KOMPUTASI PARALEL PERSAMAAN DIFUSI NEUTRON PADA REAKTOR CEPAT MENGGUNAKAN ITERASI JACOBI

SKRIPSI

Fatmi Sastri
07 135 030

JURUSAN FISIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS ANDALAS
PADANG 2011

ABSTRAK

Perhitungan persamaan difusi neutron 2-dimensi telah berhasil diselesaikan secara paralel dengan menggunakan metoda Jacobi. Program ditulis dalam bahasa C++ dengan bantuan program Intel *Threading Building Blocks*. Dari proses *benchmarking* nilai fluks neutron dengan program FI-ITBCHI diperoleh bahwa kode program yang dibuat valid dengan akurasi hingga 5 angka penting. Hasil running program menunjukkan bahwa program yang dibuat bersifat *scalable*, baik pada prosesor *single-core* maupun *multi-core*. Untuk prosesor dengan jumlah *core* 2, dicapai nilai *speedup* 1,51 sedangkan untuk prosesor dengan jumlah *core* 4 nilai *speedup*-nya adalah 3,09.

Kata kunci : komputasi paralel, metoda Jacobi, persamaan difusi, *speedup*

ABSTRACT

The calculation of 2-dimensional neutron diffusion equation has been succesfully done parallely using Jacobi method. The program is written in C++ language by the help of Intel Threading Building Blocks. As a result of benchmarking process of neutron flux values by FI-ITBCHI, the written program is valid within 5 significant figures. The result of the running program showed that the written program is scalable, both on single-core processors and multi-core processors. The value of speedup is comparable to the number of cores used. For core 2 processor, the achieved speedup is 1.51 and for processors with 4 cores the speedup is 3.09.

Key words: diffusion equation, Jacobi method, parallel computation, speedup

KATA PENGANTAR

Alhamdulillahirabbil'alamin, puji dan syukur bagi Allah SWT atas segala rahmat, karunia dan petunjuk-Nya kepada penulis. Hanya karena pertolongan dan izinnyalah penulis dapat menyelesaikan skripsi ini yang berjudul "Komputasi Paralel Persamaan Difusi Neutron Pada Reaktor Cepat Menggunakan Iterasi Jacobi". Shalawat dan salam semoga tercurah kepada Nabi Muhammad SAW sebagai tauladan untuk menuju jalan yang benar.

Pada kesempatan ini penulis menyampaikan terimakasih banyak kepada Bapak Dr. Imam Taufiq yang senantiasa membimbing dan memberi motivasi kepada penulis dalam pelaksanaan penelitian hingga penulisan skripsi ini. Selanjutnya ucapan terimakasih penulis tujukan kepada Ibu Dr. Dian Fitriyani, Bapak Afdal, M.Si. dan Ibu Meqorry Yusfi, M.Si. selaku tim penguji yang telah memberikan kritik dan saran, sehingga skripsi ini dapat penulis selesaikan dan terimakasih juga kepada rekan-rekan yang telah membantu dalam pelaksanaan penelitian dan pembuatan skripsi ini.

Penulis menyadari bahwa skripsi ini masih jauh dari kesempurnaan. Oleh karena itu, kritik dan saran yang membangun sangat diperlukan untuk kelengkapan skripsi ini. Akhir kata semoga skripsi ini dapat bermanfaat untuk perkembangan ilmu fisika, khususnya dibidang fisika komputasi.

Padang, Oktober 2011

Penulis

DAFTAR ISI

Halaman

DAFTAR TABEL

Halaman

DAFTAR GAMBAR

Halaman

DAFTAR LAMPIRAN

Halaman

BAB I

PENDAHULUAN

1.1 Latar Belakang

Dengan perkembangan teknologi sekarang ini, telah diciptakan program-program komputasi yang dapat mempermudah dalam melakukan perhitungan. Program yang tersebut baik dalam ukuran komputasi yang mudah atau kecil maupun komputasi yang rumit besar. Program yang menggunakan komputasi yang kecil seperti animasi pergerakan gelombang, animasi sdan perhitungan tumbukan bola dan lain sebagainya. Program komputasi yang rumit dan berat seperti komputasi reaktor, dinamika fluida, simulasi tsunami dan sebagainya. Untuk program dengan ukuran skala komputasi yang kecil dan mudah ini bisa dijalankan pada PC maupun sedangkan untuk komputasi yang rumit pada umumnya menggunakan superkomputer.

Dengan perkembangan teknologi mikroprosesor, PC maupun laptop memiliki prosesor yang lebih dari satu atau istilahnya adalah *multicore processor*. Pada saat ini piranti dengan *multicore processor* tersebut lebih banyak digunakan untuk aplikasi-aplikasi yang memerlukan banyak proses komputasi seperti *video processing* dan *gaming*. Agar dapat memanfaatkan keuntungan dari kemampuan komputasinya, *software* atau program harus berbasis algoritma paralel (pemprograman paralel).

Program-program yang ada selama ini kebanyakan hanya bekerja pada satu prosesor saja sedangkan prosesor yang lainnya tidak bekerja (*idle*). Sehingga

proses eksekusi programnya akan membutuhkan waktu yang lama. Untuk dapat menggunakan semua prosesor yang ada, dibutuhkan pemprograman paralel agar dapat mempermudah dan mempercepat proses jalannya program tersebut.

Sekarang ini telah disediakan oleh perusahaan *software* Intel sebuah program bantu yang dapat digunakan pada program C++ yaitu Intel *Threading Building Blocks* atau disingkat dengan Intel TBB. Intel TBB ini adalah kumpulan *template* yang digunakan untuk mempermudah memparalelisasi program. Dengan adanya template ini paralelisasi tidak lagi sulit.

Metode Jacobi merupakan salah satu dari metode iterasi yang dapat dengan mudah diparalelisasikan, dibandingkan dengan metode iterasi lainnya. Metoda Jacobi ini dapat membantu penyelesaian persamaan-persamaan yang cukup sulit seperti perhitungan persamaan difusi.

Dalam fisika, persamaan difusi secara umum diselesaikan dengan persamaan transport Boltzman. Persamaan ini bisa mengakomodasi semua proses dalam difusi. Tetapi persamaan ini cukup rumit untuk diselesaikan. Dengan pendekatan persamaan transport Boltzmann bisa disederhanakan dengan beberapa pendekatan dan keterbatasan. Untuk kasus difusi neutron digunakan persamaan difusi neutron. Persamaan difusi neutron diturunkan dari konsep keseimbangan jumlah neutron yang lahir dan hilang dalam teras reaktor nuklir.

Dalam analisis reaktor, perhitungan persamaan difusi neutron seringkali dilakukan secara berulang-ulang. Karenanya secara keseluruhan waktu perhitungan persamaan difusi neutron menjadi cukup besar. Dengan demikian

percepatan perhitungan persamaan difusi ini akan sangat mempercepat analisis reaktor secara keseluruhan.

Untuk dapat memperoleh nilai distribusi fluks neutron dengan persamaan difusi, maka geometri teras reaktor akan dibagi menjadi bagian-bagian (mesh) kecil. Persamaan difusi ini akan di integrasi dan di diskritasikan sehingga persamaannya akan menjadi bentuk matrik. Matrik inilah yang akan diselesaikan menggunakan metode iterasi Jacobi, dimana metode Jacobi merupakan metode iterasi yang mudah diparalelisasikan dibandingkan metoda iterasi lainnya.

1.2 Perumusan Masalah

Perhitungan difusi neutron merupakan hal yang sangat penting dalam analisis reaktor, perhitungan difusi ini dilakukan secara berulang-ulang, hal ini menyebabkan waktu untuk menganalisis reaktor nuklir menjadi sangat lama. Karena itu diperlukan suatu program yang dapat mempercepat perhitungan persamaan difusi neutron tersebut.

1.3 Batasan Masalah

1. Perhitungan persamaan difusi neutron pada penelitian ini dibatasi pada geometri teras berbentuk silinder dengan distribusi bahan bakar homogen konsentris. Dengan demikian persoalan geometri dapat direduksi menjadi 2-dimensi. Sedangkan tipe reaktor yang digunakan dalam perhitungan ini adalah reaktor cepat dengan bahan bakar UN-PuN (Uranium Nitrida dan Plutonium Nitrida).

2. Program dijalankan pada komputer 2 *core* dan 4 *core*.
3. Program di *benchmark* dengan program FI-ITBCHI.

1.4 Tujuan Penelitian

1. Membangun kode program perhitungan persamaan difusi neutron secara paralel dengan metode Jacobi.
2. Melakukan analisis kinerja program paralel yang dihasilkan.

1.5 Manfaat Penelitian

Dengan dihasilkannya kode program perhitungan persamaan difusi neutron secara paralel, diharapkan perhitungan persamaan difusi neutron dapat dipercepat. Dengan demikian perhitungan analisis reaktor nuklir secara keseluruhan dapat dipercepat.

BAB II

TINJAUAN PUSTAKA

2.1 Studi Literatur

Penelitian tentang perhitungan difusi neutron juga telah dilakukan oleh Ariani dengan judul "Penentuan Parameter-Parameter Optimal Dalam Menyelesaikan Persamaan Difusi Neutron Menggunakan Metode Beda Hingga". Yang merupakan sebuah program komputer untuk menyelesaikan persamaan difusi neutron dalam ruang tiga dimensi, energi banyak kelompok dan berbasis metoda beda hingga yang dikembangkan dengan menggunakan bahasa pemrograman fortran. Untuk menyelesaikan persamaan difusi neutron dengan metode beda hingga, diterapkan metode iterasi beranting dengan *Red-Black Line SOR* untuk perhitungan iterasi dalam.

Su'ud, dkk. (2001) berhasil menjalankan program komputasi paralel untuk perhitungan konstanta grup nuklir. Pada penelitian tersebut program paralel dijalankan pada *cluster* multi-PC heterogen (tak-simetris) yang dihubungkan dengan *fast ethernet card* 100-*base* dan menggunakan PVM (*Parallel Virtual Machine*) pada *platform* Linux. Jenis prosesor yang digunakan adalah AMD K-6 dan Athlon dengan kecepatan *clock* CPU berkisar dari 300 MHz hingga 500 MHz. Penggunaan *fast ethernet* dengan kecepatan maksimum 100 Mbps masih mampu meningkatkan kinerja paralelisasi mengingat bahwa saat itu yang digunakan adalah prosesor dengan frekuensi *clock* CPU pada orde ratusan MHz.

Tahun 2010, Taufiq telah berhasil dilakukan paralelisasi perhitungan burnup bahan bakar pada reaktor cepat dengan bahan bakar UN-PuN dengan pendingin Pb-Bi. Hasilnya menunjukkan bahwa dengan menggunakan Intel TBB dan C++ telah berhasil dicapai *speedup* pada PC dengan prosesor *quad-core* sebesar 3,58 dan skalabilitas program dipenuhi.

2.2 Landasan Teori

2.2.1 Komputasi Paralel

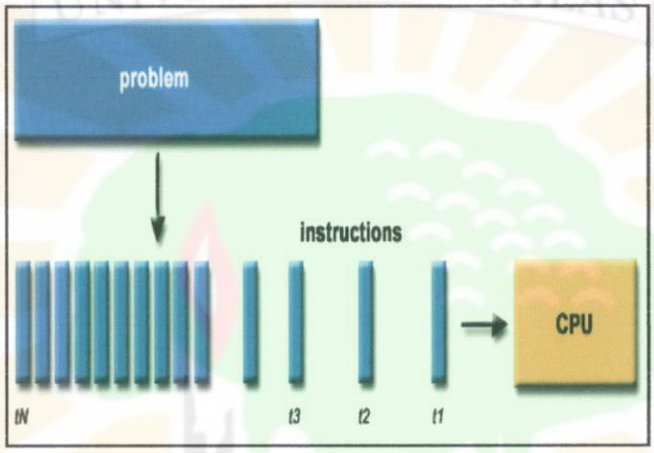
Pemrograman paralel merupakan teknik yang digunakan dalam pemrograman komputer yang memungkinkan eksekusi perintah atau operasi dijalankan secara bersamaan (komputasi paralel), baik dalam komputer dengan satu prosesor (tunggal) ataupun banyak prosesor (ganda). Tujuan utama dari pemrograman paralel adalah untuk meningkatkan kinerja komputasi. Semakin banyak hal yang bisa dilakukan secara serentak, semakin banyak pekerjaan yang bisa diselesaikan dalam waktu yang sama.

Komputasi paralel adalah proses atau pekerjaan komputasi di komputer dengan memakai suatu bahasa pemrograman yang dijalankan secara paralel pada saat bersamaan. Secara umum komputasi paralel diperlukan untuk meningkatkan kecepatan eksekusi bila dibandingkan dengan pemakaian komputasi pada komputer tunggal. Penggunaan komputasi paralel merupakan pilihan yang cukup handal pada saat ini untuk tugas komputasi yang berat.

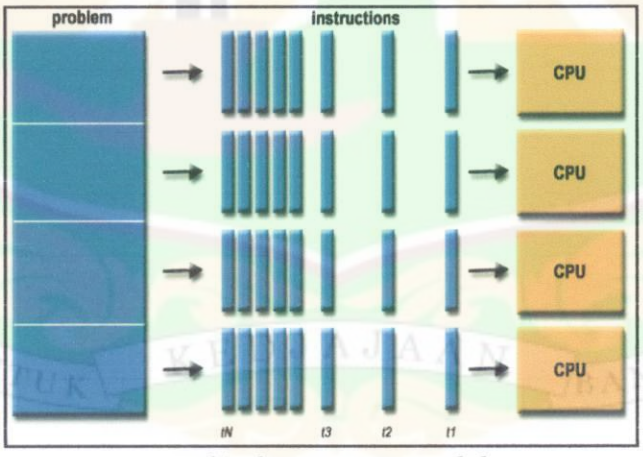
Komputasi paralel melakukan proses komputasi dengan menggunakan 2 atau lebih CPU/*Processor* dalam suatu komputer yang sama atau komputer yang

berbeda (*cluster of PCs*). Setiap masalah dibagi kedalam beberapa instruksi kemudian dikirim ke prosesor yang tersedia dan eksekusi dilakukan secara serentak.

Berikut ini adalah gambar perbedaan antara komputasi tunggal dengan paralel komputasi.



a) komputasi tunggal/serial



b) komputasi paralel

Gambar 2.1 Perbandingan antara komputasi serial dan komputasi paralel (Grama dkk., 2003)

Ada tiga cara untuk memecah masalah menjadi tugas-tugas komputasi yang lebih kecil:

a. Dekomposisi fungsional : masalah dibagi menjadi satuan tugas yang berbeda-beda yang selanjutnya akan didistribusikan ke masing-masing prosesor untuk dieksekusi secara serentak.
b. Dekomposisi domain (paralelisme data) : membagi domain data pada masalah kemudian mendistribusikannya pada masing-masing prosesor untuk dieksekusi secara serentak.
c. Kombinasi antara kedua cara dekomposisi tersebut.

Pemprograman serial atau pemprograman yang biasa dilakukan membagi tugas komputasi menjadi bagian yang kecil dan mengumpankannya kepada prosesor secara berurutan. Sedangkan pada komputasi paralel bagian yang kecil itu diumpankan secara serentak kepada beberapa prosesor yang ada. Kinerja pada komputasi paralel diukur dari peningkatan kecepatan (*speed up*) yang digunakan pada teknik paralel.

Peningkatan kecepatan dapat diformulasikan dalam persamaan berikut ini

$$Speedup = \frac{Tserial}{Tparalel} \qquad 2.1$$

dengan *Tserial* adalah waktu eksekusi program versi serial dan *Tparalel* adalah waktu eksekusi program versi paralel.

Paralelisasi perlu dilakukan untuk mendapatkan waktu yang lebih singkat dalam menjalankan program. Oleh karenanya analisis *speedup* merupakan bagian penting dari kinerja paralelisasi. Dalam pewaktuan program dikenal adanya istilah *CPU-time* dan *wall-clock time*. *CPU-time* merupakan waktu yang diperlukan oleh

CPU untuk memproses instruksi dalam suatu program (tanpa memperhitungkan waktu untuk proses lainnya, misalnya: menunggu proses *input/output*). Sedangkan *wall-clock-time* merupakan waktu nyata (*real time*) yang dihitung sejak program dijalankan hingga program selesai menghasilkan keluaran yang diinginkan.

Ada limitasi dalam usaha membuat suatu program komputer berjalan lebih efisien melalui peningkatan kecepatan, hukum yang menetapkan batasan ini dikenal sebagai Hukum Amdahl. Ide dari hukum amdahl ini adalah bahwa anda hanya akan bisa meningkatkan efisiensi program komputer anda, sebatas pada bagian tertentu dari program tersebut yang dapat di paralelkan. Sementara bagian yang memang harus dilaksanakan secara berurutan, akan menjadi penentu performa akhir.

Batas maksimum peningkatan kecepatan yang mampu dicapai menurut hukum Amdahl yaitu perbandingan terbalik dari seberapa banyak bagian serial dari suatu pekerjaan.

Menurut Hukum Amdahl, peningkatan kecepatan pada algoritma paralel tidak terbatas oleh jumlah prosesor yang bekerja tetapi oleh fraksi algoritma yang tidak dapat diparalelkan (Hesyam, 2005). Sekilas mengenai hukum Amdahl, ditandai tanpa melihat dari jumlah dari prosesor yang dipakai, terdapat batasan yang pasti pada penggunaan dari arsitektur paralel.

Telah dijelaskan bahwa dari *Tserial* (waktu yg dibutuhkan menjalankan pekerjaan dalam satu komputer/secara serial), ada sebagian yg tidak bisa diparalelkan. Untuk menyatakan ini kita gunakan notasi α dimana $0 \leq \alpha \leq 1$

menunjukkan berapa bagian dari *Tserial* yang tidak bisa dijadikan paralel (atau bagian serial dari program ini).

Maka dapat diketahui

$$\alpha \; x \; Tserial \tag{2.2}$$

adalah waktu yg tidak akan terpengaruh oleh bertambahnya komputer yg digunakan.

Sedangkan sisanya $(1 - \alpha) \; x \; Tserial$ adalah waktu yang akan berkurang menjadi

$$\frac{(1-\alpha) \; x \; Tserial}{N} \tag{2.3}$$

bila menggunakan N buah prosesor. Sehingga waktu total yang dibutuhkan untuk menjalankan pekerjaan dalam N prosesor adalah (persamaan 2.2 dan 2.3)

$$Tparalel = \alpha \text{ x } Tserial + \frac{(1-\alpha) \text{ x } Tserial}{N} \tag{2.4}$$

Peningkatan kecepatan yang kita peroleh dari persamaan ini adalah :

$$Speedup = \frac{Tserial}{\alpha \text{ x } Tserial + \frac{(1-\alpha) \text{ x } Tserial}{N}} \tag{2.5}$$

Dengan mengeliminasi komponen *Tserial* (pada bagian atas dan bawah persamaan), lalu mengatur N dan α, sehingga didapatkan

$$Speedup = \frac{N}{1 + \alpha \; (N-1)} \tag{2.6}$$

Bila anda cermati persamaan di atas, bisa dilihat bahwa jika kita menggunakan prosesor yang amat banyak $(N \rightarrow \infty)$, persamaan 2.3 dapat diabaikan, menyisakan persamaan :

$$Speedup = \frac{1}{\alpha} \tag{2.7}$$

Inilah batas maksimum peningkatan kecepatan yang bisa dicapai menurut hukum Amdahl yaitu perbandingan terbalik dari seberapa banyak bagian serial dari suatu pekerjaan.

Jika dengan bertambahnya jumlah prosesor/*core*, nilai *speedup* meningkat, maka program dikatakan bersifat *scalable*. Sebaliknya, jika dengan bertambahnya jumlah prosesor/*core* tidak meningkatkan nilai *speedup* maka program dikatakan tidak *scalable*. Pada kondisi ini jika dipaksakan ditambah jumlah prosesor, maka nilai *speedup* akan menurun karena waktu untuk *schedulling* dan sinkronisasi menjadi dominan.

2.2.2 *Intel Threading Building Blocks*

Intel *threading building bloks* (Intel TBB) adalah program bantu yang menawarkan pendekatan yang lengkap untuk memudahkan paralelisasi dalam bahasa C++. Merupakan suatu perpustakaan yang dapat meningkatkan kinerja dari multi prosesor. *Library* pada intel TBB dirancang untuk memaksimalkan kerja dari *multicore* prosesor dengan pemprograman berbasis *task*. *Template* yang disediakan memudahkan pengguna untuk melakukan paralelisasi (Reinders, 2007).

Intel TBB membantu untuk menciptakan aplikasi yang dapat memberi keuntungan untuk mengolah dengan prosesor yang semakin banyak. Intel TBB ini menggunakan *template* iterasi paralel, memungkinkan seorang programer untuk meningkatkan kecepatan eksekusi program dengan beberapa prosesor pengolah tanpa perlu direpotkan dengan sinkronisasi dan pengimbangan (Reinders, 2007).

Program yang mengunakan intel TBB dapat dieksekusi pada sistem prosesor ganda, begitu juga pada sistem multiprosesor sehingga dapat menghasilkan program paralel yang bersifat *scalable*. Selain itu intel TBB juga dapat mendukung paralelisasi bertingkat, sehingga dapat membangun komponen paralel besar dari komponen paralel yang kecil dengan relatif mudah (Reinders, 2007).

Keuntungan dari *Threading Building Block* adalah

a. Pemrograman dilakukan pada *task* dan bukan pada *thread.*
b. Dapat digabungkan dengan program lain.
c. Pemprograman data paralel dengan *scalable*.

2.2.3 Persamaan Difusi Neutron

Fluks neutron adalah kuantitas yang digunakan dalam fisika reaktor sesuai dengan panjang total yang ditempuh oleh semua neutron per satuan waktu dan volume. Biasanya fluks neutron terkuat terjadi di tengah inti reaktor, menjadi lebih rendah di bagian pinggir.

Dalam reaktor neutron bergerak ke segala arah, dan probabilitas terjadinya tumbukan antara neutron dan inti sama ke segala arah, atau dengan kata lain, secara umum probabilitas tumbukan tidak tergantung arah, tetapi bergantung pada kerapatan *n* (n/cm3) dan kecepatan neutron *v* (cm/s). Oleh karena itu didefinisikan besaran yang disebut fluks neutron, sebagai hasil kali antara kerapatan dan kecepatan neutron:

$$\emptyset \left(\frac{n}{cm^2.s}\right) = n\left(\frac{n}{cm^3}\right) \text{x } v \left(\frac{cm}{s}\right) \qquad 2.8$$

Faktor multiplikasi merupakan perbandingan antara populasi neutron dari generasi berurutan.

$$k_{eff} = \frac{N_1}{N_2} \quad 2.9$$

Dimana N_1 adalah banyaknya neutron pada generasi pertama, dan N_2 adalah banyaknya neutron pada generasi berikutnya. Berdasarkan faktor multiplikasi, didefinisikan tiga kondisi kritikalitas reaktor:

- Kondisi subkritis : faktor multiplikasi < 1

  Kondisi subkritis biasanya terjadi pada saat penurunan daya (jumlah reaksi fissi) reaktor.
- Kondisi kritis : faktor multiplikasi = 1

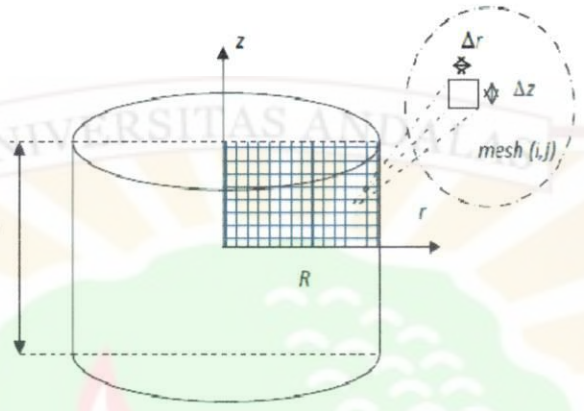
  Kondisi kritis biasanya digunakan untuk membawa reaktor pada kondisi operasi dengan daya (jumlah reaksi fissi) konstan.
- Kondisi superkritis : faktor multiplikasi > 1

  Kondisi superkritis biasanya terjadi pada saat penaikan daya (jumlah reaksi fissi) reaktor.

Ketiga kondisi diatas dijelaskan dalam konteks operasi normal, dalam kondisi anomali atau kecelakaan ketiga status kritikalitas dapat muncul dengan urutan yang tak rerduga.

Perhitungan distribusi fluks neutron dilakukan dengan menggunakan metode penghampiran terhadap persamaan transport neutron, yakni persamaan difusi. Geometri teras reaktor yang ditinjau berbentuk silinder sebagaimana tampak pada gambar 2.2. Dengan asumsi distribusi bahan bakar bersifat simetris radial, maka perhitungan dapat dilakukan dengan menggunakan koordinat silinder

2-dimensi (r,z). Bagian teras reaktor yang ditinjau dibagi menjadi bagian-bagian kecil yang disebut *mesh* spasial yang posisinya ditandai dengan indeks (i,j). Dengan nilai $i = 1, 2, ..., i\text{-}1, i$ dan $j = 1, 2, ..., j\text{-}1, j$.



Gambar 2.2 Geometri teras reaktor berbentuk silinder.

Persamaan difusi neutron dapat dituliskan sebagai berikut (Stacey, 2004) :

$$-\vec{\nabla}.D(\vec{r})\vec{\nabla}\phi(\vec{r}) + \Sigma_a(\vec{r})\phi(\vec{r}) = \frac{1}{k_{eff}}\upsilon\Sigma_f(\vec{r})\phi(\vec{r}) \qquad 2.10$$

dengan ϕ adalah fluks neutron, Σ_a dan Σ_f masing-masing adalah tampang lintang absorpsi makroskopik dan tampang lintang fisi makroskopik, k_{eff} adalah faktor multiplikasi efektif neutron. Integrasi persamaan di atas terhadap *mesh* (i,j) menghasilkan

$$\int_{i,j} -\vec{\nabla}.D(\vec{r})\vec{\nabla}\phi(\vec{r})dV + \int_{i,j}\Sigma_a(\vec{r})\phi(\vec{r})dV = \int_{i,j}\frac{1}{k_{eff}}\upsilon\Sigma_f(\vec{r})\phi(\vec{r})dV \qquad 2.11$$

Suku kedua di ruas kiri dan suku tunggal di ruas kanan dari persamaan (2.10) dapat dengan mudah dintegrasikan, dan suku pertama ruas kiri dapat diubah menjadi integral luasan, sehingga persamaan di atas dapat diubah menjadi

$$-\oint_{i,j}D(\vec{r})\vec{\nabla}\phi(\vec{r}).dA + \Sigma_{a,i,j}\phi_{i,j}V_{i,j} = \frac{1}{k_{eff}}\upsilon\Sigma_{f,i,j}\phi_{i,j}V_{i,j} \qquad 2.12$$

Selanjutnya suku pertama ruas kiri dari persamaan (2.12), dijabarkan menjadi

$$-\oint_{i,j} D(\vec{r})\vec{\nabla}\phi(\vec{r}) \cdot d\vec{A} = -\int_{i+\frac{1}{2}} D(\vec{r})\vec{\nabla}\phi(\vec{r}) \cdot d\vec{A} + \int_{i-\frac{1}{2}} D(\vec{r})\vec{\nabla}\phi(\vec{r}) \cdot d\vec{A} -$$

$$\int_{j+\frac{1}{2}} D(\vec{r})\vec{\nabla}\phi(\vec{r}) \cdot d\vec{A} + \int_{j-\frac{1}{2}} D(\vec{r})\vec{\nabla}\phi(\vec{r}) \cdot d\vec{A} \quad 2.13$$

Suku pertama ruas kanan persamaan (2.13) dapat diselesaikan menjadi

$$\int_{i+\frac{1}{2}} D(\vec{r})\vec{\nabla}\phi(\vec{r}) \cdot d\vec{A} = -D^{i+\frac{1}{2},j} \left.\frac{d\phi}{dz}\right|_{i+\frac{1}{2}} A_{i+\frac{1}{2},j} = -D^{i+\frac{1}{2},j} \frac{(\phi_{i+1,j}-\phi_{i,j})}{r_{i+1}-r_i} A_{i+\frac{1}{2},j} \quad 2.14$$

Suku kedua ruas kanan persamaan (2.13) menjadi

$$\int_{i-\frac{1}{2}} D(\vec{r})\vec{\nabla}\phi(\vec{r}) \cdot d\vec{A} = D^{i-\frac{1}{2},j} \left.\frac{d\phi}{dz}\right|_{i-\frac{1}{2}} A_{i-\frac{1}{2},j} = D^{i-\frac{1}{2},j} \frac{(\phi_{i,j}-\phi_{i-1,j})}{r_i-r_{i-1}} A_{i-\frac{1}{2},j} \quad 2.15$$

Suku ketiga ruas kanan persamaan (2.13) menjadi

$$-\int_{j+\frac{1}{2}} D(\vec{r})\vec{\nabla}\phi(\vec{r}) \cdot d\vec{A} = -D^{i,j+\frac{1}{2}} \left.\frac{d\phi}{dz}\right|_{j+\frac{1}{2}} A_{i,j+\frac{1}{2}} = -D^{i,j+\frac{1}{2}} \frac{(\phi_{i,j+1}-\phi_{i,j})}{z_{j+1}-z_j} A_{i,j+\frac{1}{2}} \quad 2.16$$

Dan suku keempat ruas kanan persamaan (2.13) menjadi

$$\int_{j+\frac{1}{2}} D(\vec{r})\vec{\nabla}\phi(\vec{r}) \cdot d\vec{A} = D^{i,j-\frac{1}{2}} \left.\frac{d\phi}{dz}\right|_{j-\frac{1}{2}} A_{i,j-\frac{1}{2}} = D^{i,j-\frac{1}{2}} \frac{(\phi_{i,j}-\phi_{i,j-1})}{z_j-z_{j-1}} A_{i,j-\frac{1}{2}} \quad 2.17$$

Dengan demikian, dapat dituliskan persamaan difusi yang telah didiskretisasi sebagai berikut

$$-D^{i+\frac{1}{2},j} \frac{(\phi_{i+1,j}-\phi_{i,j})}{r_{i+1}-r_i} A_{i+\frac{1}{2},j} + D^{i-\frac{1}{2},j} \frac{(\phi_{i,j}-\phi_{i-1,j})}{r_i-r_{i-1}} A_{i+\frac{1}{2},j}$$

$$-D^{i,j+\frac{1}{2}} \frac{(\phi_{i,j+1}-\phi_{i,j})}{z_{j+1}-z_j} A_{i,j+\frac{1}{2}} + D^{i,j-\frac{1}{2}} \frac{(\phi_{i,j}-\phi_{i,j-1})}{z_j-z_{j-1}} A_{i,j-\frac{1}{2}} + \Sigma_{a,i,j}\phi_{i,j}V_{i,j}$$

$$= \frac{1}{k_{eff}} \upsilon\Sigma_{f,i,j}\phi_{i,j}V_{i,j} \quad 2.18$$

Jika dikelompokkan kembali berdasarkan fluks neutronnya, dapat dituliskan menjadi

$$\phi_{i,j-1}\left(-\frac{D^{i,j-\frac{1}{2}}}{z_j - z_{j-1}}A_{i,j-\frac{1}{2}}\right) + \phi_{i-1,j}\left(-\frac{D^{i-\frac{1}{2},j}}{r_i - r_{i-1}}A_{i+\frac{1}{2},j}\right)$$

$$+ \phi_{i,j}\left(\frac{D^{i,j+\frac{1}{2}}}{z_{j+1} - z_j}A_{i,j+\frac{1}{2}} + \frac{D^{i+\frac{1}{2},j}}{r_{i+1} - r_i}A_{i+\frac{1}{2},j} + \frac{D^{i,j-\frac{1}{2}}}{z_j - z_{j-1}}A_{i,j-\frac{1}{2}} + \frac{D^{i-\frac{1}{2},j}}{r_{i-1} - r_i}A_{i-\frac{1}{2},j}\right.$$

$$\left. + \Sigma_{a,i,j}\phi_{i,j}V_{i,j}\right)$$

$$+\phi_{i+1,j}\left(\frac{-D^{i+\frac{1}{2},j}}{r_{i+1} - r_i}A_{i+\frac{1}{2},j}\right) + \phi_{i,j+1}\left(\frac{-D^{i,j+\frac{1}{2}}}{z_j - z_{j+1}}A_{i,j+\frac{1}{2}}\right)$$

$$= \frac{1}{k_{eff}}\upsilon\Sigma_{f,i,j}\phi_{i,j}V_{i,j} \qquad 2.19$$

Persamaan (2.19) dapat diringkas penulisannya menjadi

$$-\gamma_{i,j}\phi_{i,j-1} - \alpha_{i,j}\phi_{i-1,j} + \beta_{i,j}\phi_{i,j} - \alpha_{i,j}\phi_{i+1,j} - \gamma_{i,j+1}\phi_{i,j+1} = S_{i,j} \qquad 2.20$$

dengan mendefinisikan variabel-variabel baru sebagai berikut

$$\gamma_{i,j} = -\frac{D^{i,j-\frac{1}{2}}}{z_j - z_{j-1}}A_{i,j-\frac{1}{2}} \qquad 2.21$$

$$\alpha_{i,j} = -\frac{D^{i-\frac{1}{2},j}}{r_i - r_{i-1}}A_{i+\frac{1}{2},j} = \frac{-D^{i+\frac{1}{2},j}}{r_{i+1} - r_i}A_{i+\frac{1}{2},j} \qquad 2.22$$

$$\beta_{i,j} = \frac{D^{i,j+\frac{1}{2}}}{z_{j+1} - z_j}A_{i,j+\frac{1}{2}} + \frac{D^{i+\frac{1}{2},j}}{r_{i+1} - r_i}A_{i+\frac{1}{2},j} + \frac{D^{i,j-\frac{1}{2}}}{z_j - z_{j-1}}A_{i,j-\frac{1}{2}} + \frac{D^{i-\frac{1}{2},j}}{r_{i-1} - r_i}A_{i-\frac{1}{2},j} + \Sigma_{a,i,j}\phi_{i,j}V_{i,j} \qquad 2.23$$

$$\gamma_{i,j+1} = \frac{-D^{i,j+\frac{1}{2}}}{z_j - z_{j+1}}A_{i,j+\frac{1}{2}} \qquad 2.24$$

$$S_{i,j} = \frac{1}{k_{eff}}\upsilon\Sigma_{f,i,j}\phi_{i,j}V_{i,j} \qquad 2.25$$

Untuk dapat menuliskan keseluruhan persamaan berlaku pada seluruh *mesh* spasial yang ada, diterapkan syarat batas sebagai berikut

$$\phi(R + 0{,}71\lambda_{tr}) = 0$$

$$\left.\frac{d\phi}{dr}\right|_{r=0} = 0$$

$$\phi\left(r, \frac{h}{2} + 0{,}71\lambda_{tr}\right) = 0$$

$$\phi\left(r, -\frac{h}{2} + 0{,}71\lambda_{tr}\right) = 0 \qquad 2.26$$

Dengan R adalah jari-jari silinder, h adalah tinggi reaktor, dan λ_{tr} adalah jarak bebas rata-rata transport (*transport mean free path*). Nilai $0{,}71\lambda_{tr}$ ditambahkan pada syarat batas sebagai koreksi agar nilai yang dihasilkan dari persamaan difusi menjadi lebih akurat pada batas-batas medium (Deuderstadt, 1976).

Untuk nilai $i = 1, j =$ bebas,

$$\gamma_{i,j}\phi_{i,j-1} - \alpha_{i,j}\phi_{0,j} + \beta_{1,j}\phi_{1,j} - \alpha_{i,j}\phi_{0,j} - \gamma_{1,j+1}\phi_{1,j+1} = S_{1,j} \qquad 2.27$$

Karena $\phi_{0,j} = \phi_{1,j}$, maka persamaan (2.27) menjadi

$$\gamma_{i,j}\phi_{i,j-1} + (\beta_{1,j} - \alpha_{1,j})\phi_{1,j} - \alpha_{i,j}\phi_{0,j} - \gamma_{1,j+1}\phi_{1,j+1} = S_{1,j} \qquad 2.28$$

Untuk nilai i=I dan j=bebas

$$\gamma_{I,j}\phi_{I,j-1} - \alpha_{I,j}\phi_{I-1,j} + \beta_{I,j}\phi_{I,j} - \alpha_{I,j}\phi_{I+1,j} - \gamma_{I,j+1}\phi_{I,j+1} = S_{I,j} \qquad 2.29$$

Karena $\phi_{I+1,j} = 0$, maka persamaan (2.29) menjadi

$$-\gamma_{I,j}\phi_{I,j-1} - \alpha_{I,j}\phi_{I-1,j} + \beta_{I,j}\phi_{I,j} - \gamma_{I,j+1}\phi_{I,j+1} = S_{I,j} \qquad 2.30$$

Untuk nilai i = bebas dan j = 1

$$-\gamma_{i,1}\phi_{i,0} - \alpha_{i,1}\phi_{i-1,1} + \beta_{i,1}\phi_{i,1} - \alpha_{i,1}\phi_{i+1,1} - \gamma_{i,2}\phi_{i,2} = S_{i,1} \qquad 2.31$$

Karena $\phi_{i,0} = 0$, maka persamaan (2.31) menjadi

$$\alpha_{i,1}\phi_{i-1,1} + \beta_{i,1}\phi_{i,1} - \alpha_{i,1}\phi_{i+1,1} - \gamma_{1,2}\phi_{i,2} = S_{i,1} \qquad 2.32$$

Untuk nilai i = bebas, j = J

$$\alpha_{i,J}\phi_{i-1,J} + \beta_{i,J}\phi_{i,J} - \alpha_{i,J}\phi_{i+1,J} - \gamma_{1J+1}\phi_{i,J+1} = S_{i,J} \quad 2.33$$

Karena $\phi_{i+1,J} = 0$, maka persamaan (2.33) menjadi

$$-\gamma_{i,J}\phi_{i,J-1} - \alpha_{i,J}\phi_{i-1,J} + \beta_{i,J}\phi_{i,J} - \alpha_{i,J}\phi_{i+1,J} = S_{i,J} \quad 2.34$$

Kemudian persamaan ini dapat disusun ke dalam sebuah matriks *sparse* pentadiagonal **A** yang berisi nilai fluks dan matriks **S** yang berisi suku sumber neutron, sehingga persamaan difusi neutron dapat dituliskan sebagai persamaan matriks sebagai berikut

$$A\Phi = S \quad 2.35$$

2.2.4 Metoda Iterasi

Metode-metode solusi numerik yang banyak dipakai, dapat diklasifikasikan sebagai:

1. Metode Langsung

- Metode Eliminasi *Gauss* (EG), merupakan operasi eliminasi dan substitusi variabel-variabelnya sedemikian rupa sehingga dapat terbentuk matriks segitiga atas, dan akhirnya solusinya diselesaikan menggunakan teknik substitusi balik (*back substitution*).
- Metode Eliminasi *Gauss-Jordan* (EGJ), mirip sekali dengan metode EG, namun dalam metode ini jumlah operasi numerik yang dilakukan jauh lebih besar, karena matriks *A* mengalami inversi terlebih dahulu untuk mendapatkan matriks identitas (*I*). Karena kendala tersebut, maka metode

ini sangat jarang dipakai, namun sangat bermanfaat untuk menginversikan matriks.

- Dekomposisi LU (DECOLU), melakukan dekomposisi matriks *A* terlebih dahulu sehingga dapat terbentuk matriks-matrik segitiga atas dan bawah, kemudian secara mudah dapat melakukan substitusi balik (*back substitution*) untuk berbagai vektor VRK (vektor ruas kanan).
- Solusi sistem tridiagonal (S3DIAG), merupakan solusi dengan bentuk matrik pita (satu diagonal bawah, satu diagonal utama, dan satu diagonal atas) pada matriks *A*.

2. Metode Tak-Langsung (Metode Iteratif)

a. Metode Jacobi, merupakan metode iteratif yang melakukan perbaharuan nilai x yang diperoleh tiap iterasi (mirip metode substitusi berurutan, *successive substitution*).

Jika matriks A merupakan matriks singular maka untuk dapat menyelesaikan persamaan ini menggunakan (Varga, 2009)

$$\Phi = A^{-1} S \qquad 2.36$$

dengan A^{-1} adalah invers dari matriks A. Sedangkan matriks A dapat diurai (dekomposisi) menjadi jumlahan dari matriks diagonalnya D, matriks segitiga-atas (*upper-triangular*) U, dan matriks segitiga-bawah (*lower-triangular*) L, seperti dituliskan di bawah ini (Vesely, 1994)

$$A = D + U + L \qquad 2.37$$

Dengan demikian persamaan (2.37) dapat dituliskan sebagai (Varga, 2009)

$$D\Phi = (U + L)\,\Phi + S \qquad 2.38$$

Persamaan iteratif dari persamaan (2.36) dapat ditulis (Varga, 2009)

$$a_{i,i}\phi_i^{m+1} = -\Sigma_{j=1}^{n} a_{i,j}\phi_j^m + S_i, 1 \leq i \leq n, m \geq 0, \; j \neq i \qquad 2.39$$

Dengan ϕ_i^0 adalah nilai tebakan awal. Kemudian persamaan (2.36) dapat ditulis ke dalam bentuk (Varga, 2009)

$$\phi_i^{m+1} = -\Sigma_{j=1}^{n} \frac{a_{i,j}}{a_{i,i}}\phi_j^m + \frac{S_i}{a_{i,i}}, 1 \leq i \leq n, m \geq 0, \qquad 2.40$$

Perhitungan iteratif seperti persamaan (2.40) disebut sebagai metode Jacobi, atau dikenal dengan *total-step iterative method* (Varga, 2009).

b. Metode Gauss-Seidel, mirip metode *Jacobi*, namun melibatkan perhitungan implisit.

Telaah terhadap metode Jacobi menunjukkan bahwa seluruh komponen vektor ϕ^m harus tetap disimpan dalam memori sementara dilakukan perhitungan terhadap nilai komponen ϕ^{m+1}. Metode Gauss-Seidel memanfaatkan sebagian nilai komponen ϕ^{m+1} yang telah dihitung untuk menghitung nilai komponen ϕ^{m+1}. lainnya. Dengan demikian akan lebih menghemat memori yang digunakan karena nilai komponen yang lama bisa segera ditimpa (*overwrite*) dengan nilai perhitungan yang baru. Persamaan iteratif Gauss-Seidel berbentuk sebagai berikut

$$\phi_i^{m+1} = -\sum_{j=1}^{i-1}\left(\frac{a_{i,j}}{a_{i,i}}\right)\phi_j^{m+1} - \sum_{j=i+1}^{n}\left(\frac{a_{i,j}}{a_{i,i}}\right)\phi_j^m + \frac{S_i}{a_{i,i}}, 1 \leq i \leq n, m \geq 0 \quad 2.41$$

Metode Gauss-Seidel ini dikenal pula dengan nama *single-step iterative method.*

c. Metode *Successive Over Relaxation* (SOR), merupakan perbaikan secara langsung dari Metode *Gauss-Seidel* dengan cara menggunakan faktor relaksasi (faktor pembobot) pada setiap tahap/proses iterasi.

Untuk mendapatkan bentuk persamaan iteratif SOR, pertama kali didefinisikan variabel baru $\tilde{\phi}_i^{m+1}$ yang menggantikan posisi ϕ_j^{m+1} dalam persamaan (2.39) sebagai berikut (Varga, 2009)

$$\tilde{\phi}_i^{m+1} = -\sum_{j=1}^{i-1}\left(\frac{a_{i,j}}{a_{i,i}}\right)\phi_j^{m+1} - \sum_{j=i+1}^{n}\left(\frac{a_{i,j}}{a_{i,i}}\right)\phi_j^{m} + \frac{s_i}{a_{i,i}} \qquad 2.42$$

nilai ϕ_j^{m+1} yang sesungguhnya dihitung melalui persamaan iteratif berikut ini

$$\phi_j^{m+1} = \phi_j^{m} + \omega\left(\tilde{\phi}_i^{m+1} - \phi_j^{m}\right) \qquad 2.43$$

atau dapat juga dituliskan dalam bentuk

$$\phi_j^{m+1} = (1-\omega)\phi_j^{m} + \omega\tilde{\phi}_i^{m+1} \qquad 2.44$$

Dengan ω disebut sebagai faktor relaksasi. Untuk kasus $\omega < 1$ disebut sebagai *under-relaxation* dan untuk $\omega > 1$ disebut *over-relaxation*. Metode SOR menggunakan nilai $1 < \omega < 2$.

BAB III

METODOLOGI PENELITIAN

3.1 Waktu dan Lokasi Penelitian

Penelitian ini dilakukan di laboratorium Komputer Jurusan Fisika Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Andalas dari bulan Juni 2011.

3.2 Prosedur Penelitian

Pada penelitian ini akan dilakukan perhitungan fluks neutron. Dari penyelesaian persamaan difusi neutron dapat diperoleh nilai fluks neutron disetiap *mesh* spasial yang ditinjau.

3.2.1 Perumusan Masalah Fisis

Dari ketiga metoda, yaitu metoda SOR, Gauss Seidel dan Jacobi, pada penelitian ini menggunakan metoda Jacobi karena metoda Jacobi ini lebih mudah diparalelisasikan. Hal ini dapat dipahami dengan Persamaan 3.1 sebagai berikut (Deuderstat, 1976).

$$\begin{aligned} \boxed{a_{11}\phi_1^{m+1}} + a_{12}\phi_2^m + a_{13}\phi_3^m + \ldots + a_{1N}\phi_N^m &= S_1 \\ a_{21}\phi_1^m + \boxed{a_{22}\phi_2^{m+1}} + a_{23}\phi_3^m + \ldots + a_{2N}\phi_N^m &= S_2 \\ a_{N1}\phi_1^m + a_{N2}\phi_2^m + a_{N3}\phi_3^m + \ldots + \boxed{a_{NN}\phi_N^{m+1}} &= S_N \end{aligned} \qquad 3.1$$

Sehingga kita dapat menyelesaikan nilai fluk $\boldsymbol{m+1}$ dengan Persamaan 3.2 (Deuderstat, 1976).

$$\phi_i^{m+1} = -\Sigma_{j=1}^{n} \frac{a_{i,j}}{a_{i,i}} \phi_j^m + \frac{S_i}{a_{i,i}}, i = 1,2 \ldots, n \quad 3.2$$

Dari persamaan ini, iterasi Jacobi tidak menggunakan semua informasi selama tiap iterasi. Dalam menyelesaikan persamaan dalam urutan dari i=1 ke i=n, untuk mendapatkan nilai ϕ_i^{m+1} yang baru hanya memerlukan perhitungan nilai ϕ_j^m.

Sedangkan pada metoda Gauss Seidel dan SOR, untuk memperoleh ϕ_i^{m+1}, memerlukan nilai ϕ pada perhitungan sebelumnya (ϕ_j^{m+1}) dan nilai ϕ_j^m (dalam arti untuk perhitungan nilai ϕ baru masih diperlukan nilai ϕ pada perhitungan sebelumnya). Dapat dilihat seperti Persamaan 3.3 (Deuderstat, 1976).

$$\begin{aligned}
&a_{11}\phi_1^{m+1} + a_{12}\phi_2^m + a_{13}\phi_3^m + \ldots + a_{1N}\phi_N^m = S_1 \\
&a_{21}\phi_1^{m+1} + a_{22}\phi_2^{m+1} + a_{23}\phi_3^m + \ldots + a_{1N}\phi_N^m = S_2 \\
&a_{31}\phi_1^{m+1} + a_{32}\phi_2^{m+1} + a_{33}\phi_3^{m+1} + \ldots + a_{3N}\phi_N^m = S_3 \\
&a_{N1}\phi_1^{m+1} + a_{N2}\phi_2^{m+1} + a_{N3}\phi_3^{m+1} + \ldots + a_{NN}\phi_N^{m+1} = S_N
\end{aligned} \quad 3.3$$

Dalam menjalankan sitem paralelisasi program difusi ini, dibutuhkan nilai yang tidak membutuhkan nilai dari perhitungan sebelumnya. Maka dari itu, penelitian ini menggunakan metoda Jacobi dalam memparalelkan persamaan difusi neutron.

3.2.2 Pengetikan Kode Program

Pada penelitian ini digunakan bahasa pemrograman C++ pada *Microsoft Visual Studio* 2005 dengan menggunakan *template intel Threading Building Block.*

3.2.3 Peralatan Penelitian

Dalam menjalankan program, peneliti menggunakan 2 buah komputer dengan spesifikasi yang berbeda. Komputer pertama yaitu komputer yang menggunakan Intel atom N550, 2 *core* dengan kecepatan 1,5 GHz, *cache memory* L2 1Mb, tipe memori DDR3, ukuran memori 2 GB, sedangkan komputer kedua menggunakan Intel core i5 2320, 4 *core,* kecepatan 3 GHz, *cache memory* 6Mb, tipe memori DDR3-1333, ukuran memori 2 GB.

3.3 Prosedur Perhitungan

Langkah-langkah perhitungan yang harus dilakukan untuk mendapatkan penyelesaian dari persamaan difusi dapat dirangkum seperti berikut :

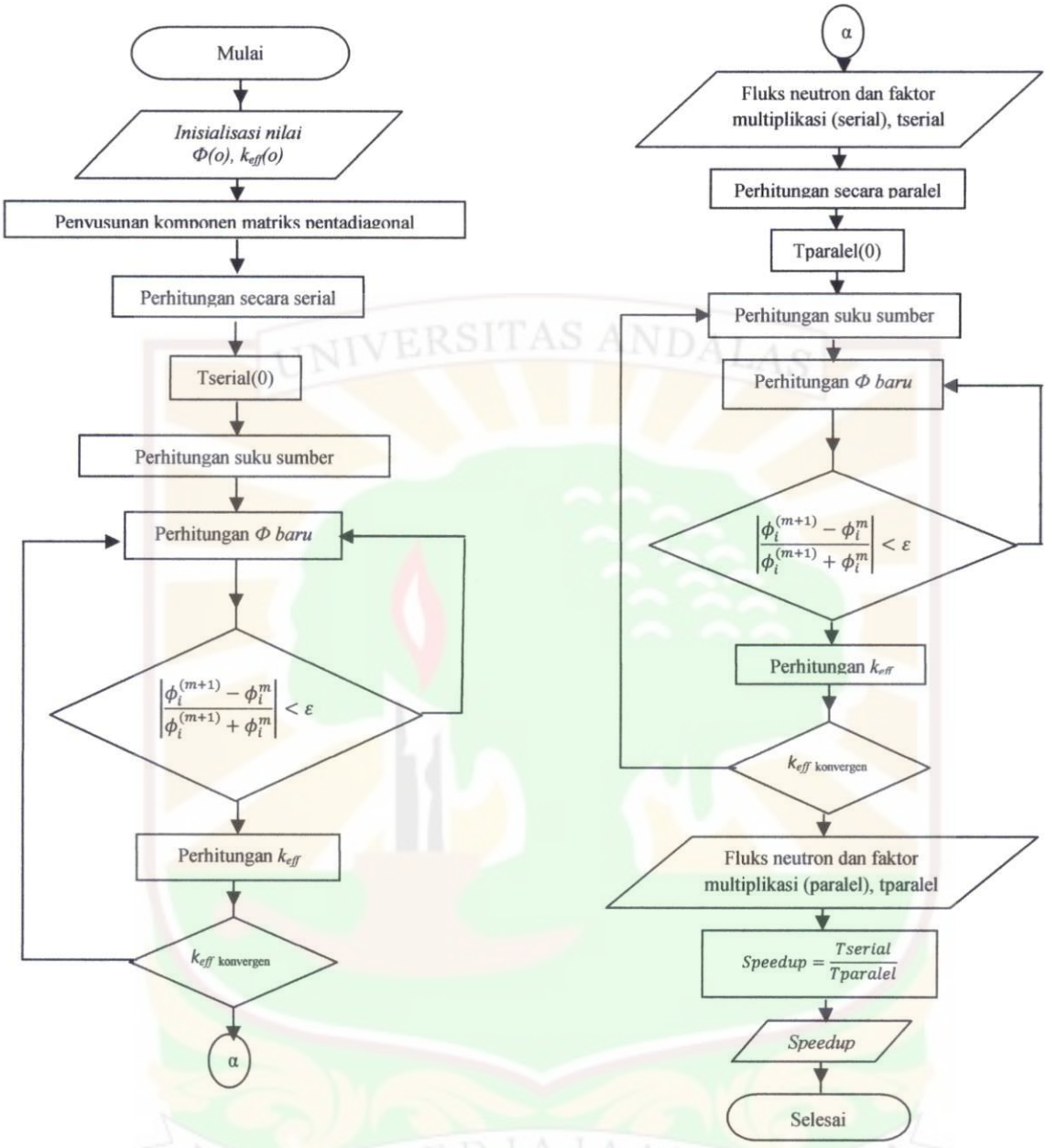
1. Tentukan nilai fluks awal dan nilai *keff* awal.
2. Hitung vektor sumber $S_{i,j}$ dari persamaan (2.25).
3. Hitung nilai fluks baru dari persamaan (2.35)
4. Hitung sumber fisi baru dengan persamaan berikut.

$$F^{m+1} = \sum_{i,j} \upsilon \sum_{f,i} \phi_{i,j}^{m} \tag{3.4}$$

5. Hitung *keff* baru dengan persamaan berikut

$$k_{eff}^{m+1} = k_{eff}^{m}(F^{m+1}/F^{m}) \tag{3.5}$$

6. Perhitungan *speedup*



Gambar 3.1 Diagram alir perhitungan difusi neutron.

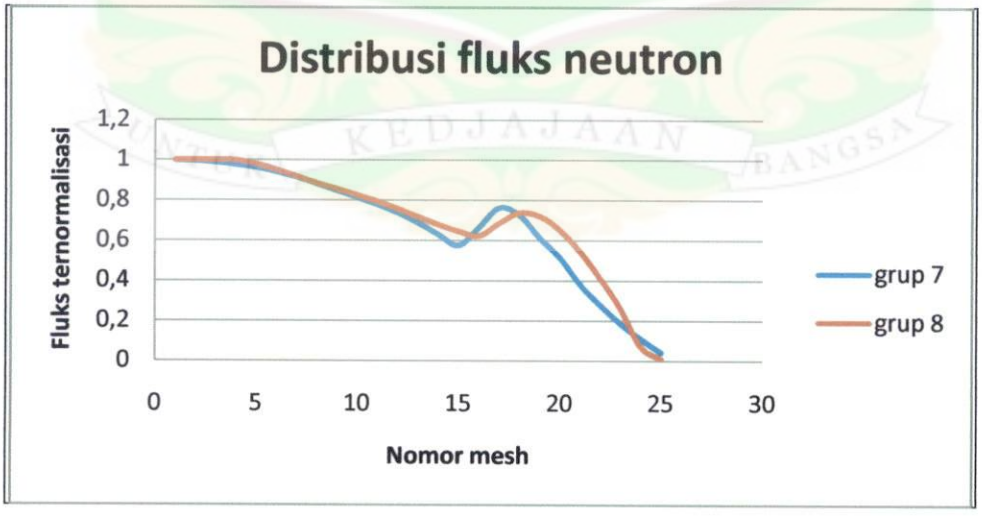
BAB IV

HASIL DAN PEMBAHASAN

4.1 Analisis Program

Perhitungan persamaan difusi neutron ditulis menggunakan program *microsoft visual studio* dan bahasa yang dipakai adalah bahasa pemprograman C++ (Lampiran 1). Program ini terbagi atas dua yaitu program secara serial dan program secara paralel. Masing-masing program mempunyai masukan dan keluaran yang sama. Sehingga dapat dilihat kinerja yang dihasilkannya.

Hasil yang diperoleh adalah distribusi fluks neutron. Dalam penelitian ini ditampilkan hasil running program berupa distribusi fluks neutron untuk neutron grup ke-7 dan dan neutron grup ke-8. Nilai distribusi fluks yang diperoleh kemudian dinormalisasi dan didapatkan grafik distribusi fluks neutron grup ke-8 pada Gambar 4.1. Dari gambar tersebut dapat dilihat bahwa masing-masing grup mempunyai pola distribusi yang sama.



Gambar 4.1 Distribusi fluks neutron untuk grup neutron ke-7 dan neutron grup ke-8 (neutron cepat)

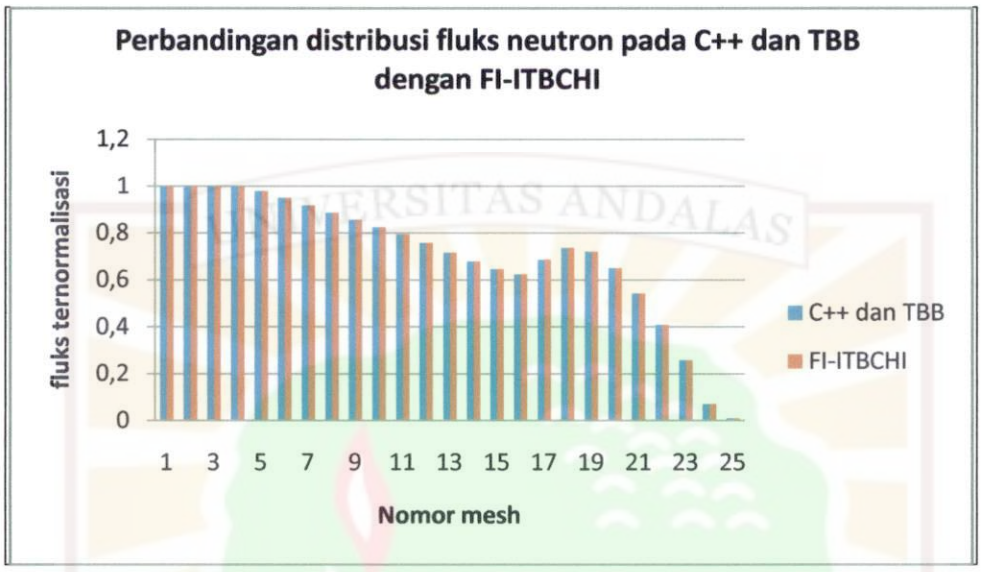
Untuk memvalidasi kode program yang dihasilkan, keluaran dari program ini dibandingkan dengan kode yang sudah dianggap standar (sudah *dibenchmark*). Dalam hal ini dibandingkan dengan kode program FI-ITBCHI yang dibuat Prof. Zaki Su'ud dari ITB.

Tabel 4.1 memperlihatkan bahwa hasil perhitungan kode program ini akurat hingga 5 angka penting (signifikan) jika dibandingkan dengan FI-ITBCHI.

Tabel 4.1 Perbandingan nilai fluks untuk grup ke-8

No	Fluks Neutron cepat (grup ke-8)	
	C++ dan TBB	**FI-ITBCHI**
1	1,000124	1,000112
2	1,000092	1,000089
3	1,000071	1,000072
4	0,998766	0,998766
5	0,979809	0,979809
6	0,951308	0,951308
7	0,918307	0,918307
8	0,885952	0,885952
9	0,856425	0,856425
10	0,825617	0,825617
11	0,794174	0,794177
12	0,757584	0,757591
13	0,717366	0,717367
14	0,679494	0,679493
15	0,646337	0,646345
16	0,624263	0,624255
17	0,687385	0,687397
18	0,737716	0,737732
19	0,721402	0,721456
20	0,651364	0,651363
21	0,543732	0,543744
22	0,409724	0,409727
23	0,258934	0,258929
24	0,071835	0,071838
25	0,010199	0,010197

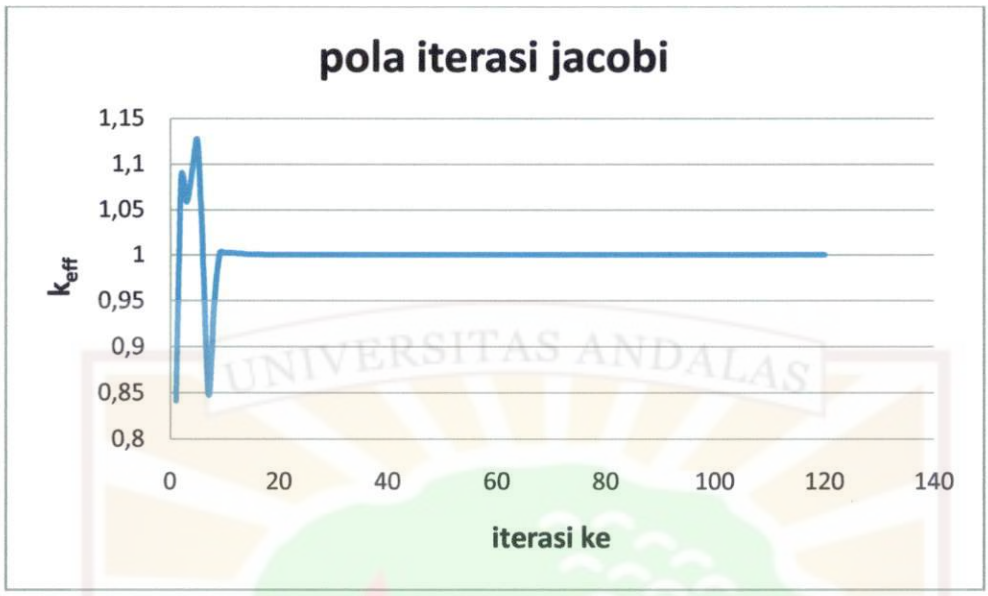
Gambar 4.2 merupakan perbandingan nilai fluks neutron untuk grup ke-8 (grup neutron cepat) antara program C++ dan TBB dengan FI-ITBCHI.



Gambar 4.2 Perbandingan distribusi Fluks neutron pada C++ dan TBB dengan FI-ITBCHI

Dari Gambar 4.2 tersebut dapat dilihat bahwa nilai fluks yang diperoleh pada masing-masing program hampir sama. Kedekatan hasil tersebut diupayakan melalui penggunaan semua masukan dan parameter yang sama. Beda yang mendasar adalah, FI-ITBCHI ditulis dengan bahasa *Delphy* dan menggunakan metode SOR dalam penyelesaian persamaan difusi sedangkan program penulis menggunakan C++ dan menggunakan metode Jacobi.

Dalam penelitian ini, penyelesaian persamaan difusi dilakukan secara iterasi Jacobi, dimana proses iterasi dilakukan sampai dicapai suatu nilai yang konvergen sampai toleransi yang diberikan. Pada penelitian ini batas kekonvergenan yang diinginkan sampai 10^{-6}.



Gambar 4.3 Pola osilasi nilai k_{eff} hingga mencapai nilai konvergennya pada versi serial

Gambar 4.3 merupakan nilai osilasi k_{eff} yang didapatkan pada program versi serial. K_{eff} yang didapatkan konvergen pada iterasi ke-120 hingga mencapai nilai konvergennya. Sedangkan untuk Gambar 4.4 merupakan osilasi dari k_{eff} pada versi paralel, yang konvergen pada iterasi ke-119.

Gambar 4.4 Pola osilasi nilai k_{eff} hingga mencapai nilai konvergennya pada versi serial

4.2 Analisis Speedup

Tabel 4.2 Hubungan nilai *grain size* dan *speedup* ketika dijalankan pada intel i5 2320 3,0 GHz (4 *core*)

No	*Grain Size*	Komputer intel i5 2320		*Speedup*
		Serial (detik)	Paralel (detik)	
1	100	2,022	0,781	2,589
2	200	2,013	0,767	2,623
3	300	2,014	0,783	2,570
4	400	2,006	0,781	2,568
5	500	2,017	0,670	3,009
6	600	2,009	0,669	3,007
7	700	2,018	0,675	2,986
8	800	2,017	0,913	2,208
9	900	2,864	1,046	2,737
10	1000	2,070	1,250	1,654
11	1100	2,019	1,337	1,510
12	1200	2,028	1,343	1,509
13	1300	2,029	1,322	1,536
14	1400	2,023	1,365	1,481
15	1500	2,079	1,346	1,534
16	1600	2,029	1,602	1,266
17	1700	2,017	1,681	1,199
18	1800	2,016	1,759	1,146
19	1900	2,017	1,902	1,061
20	2000	2,014	1,849	1,089

Setelah program berhasil dijalankan, maka didapatkan nilai *speedup* yang diperoleh dari perbandingan antara waktu menjalankan program secara serial dengan paralel. Pada Tabel 4.1 ditunjukkan hubungan antara *grain size* dengan *speedup*, dapat dilihat bahwa ada ukuran *grain* yang tepat agar menghasilkan *speedup* tertinggi.

Tabel 4.3 Hubungan nilai *grain size* dan *speedup* ketika dijalankan pada intel atom N550 1,5 GHz (2 *core*)

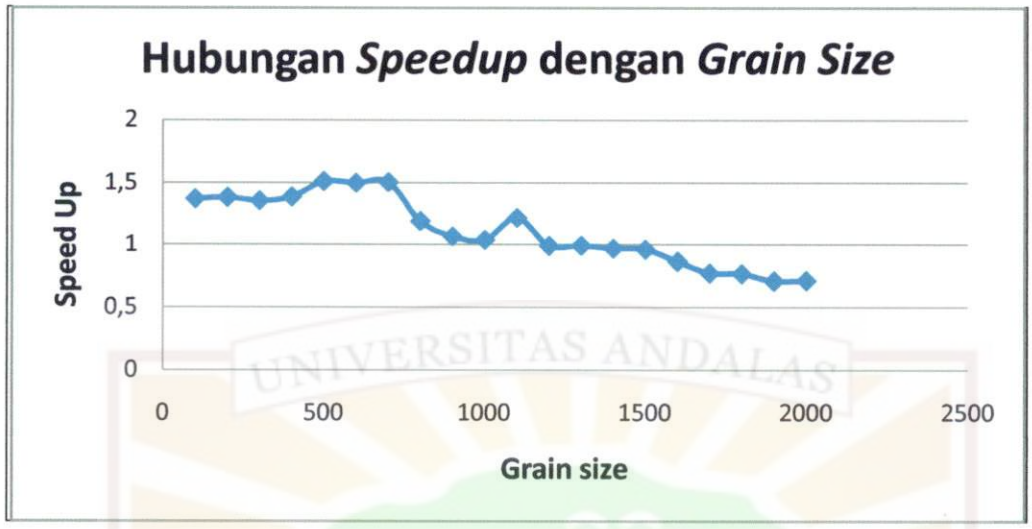
No	*Grain Size*	Komputer intel i5 2320		*Speedup*
		Serial (detik)	Paralel (detik)	
1	100	9,684	7,057	1,372
2	200	9,691	7,001	1,384
3	300	9,368	6,903	1,357
4	400	9,816	7,071	1,388
5	500	9,476	6,261	1,513
6	600	9,464	6,306	1,500
7	700	9,534	6,328	1,506
8	800	9,469	7,942	1,192
9	900	9,517	8,9	1,069
10	1000	9,639	9,267	1,040
11	1100	9,671	7,93	1,219
12	1200	9,379	9,436	0,993
13	1300	9,781	9,838	0,994
14	1400	9,754	10,045	0,971
15	1500	9,711	10,077	0,963
16	1600	9,365	10,797	0,867
17	1700	9,537	12,307	0,774
18	1800	9,474	12,304	0,769
19	1900	9,514	13,381	0,711
20	2000	9,869	13,7934	0,715

Grain digunakan pada saat memparalelkan program. *Grain* disini berfungsi untuk membagi *problem* menjadi bagian kecil yang diumpankan ke semua prosesor. Proses kinerja *grain size* dapat dianalogikan sebagai berikut. Jika dalam suatu program paralel dijalankan pada komputer *dual core*, menggunakan *grain* size misalnya 1000 dan data yang diolah sebanyak 6000, maka masing-masing prosesor akan mulai melakukan kerja 1000 pertama. Kecepatan dari masing-masing prosesor belum tentu sama sehingga, kemungkinan bisa terjadi bahwa satu dari prosesor tersebut bisa menyelesaikan perhitungan lebih cepat dari prosesor lainnya, sehingga prosesor ini akan menunggu sampai prosesor kedua

selesai. Hal ini menyebabkan data yang diperoleh (*speedup*) menjadi tidak akurat. *Grain size* yang semakin besar juga membuat proses *schedulling* dan sinkronisasi menjadi lama. Karena itu dengan mengecilkan nilai *grain size* dapat diharapkan mendapatkan nilai *speedup* yang lebih akurat. Karena prosesor itu akan bekerja terus, tanpa harus menunggu selesainya prosesor lainnya. Tetapi jika nilai *grain*-nya juga sangat kecil, maka hasil yang diperoleh juga tidak bagus.

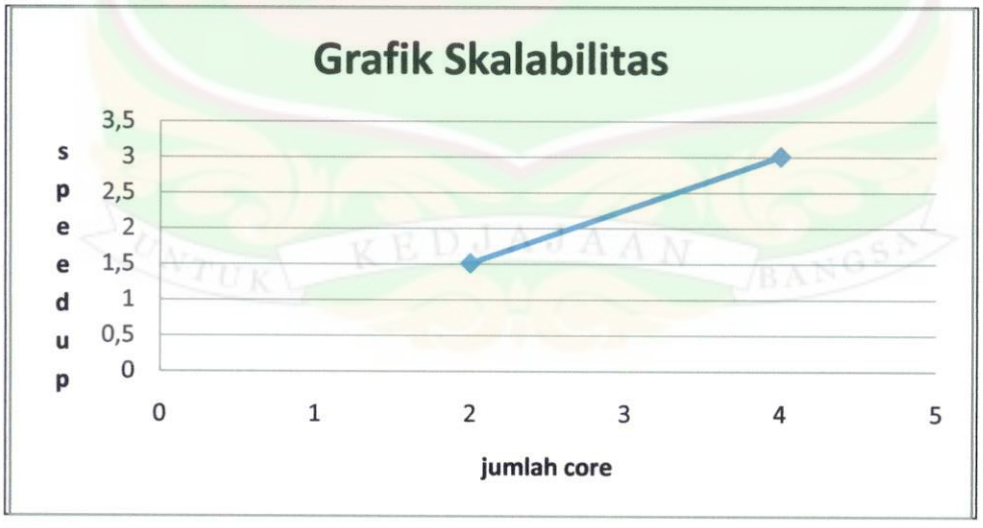
Nilai grain yang digunakan khas untuk tiap *problem*, artinya berbeda masalah berbeda pula nilai *grain size* yang tepat. Pada penelitian ini, untuk komputer intel i5 2320 3 GHz mempunyai nilai *speedup* tertinggi, yaitu 3,009 dicapai pada nilai *grain size* 500 (Gambar 4.5). Sedangkan untuk komputer intel atom N550 1,5 GHz mempunyai nilai speedup tertinggi yaitu 1,513 pada *grain size* yang sama yaitu 500 (Gambar 4.6).

Gambar 4.5 Hubungan *Speedup* dengan *Grain Size* pada komputer intel i5 2320 3,0 GHz (4 *core*)



Gambar 4.6 Hubungan *Speedup* dengan *Grain Size* pada komputer intel atom N550 1,5 GHz (2 *core*)

Program paralel yang baik haruslah bersifat *scalable*, artinya program harus berjalan lebih cepat sejalan dengan bertambahnya jumlah prosesor yang digunakan. Jika dijalankan pada 2-*core speedup* yang dicapai adalah 1,513 (Gambar 4.6) sedangkan pada 4-*core speedup*-nya 3,009 (Gambar 4.5).



Gambar 4.7 Skalabilitas program yang dihasilkan dengan intel TBB.

Keluaran program pada penelitian ini ada dua macam, pertama berupa jendela *console* dan yang kedua berupa file.txt (*notepad*). Keluaran dalam bentuk *console* ini berisikan proses yang dilaksanakan oleh program, dapat dilihat pada Lampiran 2. Sedangkan dalam bentuk file.txt, berisikan hasil keinginan yang diinginkan seperti distribusi fluks yang dihasilkan, nilai *keff* dan nilai *speedup* dari program ini (Lampiran 3, 4 dan 5).

BAB V

KESIMPULAN DAN SARAN

5.1 KESIMPULAN

Kode program perhitungan penyelesaian persamaan difusi neutron 2-dimensi dengan metode Jacobi paralel telah berhasil disusun ditulis dalam bahasa C++ dan intel TBB. *Speedup* tertinggi dicapai sebesar 3,09 yang dijalankan pada prosesor intel i5 2320 3GHz. Dan dengan intel atom N550 didapatkan speedup tertinggi yaitu 1,51. Program bersifat skalabel, sehingga jika ada prosesor dengan jumlah *core* lebih banyak, maka dapat diharapkan program dapat berjalan lebih cepat lagi.

Hasil *benchmarking* dengan program FI-ITBCHI menunjukkan bahwa kode yang ditulis menghasilkan keluaran fluks yang valid dengan tingkat kesamaan hingga 5 angka penting.

5.2 SARAN

Perlu dilakukan perbandingan dengan metode lain untuk dapat mengetahui efisiensi dari metode Jacobi paralel ini, karena dalam penelitian ini aspek tersebut belum diteliti.

DAFTAR KEPUSTAKAAN

Ariani, I, dan Dody K., "*Determination Of Optimum Parameters In Solving Neutron Diffusion Equation Using Finite Difference Method*", BATAN.

Duderstadt, J.J., dan Hamilton, L.J., 1976 , *Nuclear Reactor Analisis*, Jhon Wiley & Son Inc, Canada.

Grama, A., Gupta, A., Karypis, G., dan Kumar, V., (2003), *Introduction to Parallel Computing*, 2nd.ed., Addison Wesley, Reading, MA., 72-76.

El-Rewini, H, 2005, *"Advanced Computer Architecture and Parallel Processing"*, Jhon Wiley & Son Inc, Canada.

Reinders, J. 2007, " *Intel Threading Building Bloks* ", O'Reilly Media, Inc, United States of America.

Stacey, W. M., 2001, "*Nuclear Reactor Physics*", Jhon Wiley & Son Inc, Canada.

Su'ud, Z., 2001, Komputasi Paralel dalam Analisa Reaktor Nuklir, *Seminar Komputasi 2001*, Bandung.

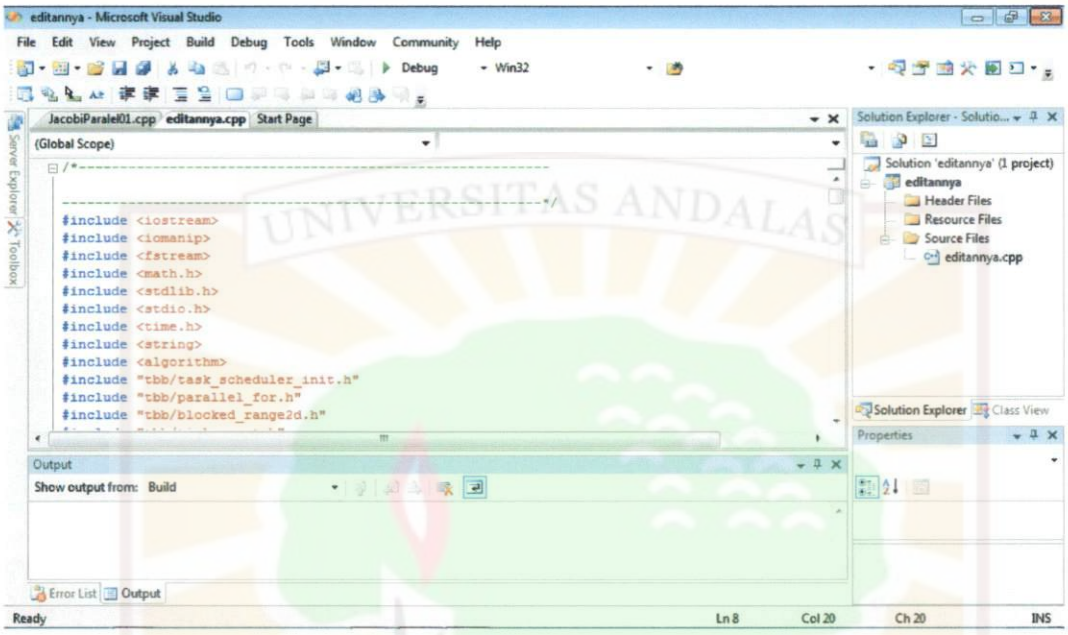
Taufiq, I., 2010, *Komputasi Paralel Persamaan Burnup Pada Reaktor Cepat dengan Pendingin Pb-Bi Menggunakan Pemrograman Multicore*, Disertasi S3, Jurusan Fisika FMIPA ITB, Bandung.

Varga, R. S., 2009, " *Matrix Iterative Analysis",* Springer Heidelberg Dordrecht London, New York.

Vesely, F. J., 1994, "*Computational Physics An Introduction*", Plenum Press. New York.

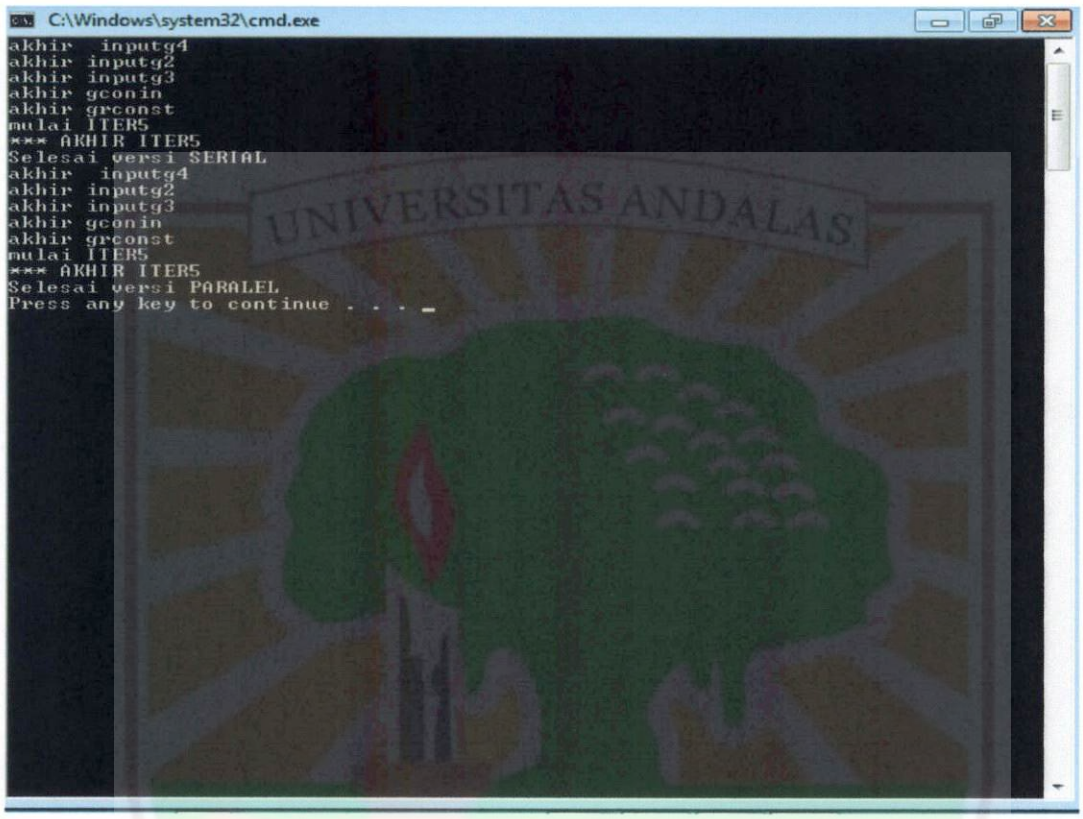
LAMPIRAN I

Pengetikan program pada Microsoft Visual Studio



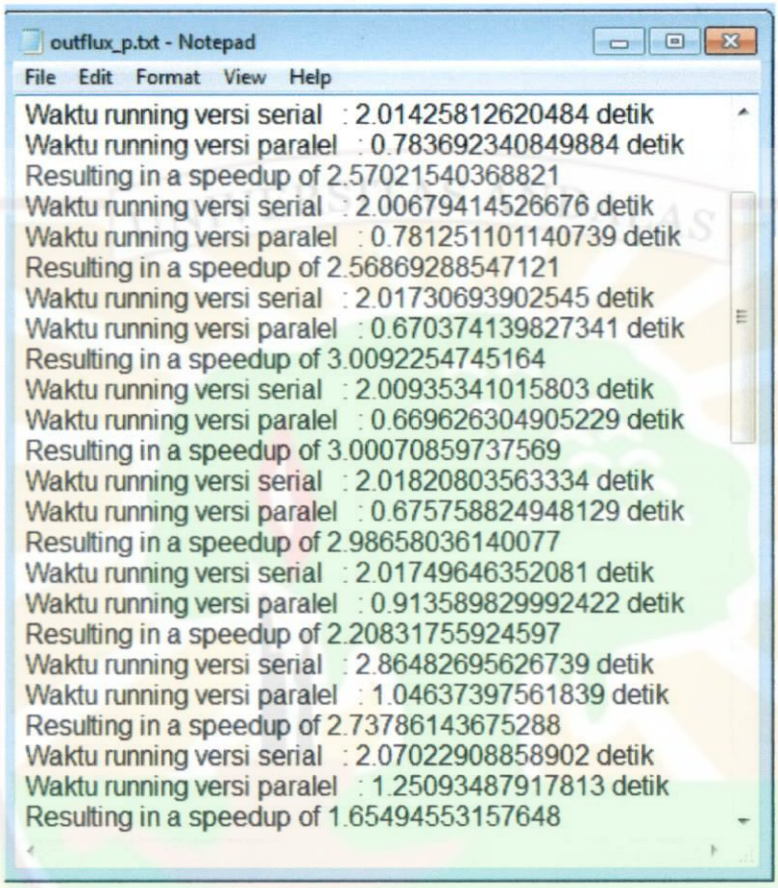
LAMPIRAN II

Bentuk keluaran pada jendela *console*



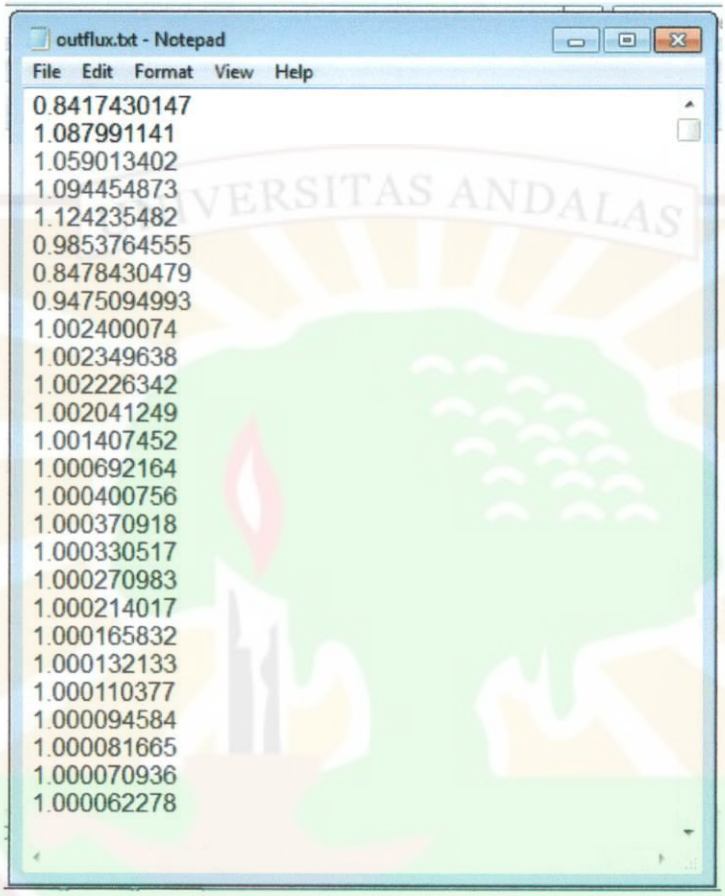
LAMPIRAN III

Hasil keluaran *speedup* dalam bentuk file.txt



LAMPIRAN IV

Hasil keluaran keff dalam bentuk file.txt



LAMPIRAN V

Hasil keluaran distribusi fluks dalam bentuk file.txt