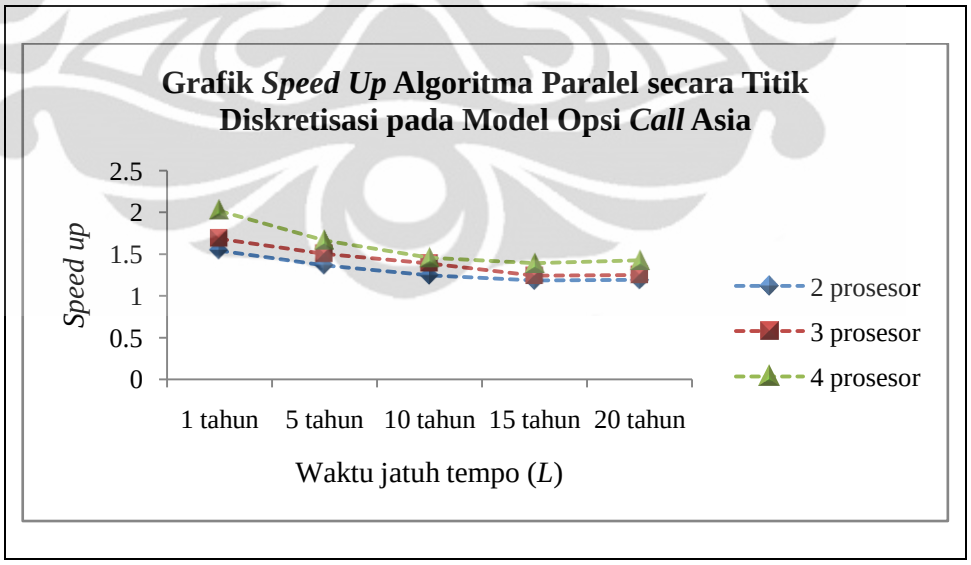
Tabel 4.11 *Speed up* algoritma paralel secara titik diskretisasi untuk menghitung $\bar{U}$ pada model harga opsi *call* Asia

| **Waktu Jatuh Tempo ($L$)** | ***Speed Up*** | | |
|---|---|---|---|
| | **2 Prosesor** | **3 Prosesor** | **4 Prosesor** |
| **1 tahun** | 1.54 | 1.68 | 2.02 |
| **5 tahun** | 1.36 | 1.50 | 1.66 |
| **10 tahun** | 1.25 | 1.39 | 1.46 |
| **15 tahun** | 1.18 | 1.24 | 1.39 |
| **20 tahun** | 1.19 | 1.25 | 1.43 |

Tabel 4.12 Efisiensi algoritma paralel secara titik diskretisasi untuk menghitung $\bar{U}$ pada model harga opsi *call* Asia

| **Waktu Jatuh Tempo ($L$)** | **Efisiensi** | | |
|---|---|---|---|
| | **2 Prosesor** | **3 Prosesor** | **4 Prosesor** |
| **1 tahun** | 0.77 | 0.56 | 0.50 |
| **5 tahun** | 0.68 | 0.50 | 0.42 |
| **10 tahun** | 0.62 | 0.46 | 0.36 |
| **15 tahun** | 0.59 | 0.41 | 0.35 |
| **20 tahun** | 0.60 | 0.42 | 0.36 |

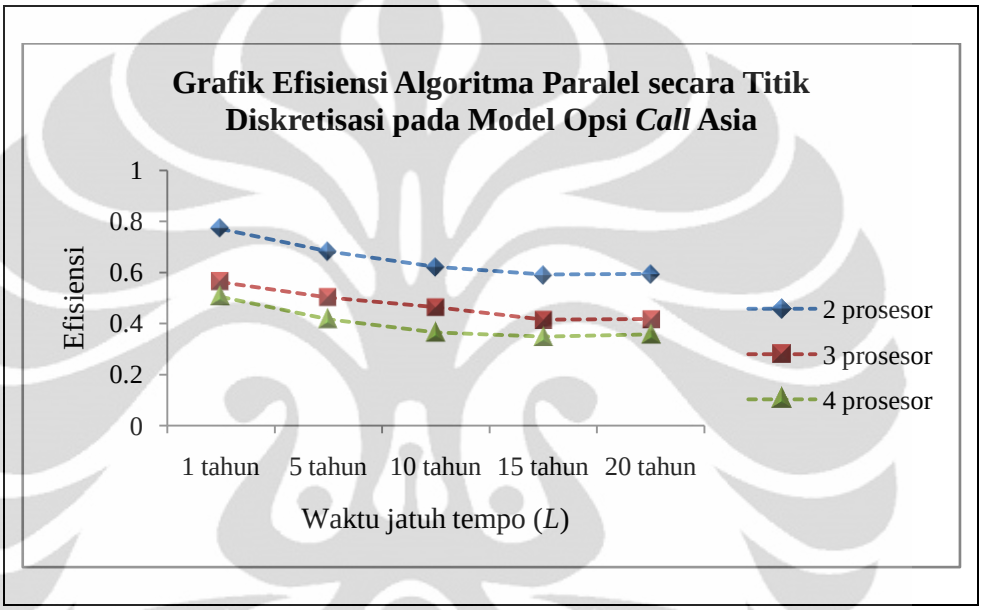
Grafik *speed up* algoritma paralel secara titik diskretisasi untuk menghitung $\bar{U}$ pada model harga opsi *call* Asia dapat dilihat pada Gambar 4.9. Berdasarkan Gambar 4.9 terlihat bahwa semakin lamanya waktu jatuh tempo ($L$) suatu opsi, maka nilai *speed up* cenderung menurun. *Speed up* tertinggi diperoleh ketika waktu jatuh tempo adalah 1 tahun. Terlihat bahwa algoritma paralel secara titik diskretisasi untuk menghitung $\bar{U}$ pada model harga opsi *call* Asia lebih efektif dengan menggunakan 4 prosesor karena menghasilkan *speed up* yang lebih baik daripada menggunakan 2 atau 3 prosesor (lihat Gambar 4.9).




Gambar 4.9 Grafik *speed up* algoritma paralel secara titik diskretisasi untuk menghitung $\bar{U}$ pada model harga opsi *call* Asia

Grafik efisiensi dari algoritma paralel secara titik diskretisasi untuk menghitung $\bar{U}$ pada model harga opsi *call* Asia dapat dilihat pada Gambar 4.10.

Berdasarkan Gambar 4.10 terlihat bahwa semakin lamanya waktu jatuh tempo ($L$) suatu opsi, maka nilai efisiensi cenderung menurun. Terlihat pula bahwa algoritma paralel secara titik diskretisasi untuk menghitung $\overline{U}$ pada model harga opsi *call* Asia lebih efisien dengan menggunakan 2 prosesor karena menghasilkan efisiensi yang lebih baik daripada menggunakan 3 atau 4 prosesor (lihat Gambar 4.10).




Gambar 4.10 Grafik efisiensi algoritma paralel secara titik diskretisasi untuk menghitung $\overline{U}$ pada model harga opsi *call* Asia

# BAB 5
# KESIMPULAN

Pada skripsi ini telah dibangun algoritma paralel secara *sample path* untuk PDS, algoritma paralel untuk sistem PDS dimana matriks koefisien suku deterministik $A$ dan matriks koefisien suku stokastik $B^k$ merupakan matriks tridiagonal, algoritma paralel secara *sample path* untuk menghitung $H$ *sample path* untuk mendapatkan harga opsi *call* Asia, dan algoritma paralel secara titik diskretisasi untuk menghitung $\bar{U}$ pada model harga opsi *call* Asia. Algoritma-algoritma tersebut telah diimplementasikan dengan program MATLAB. Simulasi program-program tersebut dengan menggunakan MATLAB dan *Parallel Computing Toolbox*-nya telah memberikan *speed up* dan efisiensi yang cukup baik. Secara umum, program-program ini dapat dijalankan untuk sembarang $p$ prosesor. Namun karena keterbatasan waktu dan mesin *multicore* (*laptop*), maka hanya menggunakan maksimum 4 prosesor. Hasil yang diperoleh pada skripsi ini adalah sebagai berikut:

1. Algoritma paralel secara *sample path* untuk PDS.
   - Program MATLAB sebagai implementasi dari algoritma paralel secara *sample path* untuk PDS.
   - *Speed up* dari algoritma paralel secara *sample path* untuk PDS untuk banyaknya simulasi adalah 1000, 5000, 10000, 15000, 20000, dan 25000 diperoleh
     - untuk 2 prosesor terletak antara 1.77 hingga 2.03,
     - untuk 3 prosesor terletak antara 2.23 hingga 2.55,
     - untuk 4 prosesor terletak antara 2.30 hingga 2.67.
   - Sedangkan efisensi dari algoritma paralel secara *sample path* untuk PDS untuk banyaknya simulasi adalah 1000, 5000, 10000, 15000, 20000, dan 25000 diperoleh
     - untuk 2 prosesor terletak antara 0.88 hingga 1.01,
     - untuk 3 prosesor terletak antara 0.74 hingga 0.85,

- untuk 4 prosesor terletak antara 0.57 hingga 0.67.

➢ Algoritma paralel secara *sample path* untuk PDS lebih efektif dengan menggunakan 4 prosesor karena menghasilkan waktu komputasi yang lebih cepat daripada menggunakan 2 atau 3 prosesor. Akan tetapi, efisiensi terbaik diperoleh ketika menggunakan 2 prosesor.

2. Pada aplikasi yang melibatkan sistem PDS seringkali dijumpai matriks suku deterministik $A$ dan matriks suku stokastik $B^k$ merupakan matriks penuh, maka menurut (Bai & Ward, 2008), dengan menggunakan transformasi *Householder*, maka suatu matriks penuh dapat ditransformasi ke bentuk matriks tridiagonal. Oleh karena itu, matriks penuh $A$ dan $B^k$ yang terdapat pada sistem PDS dapat ditransformasi ke bentuk matriks tridiagonal. Pada skripsi ini, transformasi *Householder* tidak dibahas, tetapi diasumsikan bahwa telah diterapkan transformasi *Householder* pada matriks penuh $A$ dan $B^k$ sehingga $A$ dan $B^k$ yang digunakan pada algoritma paralel untuk sistem PDS merupakan matriks tridiagonal. Jadi, algoritma paralel untuk sistem PDS yang dibangun pada skripsi ini adalah untuk kasus $A$ dan $B^k$ adalah matriks tridiagonal.
   ➢ Program MATLAB sebagai implementasi dari algoritma paralel untuk sistem PDS.
   ➢ *Speed up* dari algoritma paralel untuk sistem PDS berdasarkan program MATLAB tersebut untuk banyaknya PDS dalam sistem adalah 16, 64, 256, 512, 1024, 2048, dan 4096 serta banyaknya proses *Wiener* yang saling bebas dalam sistem adalah 16, 64, 128, dan 256 diperoleh
      - untuk 2 prosesor terletak antara 0.79 hingga 2.21,
      - untuk 4 prosesor terletak antara 0.47 hingga 2.75.
   ➢ Sedangkan efisiensi dari algoritma paralel untuk sistem PDS berdasarkan program MATLAB tersebut untuk banyaknya PDS dalam sistem adalah 16, 64, 256, 512, 1024, 2048, dan 4096 serta banyaknya proses *Wiener* yang saling bebas dalam sistem adalah 16, 64, 128, dan 256 diperoleh

- untuk 2 prosesor terletak antara 0.40 hingga 1.10,
- untuk 4 prosesor terletak antara 0.12 hingga 0.68.

➢ Untuk *N* yang relatif kecil (kurang dari 512) dan nilai *M* yang relatif kecil (kurang dari 16), maka algoritma paralel untuk sistem PDS lebih efektif jika menggunakan 2 prosesor karena menghasilkan waktu komputasi yang lebih cepat daripada menggunakan 3 atau 4 prosesor, sedangkan untuk nilai *N* dan *M* yang relatif lebih besar, maka algoritma paralel untuk sistem PDS lebih efektif jika menggunakan 4 prosesor karena menghasilkan waktu komputasi yang lebih cepat daripada menggunakan 2 atau 3 prosesor.

3. Algoritma paralel secara *sample path* untuk menghitung harga opsi pada model harga opsi *call* Asia.
   ➢ Program MATLAB sebagai implementasi dari algoritma paralel secara *sample path* untuk menghitung harga opsi pada model harga opsi *call* Asia.
   ➢ *Speed up* dari algoritma paralel secara *sample path* untuk menghitung harga opsi pada model harga opsi *call* Asia berdasarkan program MATLAB tersebut untuk banyaknya simulasi adalah 100, 500, 1000, 5000, dan 10000 diperoleh
      - untuk 2 prosesor terletak antara 1.52 hingga 1.65,
      - untuk 3 prosesor terletak antara 1.58 hingga 1.90,
      - untuk 4 prosesor terletak antara 1.62 hingga 2.15.
   ➢ Sedangkan efisiensi dari algoritma paralel secara *sample path* untuk menghitung harga opsi pada model harga opsi *call* Asia berdasarkan program MATLAB tersebut untuk banyaknya simulasi adalah 100, 500, 1000, 5000, dan 10000 diperoleh
      - untuk 2 prosesor terletak antara 0.76 hingga 0.82,
      - untuk 3 prosesor terletak antara 0.53 hingga 0.63,
      - untuk 4 prosesor terletak antara 0.40 hingga 0.54.
   ➢ Pada simulasi ini, waktu jatuh tempo opsi adalah 360 hari. Algoritma paralel secara *sample path* untuk menghitung harga opsi pada model harga

opsi *call* Asia lebih efektif dengan menggunakan 4 prosesor karena menghasilkan waktu komputasi yang lebih cepat daripada menggunakan 2 atau 3 prosesor. Akan tetapi, efisiensi terbaik diperoleh ketika menggunakan 2 prosesor.

4. Algoritma paralel secara titik diskretisasi untuk menghitung $\overline{U}$ pada model harga opsi *call* Asia.
   - Program MATLAB sebagai implementasi dari algoritma paralel secara titik diskretisasi untuk menghitung $\overline{U}$ pada model harga opsi *call* Asia.
   - *Speed up* dari algoritma paralel secara titik diskretisasi untuk menghitung $\overline{U}$ pada model harga opsi *call* Asia berdasarkan program MATLAB tersebut untuk waktu jatuh tempo ($L$) opsi adalah 1 tahun, 5 tahun, 10 tahun, 15 tahun, dan 20 tahun diperoleh
     - untuk 2 prosesor terletak antara 1.18 hingga 1.54,
     - untuk 3 prosesor terletak antara 1.24 hingga 1.68,
     - untuk 4 prosesor terletak antara 1.39 hingga 2.02.
   - Sedangkan efisiensi dari algoritma paralel secara titik diskretisasi untuk menghitung $\overline{U}$ pada model harga opsi *call* Asia berdasarkan program MATLAB tersebut untuk waktu jatuh tempo ($L$) opsi adalah 1 tahun, 5 tahun, 10 tahun, 15 tahun, dan 20 tahun diperoleh
     - untuk 2 prosesor terletak antara 0.59 hingga 0.77,
     - untuk 3 prosesor terletak antara 0.41 hingga 0.56,
     - untuk 4 prosesor terletak antara 0.42 hingga 0.50.
   - Waktu jatuh tempo opsi pada simulasi ini dihitung per jam. Berdasarkan hasil simulasi, semakin lamanya waktu jatuh tempo ($L$) suatu opsi, maka nilai *speed up* cenderung menurun. Algoritma paralel secara titik diskretisasi untuk menghitung $\overline{U}$ pada model harga opsi *call* Asia lebih efektif dengan menggunakan 4 prosesor karena menghasilkan waktu komputasi yang lebih cepat daripada menggunakan 2 atau 3 prosesor. Akan tetapi, efisiensi terbaik diperoleh ketika menggunakan 2 prosesor.

Berdasarkan hasil-hasil yang telah diperoleh pada skripsi ini, komputasi paralel dapat mempercepat proses komputasi dalam mendapatkan solusi dari suatu model PDS tersebut. Algoritma paralel yang telah diberikan pada skripsi ini diharapkan dapat diterapkan pada berbagai bidang aplikasi. Karena keterbatasan waktu, maka pada skripsi ini algoritma paralel hanya diterapkan pada bidang keuangan.

Penulis menyadari bahwa masih banyak kekurangan dalam skripsi ini. Salah satu diantaranya adalah pada saat membahas algoritma paralel untuk sistem PDS yang belum meneliti bahwa hasil solusi EM yang diperoleh setelah $A$ dan $B^k$ ditranformasi *Householder* kemudian diterapkan algoritma paralel untuk sistem PDS, memiliki hasil solusi EM yang sama (atau memiliki error yang kecil) apabila menggunakan algoritma dengan $A$ dan $B^k$ sebelum dilakukan transformasi *Householder*. Hal ini diharapkan dapat dijadikan topik untuk penelitian lebih lanjut khusunya penelitian di Departemen Matematika FMIPA UI.

# DAFTAR PUSTAKA


Bai, Y., & Ward, R. (2008). Parallel Block Tridiagonalization of Real Symmetric Matrices. *Journal of Parallel and Distributed Computing* , 703-715.

Barney, B. (2010). *Introduction to Parallel Computing*. Retrieved Oktober 21, 2010, from High Performance Computing: https://computing.llnl.gov/tutorials/parallel_comp/

Burkardt, J. (2009, May 26-28). *MATLAB Parallel Computing*. Retrieved September 18, 2010, from The Florida State University: http://people.sc.fsu.edu/~burkardt/presentations/fdi 2009 matlab.pdf

Elliot, R., & Madan, D. (1998). A Discrete Time Equivalent Martingale Measure. *An International Journal of Mathematics, Statistics, and Financial Economics* , 127-152.

Evans, L. C. (2003). *An Introduction to Stochastic Differential Equations Version 1.2.* Berkeley: Citeseer.

Fouque, J., Papanicolaou, G., & Sircar, K. (2000). Mean-Reverting Stochastic Volatility. *International Journal of Theoretical and Applied Finance* , 101-142.

Higham, D. J. (2001). An Algorithmic Introduction to Numerical Simulation of Stochastic Differential Equations. *SIAM REVIEW Vol. 43 No. 3* , 525–546.

Higham, D., & Kloeden, P. (2002). MAPLE and MATLAB for Stochastic Differential Equations in Finance . In S. Nielsen, *Programming Languages and Systems in Computational Economics and Finance* (pp. 233-269). Dordrecht: Kluwer Academic Publishers.

Kanniainen, J., & Piche, R. (2009). Use of Distributed Computing in Derivative Pricing. *International Journal of Electronic Finance Vol.3 No.3* , 270-283.

Luszczek, P. (2009). Parallel Programming in MATLAB. *The International Journal of High Performance Computing Applications* , 277-283.

Sharma, G., & Martin, J. (2009). MATLAB: A Language for Parallel Computing. *International Journal of Parallel Programming* , 3–36.

Wilkinson, B., & Allen, M. (1999). *Parallel Programming: Techniques and Applications using Networked Workstations and Parallel Computers.* New Jersey: Prentice Hall.

# LAMPIRAN

Lampiran 1 *Listing* Program Implementasi Algoritma Paralel secara *sample path* untuk PDS

```
clear all;

p = input(' Masukkan banyaknya prosesor yang tersedia : ');
S = input(' Masukkan banyaknya simulasi               : ');
fprintf ('\n');

T=1; L=2^8; dt=T/L;
lambda=2; mu=1; Xzero=1;

randn('state',1)
dW = sqrt(dt)*randn(S,L);
Xem = zeros(S,L);

lab_num_array = (1:1:p);
runtime = zeros(1,p);

for lab_num = lab_num_array(1:p)

  matlabpool('open',lab_num)

  tic;
  spmd
    for i = labindex : numlabs : lab_num
      if mod(S,lab_num) ~= 0
        X_em = zeros(floor(S/lab_num),L);
        for e = 1 : mod(S,lab_num)
          if i == e
            Winc_end = dW(lab_num*floor(S/lab_num)+e,:);
            Xtemp = Xzero;
            for j = 1:L
              Xtemp = Xtemp + lambda*Xtemp*dt +
                      mu*Xtemp*Winc_end(j);
              X_em(floor(S/lab_num)+1,j) = Xtemp;
            end
          end
        end
      else
        X_em = zeros(floor(S/lab_num),L);
      end

      Winc = dW((i-1)*floor(S/lab_num) + 1:i*floor(S/lab_num),:);
      for h = 1:floor(S/lab_num)
        Xtemp = Xzero;
        for j = 1:L
          Xtemp = Xtemp + lambda*Xtemp*dt + mu*Xtemp*Winc(h,j);
          X_em(h,j) = Xtemp;
```

```
        end
      end
    end
  end
  runtime(lab_num) = toc;

  for i=1:lab_num
    X = X_em{i};
    if i <= mod(S,lab_num)
      Xem((i-1)*floor(S/lab_num) + 1:i*floor(S/lab_num),:) =
                                          X(1:floor(S/lab_num),:);
      Xem(lab_num*floor(S/lab_num)+i,:) = X(end,:);
    else
      Xem((i-1)*floor(S/lab_num) + 1:i*floor(S/lab_num),:) =
                                          X(1:floor(S/lab_num),:);
    end
  end

  matlabpool('close')
  fprintf ('\n');

  figure(lab_num)
  plot([0:dt:T],[repmat(Xzero,5,1),Xem(1:5,:)],'r-'), hold on
  title('Plot Hasil Simulasi Algoritma Paralel untuk PDS secara
        Sample Path')
  xlabel('t','Fontsize',10)
  ylabel('X(t)','Fontsize',10,'Rotation',0,'HorizontalAlignment',
        'right')
  legend('5 sample path pertama',2)

end

fprintf('     labs      runtime \n');
for i=1:p
  fprintf('       %d      %.8f \n',i,runtime(i));
end
fprintf ('\n');
```

Lampiran 2 *Listing* Program Implementasi Algoritma Paralel untuk Sistem PDS

```
clear all;

p = input(' Masukkan banyaknya prosesor yang tersedia : ');
N = input(' Masukkan banyaknya PDS dalam sistem       : ');
W = input(' Masukkan banyaknya proses Wiener          : ');
fprintf ('\n');

n(1) = 2;
count = 1;
while n(count) < floor(N/2)
  n(count+1) = n(count)*2;
  count = count + 1;
end

sign = 1;
for z=1:count
  if mod(N,n(z)) ~= 0
    fprintf(' Maaf input salah!\n Banyaknya PDS dalam sistem dan
             blok diagonal harus 2^n');
    fprintf (' \n Coba ulangi lagi ya..\n\n');
    sign = 0;
  end
  if sign == 0
    break
  end
end

if sign == 0
  break
end

T=1; L=2^8; dt=T/L;

runtime = zeros(size(W,2),p);

lab_num_array(1) = 1;
c = 1;
while lab_num_array(c) < p
  lab_num_array(c+1) = lab_num_array(c)*2;
  c = c + 1;
end

s = L+1;
plot_angka = randi(N,1,1);

for lab_num = lab_num_array

  matlabpool('open',lab_num)

  r = 1;
  for M = W
    Xzero=ones(N,1);
```

```
randn('state',1)
dW = sqrt(dt)*randn(M,L);
a1 = sparse(1:N,1:N,randn(1,N));
a2 = sparse(2:N,1:N-1,randn(1,N-1),N,N);
a3 = sparse(1:N-1,2:N,randn(1,N-1),N,N);
A = a1 + a2 + a3;
for m=1:M
  b1 = sparse(1:N,1:N,randn(1,N));
  b2 = sparse(2:N,1:N-1,randn(1,N-1),N,N);
  b3 = sparse(1:N-1,2:N,randn(1,N-1),N,N);
  B(m).diff = b1 + b2 + b3;
end

tic;
spmd
for i = labindex : numlabs : lab_num
  X_em = zeros(floor(N/lab_num),L+1);
  X1 = zeros;
  X2 = zeros;
  Y1 = zeros(1,M);
  Y2 = zeros(1,M);
  if lab_num > 1
    if i ~= lab_num
      X1 = A(i*floor(N/lab_num),i*floor(N/lab_num)+1)*
           Xzero(i*floor(N/lab_num)+1);
      for k = 1:M
        Y1(1,k) = B(k).diff(i*floor(N/lab_num),
                  i*floor(N/lab_num)+1)*Xzero(i*floor(N/lab_nu
                  m)+1);
      end
    end
    if i ~= 1
      X2 = A((i-1)*floor(N/lab_num)+1,(i-1)*
           floor(N/lab_num))*Xzero((i-1)*
           floor(N/lab_num));
      for k = 1:M
        Y2(1,k) = B(k).diff((i-1)*floor(N/lab_num)+1,(i-1)*
        floor(N/lab_num))*Xzero((i-1)*floor(N/lab_num));
      end
    end
  end
  A_local = A((i-1)*floor(N/lab_num) + 1:i*floor(N/lab_num),
  (i-1)*floor(N/lab_num) + 1:i*floor(N/lab_num));
  X_zero = Xzero((i-1)*floor(N/lab_num) + 1:
           i*floor(N/lab_num));
  X_em(1:floor(N/lab_num),1) = X_zero;
  for j = 1:L
    B_local = sparse(zeros(floor(N/lab_num),M));
    for k = 1:M
      B_local(:,k) = B(k).diff((i-1)*floor(N/lab_num)+1:
                     i*floor(N/lab_num),(i-1)*floor(N/lab_num)
                     +1:i*floor(N/lab_num))*X_em
                     (1:floor(N/lab_num),j);
    end
    B_local(floor(N/lab_num),:) =
                               B_local(floor(N/lab_num),:)+Y1;
    B_local(1,:) = B_local(1,:) + Y2;
    Atemp = A_local*X_em(1:floor(N/lab_num),j);
    Atemp(floor(N/lab_num)) = Atemp(floor(N/lab_num)) + X1;
```

```
          Atemp(1) = Atemp(1) + X2;
          X_em(1:floor(N/lab_num),j+1) =
          X_em(1:floor(N/lab_num),j) + Atemp*dt + B_local*dW(:,j);
          if lab_num > 1
            if i ~= lab_num
              labSend(X_em(floor(N/lab_num),j+1),i+1,j)
              X_em_1 = labReceive(i+1,j);
              X1 = A(i*floor(N/lab_num),i*floor(N/lab_num)+1)*
                   X_em_1;
              for k = 1:M
                Y1(1,k) = B(k).diff(i*floor(N/lab_num),
                          i*floor(N/lab_num)+1)*X_em_1;
              end
            end
            if i ~= 1
              labSend(X_em(1,j+1),i-1,j)
              X_em_2 = labReceive(i-1,j);
              X2 = A((i-1)*floor(N/lab_num)+1,(i-1)*
              floor(N/lab_num))*X_em_2;
              for k = 1:M
                Y2(1,k) = B(k).diff((i-1)*floor(N/lab_num)+1,
                          (i-1)*floor(N/lab_num))*X_em_2;
              end
            end
          end
        end
      end
    end
    runtime(r,lab_num) = toc;
    for i=1:lab_num
      Xem((i-1)*floor(N/lab_num) + 1:i*floor(N/lab_num),:)=X_em{i};
    end
    figure(lab_num)
    plot([0:dt:T],[Xem(plot_angka,:)],'r-'), hold on
    title('Plot Hasil Simulasi Algoritma Paralel untuk Sistem
          PDS')
    xlabel('t','Fontsize',10)
    ylabel('X(t)','Fontsize',10,'Rotation',0,'HorizontalAlignment',
          'right')
    r = r + 1;
    end

    matlabpool('close')
    fprintf ('\n');
end

N_write = xlswrite('running_time_tridiag.xlsx',N,1,'B4');
R_write = xlswrite('running_time_tridiag.xlsx',runtime,1,'D4');
W_write = xlswrite('running_time_tridiag.xlsx',W',1,'C4');
```

Lampiran 3 *Listing* Program Implementasi Algoritma Paralel secara *Sample Path* untuk Menghitung Harga Opsi pada Model Harga Opsi *Call* Asia

```
clear all;

p = input(' Masukkan banyaknya prosesor yang tersedia : ');
M = input(' Masukkan banyaknya simulasi               : ');
fprintf ('\n');

S0=45; v0=0.35 ;r=0.06 ; E=70;
rho=-0.5 ;kappa=0.1  ;psi=0.5;

T=1; N=360; dt=T/N;

randn('state',1)

W = randn(M,N);
Y = randn(M,N);
Z = rho*W + sqrt(1-rho^2)*Y;

lab_num_array = (1:1:p);
runtime = zeros(1,p);

for lab_num = lab_num_array(1:p)

  matlabpool('open',lab_num)

  S = zeros(M,N);
  task = floor(M/lab_num);

  tic;
  spmd
    for i = labindex : numlabs : lab_num
      if mod(M,lab_num) ~= 0
        X_local = zeros(task+1,N);
        for e = 1 : mod(M,lab_num)
          if i == e
            W_end = W(lab_num*task+e,:);
            Z_end = Z(lab_num*task+e,:);
            v = v0;
            for j = 1:N-1
              X_local(task+1,j+1) = X_local(task+1,j)+(r-0.5*v)*dt
                                    + sqrt(v*dt)*W_end(j);
              v = max(0,v + kappa*(v0-v)*dt+psi*
                  sqrt(v*dt)*Z_end(j));
            end
          end
        end
      else
        X_local = zeros(task,N);
      end

      W_local = W((i-1)*task + 1:i*task,:);
```

```
        Z_local = Z((i-1)*task + 1:i*task,:);
        for s = 1:task
          v = v0;
          for j = 1:N-1
            X_local(s,j+1) = X_local(s,j)+(r-0.5*v)*dt+ sqrt(v*dt)*
                             W_local(s,j);
            v = max(0,v + kappa*(v0-v)*dt+psi*sqrt(v*dt)*
                Z_local(s,j));
          end
        end
        S_local = S0*exp(X_local);
      end
    end
    runtime(lab_num) = toc;

    for i=1:lab_num
      temp = S_local{i};
      if i <= mod(M,lab_num)
        S((i-1)*task + 1:i*task,:)=temp(1:task,:);
        S(lab_num*task+i,:)=temp(end,:);
      else
        S((i-1)*task + 1:i*task,:)=temp(1:task,:);
      end
    end

    S_bar = sum(S,2)/N;
    price = exp(-r*T)*sum(max(S_bar-repmat(E,M,1),0))/M

    matlabpool('close')
    fprintf ('\n');
  end

  M_write = xlswrite('running_time_opsi_path.xlsx',M,1,'C4');
  R_write = xlswrite('running_time_opsi_path.xlsx',runtime,1,'D4');
```

Lampiran 4 *Listing* Program Implementasi Algoritma Paralel secara Titik Diskretisasi untuk Menghitung $\bar{U}$ pada Model Harga Opsi *Call* Asia

```
clear all;

p = input(' Masukkan banyaknya prosesor yang tersedia : ');
N = input(' Masukkan lamanya waktu jatuh tempo        : ');
fprintf ('\n');

S0=45; v0=0.35 ;r=0.06 ; E=70;
rho=-0.5 ;kappa=0.1  ;psi=0.5;

T=1; dt=T/N;

randn('state',1);

W = randn(1,N);
Y = randn(1,N);
Z = rho*W + sqrt(1-rho^2)*Y;

v = zeros(1,N);
dX = zeros(1,N);

lab_num_array = (1:1:p);
runtime = zeros(1,p);

for lab_num = lab_num_array(1:p)

  v(1) = v0;
  for j = 1:N-2
    v(j+1) = max(0,v(j)+kappa*(v0-v(j))*dt+psi*sqrt(v(j)*dt)*
             Z(j));
  end

  task = floor((N-1)/lab_num);

  matlabpool('open',lab_num)

  tic;
  spmd
    for i = labindex : numlabs : lab_num
      W_local = W((i-1)*task+1:i*task);
      v_local = v((i-1)*task+1:i*task);
      for j = 1:task
        d_X(j) = (r-0.5*v_local(j))*dt+sqrt(v_local(j)*dt)*
                 W_local(j);
      end
    end
  end
  runtime(lab_num) = toc;

  for i=1:lab_num
    temp = d_X{i};
```

```
    if i <= mod(N-1,lab_num)
      dX((i-1)*task+2:i*task+1)=temp(1:task);
      dX(lab_num*task+1+i)=temp(end);
    else
      dX((i-1)*task+2:i*task+1)=temp(1:task);
    end
  end

  matlabpool('close')

  X = cumsum(dX);
  S = S0*exp(X);
  S_bar = sum(S)/N

  fprintf ('\n');
end

N_write = xlswrite('running_time_opsi_time.xlsx',N,1,'C4');
R_write = xlswrite('running_time_opsi_time.xlsx',runtime,1,'D4');
```