ISSN 1979-4657

KOMPUTASI PARALEL PERSAMAAN DIFUSI NEUTRON PADA REAKTOR CEPAT DENGAN MENGGUNAKAN *INTEL THREADING BUILDING BLOCKS*

Imam Taufiq
Jurusan Fisika FMIPA Universitas Andalas
Kampus Limau Manis, Padang 25163
Email : imtaq69@yahoo.com

ABSTRAK

Telah dilakukan perhitungan persamaan difusi neutron 2-dimensi secara paralel dengan menggunakan persamaan Jacobi. Meskipun dikenal lebih lambat dibandingkan dengan algoritma SOR, namun iterasi Jacobi sangat mudah diparalelisasi. Program ditulis dalam bahasa C++ dengan bantuan program Intel *Threading Building Blocks*. Hasil running program menunjukkan bahwa program yang dibuat bersifat *scalable*, baik pada prosesor *single-core* maupun *multi-core*. Nilai *speedup* sebanding dengan jumlah *core* yang digunakan. 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 : Persamaan difusi, jacobi, parallel, *multicore*

1. PENDAHULUAN

Perkembangan teknologi mikroprosesor saat ini disamping senantiasa meningkatkan kecepatan prosesor juga ditandai dengan munculnya prosesor yang mengandung lebih dari satu piranti pemroses dalam satu keping, yang dikenal sebagai *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*. Namun pada dasarnya piranti *multicore processor* merupakan superkomputer skala kecil. Agar dapat memanfaatkan keuntungan dari kemampuan komputasinya, *software* atau program harus berbasis algoritma paralel (pemprograman paralel). Program-program yang ada selama ini 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 pemrograman 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 paralelisasi program. Dengan paralelisasi ini maka waktu pemrosesan menjadi lebih cepat.

Persoalan yang melibatkan model matematika banyak muncul dalam berbagai disiplin ilmu pengetahuan, seperti dalam bidang fisika, kimia, ekonomi atau pada persoalan rekayasa. Seringkali model matematika tersebut muncul dalam bentuk yang rumit, sehingga tidak dapat diselesaikan secara analitik. Untuk itu digunakan metode numerik yang pada umumnya lebih mudah dibandingkan dengan metode analitik biasa. Metode numerik banyak digunakan dalam menyelesaikan sistem persamaan yang berskala besar. Seperti contoh pada sistem persamaan diferensial, jika dalam skala kecil persamaan ini masih bisa diselesaikan dengan metode analitik tetapi jika sudah dalam ukuran yang besar, maka persamaan ini menjadi lebih rumit.

Dalam analisis reaktor, persamaan difusi seringkali dihitung secara berulang-ulang. Misalnya, untuk perhitungan *burnup* bahan bakar pada jangka waktu yang cukup lama, nilai fluks neutron harus diperbaharui secara berkala. Karenanya secara keseluruhan waktu perhitungan persamaan difusi menjadi cukup besar. Dengan demikian percepatan perhitungan persamaan difusi akan sangat mempercepat analisis reaktor secara keseluruhan. Persamaan difusi ini merupakan

persamaan yang cukup sulit dihitung secara paralel dengan menggunakan metoda iterasi *Successive Over Relaxation* (SOR) maupun dengan metoda Gauss-Seidel. Metoda Jacobi merupakan metoda iterasi lainnya, yang memungkinkan dilakukan paralelisasi.

Pada penelitian ini kode program perhitungan persamaan difusi neutron disusun dengan metoda Jacobi. Dengan dibangunnya kode program paralel ini diharapkan dapat mempercepat perhitungan persamaan difusi neutron, sehingga perhitungan analisis reaktor nuklir secara keseluruhan juga dapat dipercepat.

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.

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 (*speedup*) yang digunakan pada teknik paralel.

Paralelisasi 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.

Hukum Amdahl dapat digunakan untuk memperkirakan nilai *speedup* yang bisa diperoleh jika kita mengubah suatu program serial menjadi program paralel.

Tparalel = {(1-*P*) + *P*/Jumlah prosesor}*Tserial*+*overhead*

dengan *Tparalel* adalah waktu yang diperlukan oleh program versi paralel, *P* adalah fraksi program yang dapat diparalelisasi, *Tserial* adalah waktu yang diperlukan oleh program versi serial. *Overhead* muncul pada eksekusi versi paralel dan tidak muncul pada program versi serial, akibat adanya proses penjadualan (*schedulling*) bagian program yang akan diproses pada tiap-tiap *core* (Hesyam, 2005). Dalam penelitian ini digunakan perhitungan *speedup* dengan rumusan sebagai berikut :

$$Speedup = \frac{Tserial}{Tparalel} \quad (1)$$

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

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.

Intel Threading Building Blocks

Intel Threading Building Blocks (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 Threading building block membantu untuk menciptakan aplikasi yang dapat memberi keuntungan untuk mengolah dengan prosesor yang semakin banyak. *Threading building block* 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 *threading block* dapat dieksekusi pada sistem prosesor ganda, begitu juga pada sistem multiprosesor sehingga dapat menghasilkan program paralel yang bersifat *scalable*. Selain itu *Threading Building Block* juga dapat mendukung paralelisasi bertingkat, sehingga dapat membangun komponen paralel besar dari komponen paralel yang kecil dengan relatif mudah (Reinders, 2007).

Persamaan Difusi Neutron

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*-1, *I* dan *j* = 1, 2, ..., *J*-1, *J*.

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

$$-\vec{\nabla}.D(\vec{r})\vec{\nabla}\phi(\vec{r}) + \Sigma_a(\vec{r})\phi(\vec{r}) = \frac{1}{k_{eff}}\upsilon\Sigma_f(\vec{r})\phi(\vec{r}) \quad (2)$$

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

$$\int_{ij} -\vec{\nabla}.D(\vec{r})\vec{\nabla}\phi(\vec{r})dV + \int_{ij}\Sigma_a(\vec{r})\phi(\vec{r})dV = \int_{ij}\frac{1}{k_{eff}}\upsilon\Sigma_f(\vec{r})\phi(\vec{r})dV \quad (3)$$

Suku kedua di ruas kiri dan suku tunggal di ruas kanan dari persamaan (3) dapat dengan mudah dintegrasikan, dan suku pertama ruas kiri dapat diubah menjadi integral luasan, sehingga persamaan di atas dapat diubah menjadi

$$-\oint_{ij}D(\vec{r})\vec{\nabla}\phi(\vec{r}).d\vec{A} + \Sigma_{a,i,j}\phi_{i,j}V_{i,j} = \frac{1}{k_{eff}}\upsilon\Sigma_{f,i,j}\phi_{i,j}V_{i,j} \quad (4)$$

Selanjutnya suku pertama ruas kiri dari persamaan (4), dijabarkan menjadi

$$-\oint_{ij}D(\vec{r})\vec{\nabla}\phi(\vec{r}).d\vec{A} = -\int_{i+\frac{1}{2}}D(\vec{r})\vec{\nabla}\phi(\vec{r}).d\vec{A} + \int_{i-\frac{1}{2}}D(\vec{r})\vec{\nabla}\phi(\vec{r}).d\vec{A} - \int_{j+\frac{1}{2}}D(\vec{r})\vec{\nabla}\phi(\vec{r}).d\vec{A} + \int_{j-\frac{1}{2}}D(\vec{r})\vec{\nabla}\phi(\vec{r}).d\vec{A} \quad (5)$$

Setelah dilakukan diskritasi, maka didapatkan persamaan difusi menjadi sebagai berikut :

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

$$-D^{i,j+\frac{1}{2}}\frac{(\phi_{i,j+1}-\phi_{i,j})}{z_{j+1}-z_j}A_{i,j+\frac{1}{2}}+D^{i,j-\frac{1}{2}}\frac{(\phi_{i,j}-\phi_{i,j-1})}{z_j-z_{j-1}}A_{i,j-\frac{1}{2}}+\Sigma_{a,i,j}\phi_{i,j}V_{i,j}=\frac{1}{k_{eff}}\upsilon\Sigma_{f,i,j}\phi_{i,j}V_{i,j} \quad (6)$$

Jika dikelompokkan kembali berdasarkan fluks neutronnya, dapat dituliskan menjadi

$$\phi_{i,j-1}\left(-\frac{D^{i,j-\frac{1}{2}}}{z_j-z_{j-1}}A_{i,j-\frac{1}{2}}\right)+\phi_{i-1,j}\left(-\frac{D^{i-\frac{1}{2}j}}{r_i-r_{i-1}}A_{i+\frac{1}{2}j}\right)$$
$$+\phi_{i,j}\left(\frac{D^{i,j+\frac{1}{2}}}{z_{j+1}-z_j}A_{i,j+\frac{1}{2}}+\frac{D^{i+\frac{1}{2}j}}{r_{i+1}-r_i}A_{i+\frac{1}{2}j}+\frac{D^{i,j-\frac{1}{2}}}{z_j-z_{j-1}}A_{i,j-\frac{1}{2}}+\frac{D^{i-\frac{1}{2}j}}{r_{i-1}-r_i}A_{i-\frac{1}{2}j}+\Sigma_{a,i,j}\phi_{i,j}V_{i,j}\right)$$
$$+\phi_{i+1,j}\left(\frac{-D^{i+\frac{1}{2}j}}{r_{i+1}-r_i}A_{i+\frac{1}{2}j}\right)+\phi_{i,j+1}\left(\frac{-D^{i,j+\frac{1}{2}}}{z_j-z_{j+1}}A_{i,j+\frac{1}{2}}\right)$$
$$=\frac{1}{k_{eff}}\upsilon\Sigma_{f,i,j}\phi_{i,j}V_{i,j} \quad (7)$$

Persamaan (7) dapat diringkas penulisannya menjadi

$$-\gamma_{i,j}\phi_{i,j-1}-\alpha_{i,j}\phi_{i-1,j}+\beta_{i,j}\phi_{i,j}-\alpha_{i,j}\phi_{i+1,j}-\gamma_{i,j+1}\phi_{i,j+1}=S_{i,j} \quad (8)$$

dengan mendefinisikan variabel-variabel baru sebagai berikut

$$\gamma_{i,j}=-\frac{D^{i,j-\frac{1}{2}}}{z_j-z_{j-1}}A_{i,j-\frac{1}{2}} \quad (9)$$

$$\alpha_{i,j}=-\frac{D^{i-\frac{1}{2}j}}{r_i-r_{i-1}}A_{i+\frac{1}{2}j}=\frac{-D^{i+\frac{1}{2}j}}{r_{i+1}-r_i}A_{i+\frac{1}{2}j} \quad (10)$$

$$\beta_{i,j}=\frac{D^{i,j+\frac{1}{2}}}{z_{j+1}-z_j}A_{i,j+\frac{1}{2}}+\frac{D^{i+\frac{1}{2}j}}{r_{i+1}-r_i}A_{i+\frac{1}{2}j}+\frac{D^{i,j-\frac{1}{2}}}{z_j-z_{j-1}}A_{i,j-\frac{1}{2}}+\frac{D^{i-\frac{1}{2}j}}{r_{i-1}-r_i}A_{i-\frac{1}{2}j}+\Sigma_{a,i,j}\phi_{i,j}V_{i,j} \quad (11)$$

$$\gamma_{i,j+1}=\frac{-D^{i,j+\frac{1}{2}}}{z_j-z_{j+1}}A_{i,j+\frac{1}{2}} \quad (12)$$

$$S_{i,j}=\frac{1}{k_{eff}}\upsilon\Sigma_{f,i,j}\phi_{i,j}V_{i,j} \quad (13)$$

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

$$\phi(R+0,71\lambda_{tr})=0 \;\Big|\; \left.\frac{d\phi}{dr}\right|_{r=0}=0 \;\Big|\; \phi\left(r,\frac{h}{2}+0,71\lambda_{tr}\right)=0 \;\Big|\; \phi\left(r,-\frac{h}{2}+0,71\lambda_{tr}\right)=0 \quad (14)$$

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 (Duderstadt, 1976).

Kemudian persamaan (11) 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

$$\mathbf{A\Phi = S} \tag{15}$$

Metoda Iterasi

Metode-metode solusi numerik yang banyak dipakai, dapat diklasifikasikan sebagai metoda langsung dan metoda tidak langsung

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

2. METODE PENELITIAN

Penelitian ini dilakukan di Laboratorium Komputer Jurusan Fisika Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Andalas pada bulan Juni 2011. Dengan menggunakan bahasa pemprograman C++ pada *Microsoft Visual Studio* 2005 dengan menggunakan *template intel Threading Building Block.*

Pada penelitian ini akan dilakukan perhitungan fluks neutron. Dari penyelesaian persamaan difusi neutron (persamaan 15) dapat diperoleh nilai fluks neutron disetiap *mesh* spasial yang ditinjau. Jika matriks A merupakan matriks singular maka untuk dapat menyelesaikan persamaan ini menggunakan (Varga, 2009)

$$\Phi = \mathrm{A}^{-1}\,\mathrm{S} \tag{16}$$

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)

$$\mathbf{A = D + U + L} \tag{17}$$

Dengan demikian persamaan (15) dapat dituliskan sebagai (Varga, 2009)

$$\mathbf{D\Phi = (U + L)\,\Phi + S} \tag{18}$$

Persamaan iteratif dari persamaan (18) dapat ditulis (Varga, 2009)

$$a_{i,i}\phi_i^{m+1} = -\Sigma_{j=1}^{n}\, a_{i,j}\phi_i^m + S_i\,, 1 \le i \le n, m \ge 0,\ \ j \ne i \tag{19}$$

Dengan ϕ_i^0 adalah nilai tebakan awal. Kemudian persamaan (19) dapat ditulis ke dalam bentuk (Varga, 2009)

$$\phi_i^{m+1} = -\Sigma_{j=1}^{n} \frac{a_{i,j}}{a_{i,i}} \phi_i^m + \frac{S_i}{a_{i,i}}, 1 \leq i \leq n, m \geq 0, \quad (20)$$

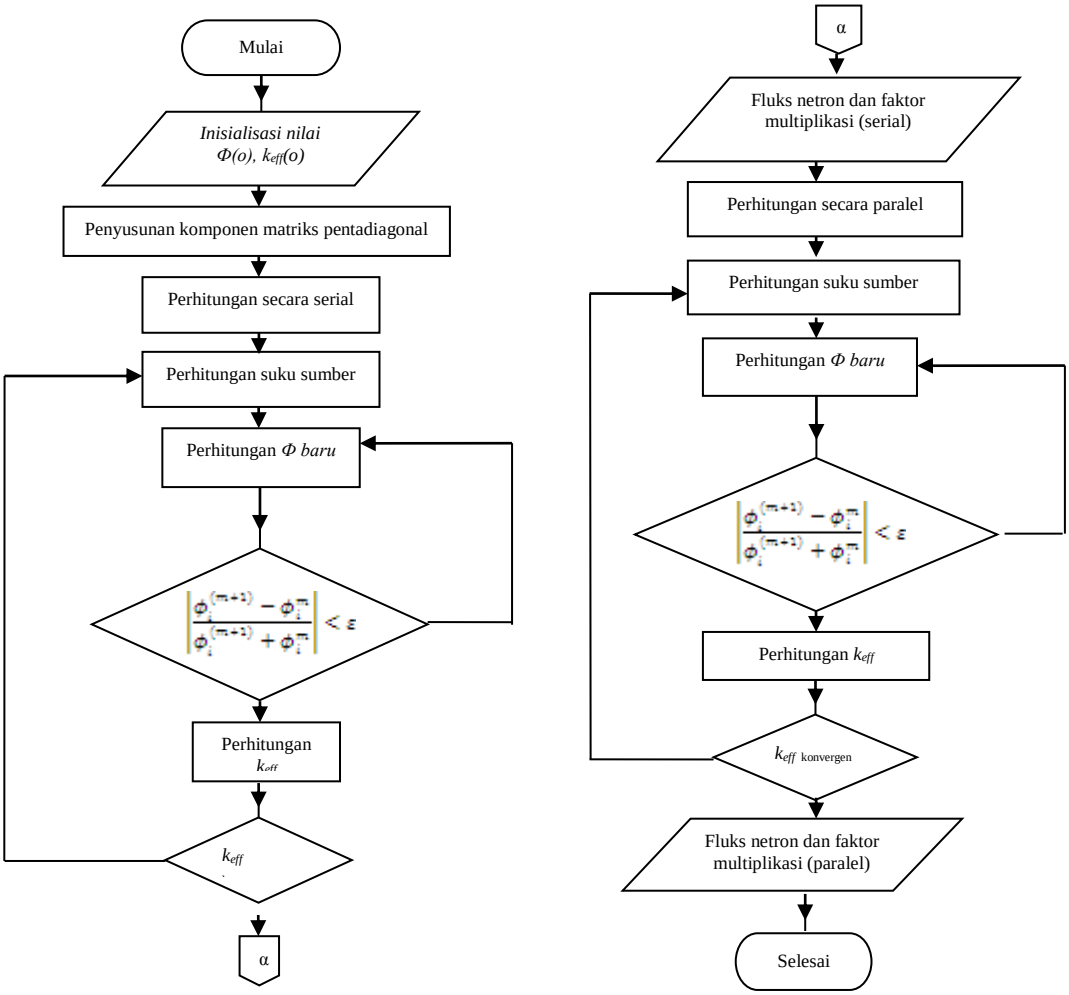
Perhitungan iteratif seperti persamaan (20) disebut sebagai metode Jacobi, atau dikenal dengan *total-step iterative method* (Varga, 2009).

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 skema sebagai berikut (Duderstat, 1976)

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

Sehingga kita dapat menyelesaikan nilai fluk $m+1$ dengan persamaan (Duderstat, 1976).

$$\phi_i^{m+1} = -\Sigma_{j=1}^{n} \frac{a_{i,j}}{a_{i,i}} \phi_i^m + \frac{S_i}{a_{i,i}}, i = 1, 2 \ldots, n \quad (22)$$



Gambar 3 Diagram alir perhitungan persamaan difusi neutron.

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

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 (13).
3. Hitung nilai fluks baru dari persamaan (15)
4. Hitung sumber fisi baru dengan persamaan berikut.
$$F^{m+1} = \sum_{i,j} \upsilon \sum_{f,i} \phi_{i,j}^m \quad (23)$$
5. Hitung *keff* baru dengan persamaan berikut
$$k_{eff}^{m+1} = k_{eff}^m (F^{m+1}/F^m) \quad (24)$$

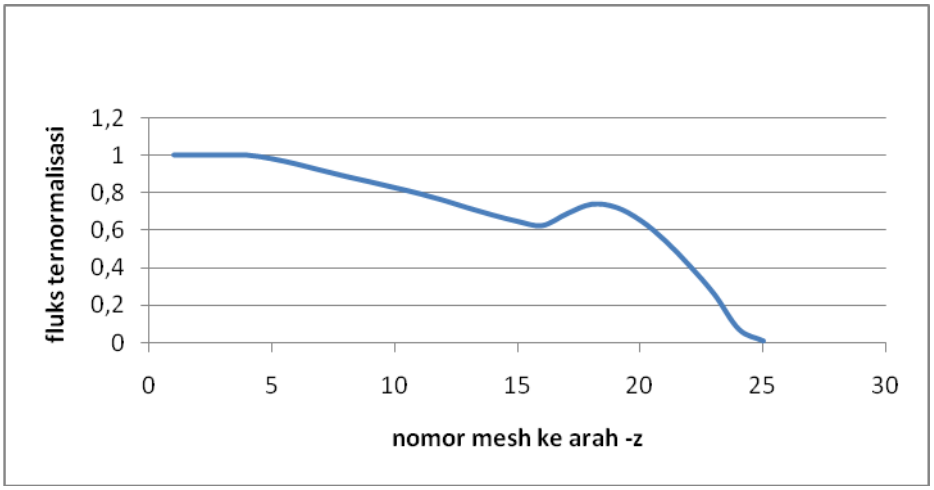
3. HASIL DAN PEMBAHASAN

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.

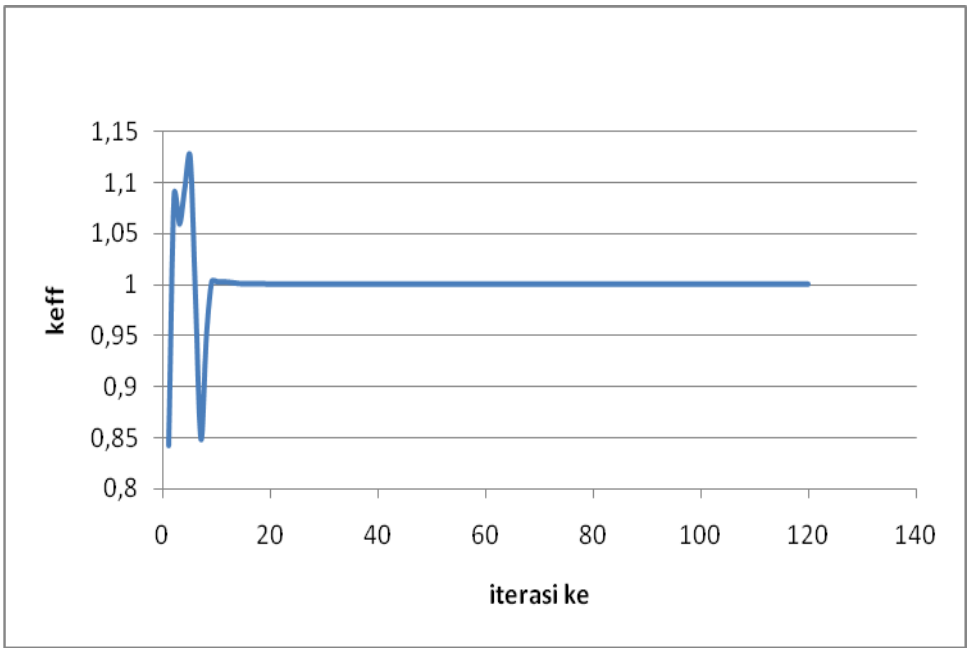
Hasil running program pada gambar 1 tampak profil distribusi neutron grup cepat ke arah sumbu-z. Hasil tersebut diperoleh setelah iterasi ke 120 pada program versi serial dan iterasi ke 119 pada program versi paralel. Pola osilasi pada saat iterasi hingga mencapai nilai konvergen tampak pada gambar 2.

Setelah program berhasil dijalankan, maka didapatkan nilai *speedup* yang diperoleh dari perbandingan antara waktu menjalankan program secara serial dengan paralel. Pada Tabel 2 ditunjukkan hubungan antara *grain size* dengan *speedup*, dapat dilihat bahwa ada ukuran *grain* yang tepat agar menghasilkan *speedup* tertinggi. Hal ini khas untuk tiap problem, artinya berbeda masalah berbeda pula nilai *grain size* yang tepat. Untuk kasus ini nilai *speedup* tertinggi, yakni 3,09 dicapai pada nilai *grain size* 500. Pola hubungan antara *grain size* dan *speedup* ini ditunjukkan pada Gambar 3.

Program paralel yang baik haruslah bersifat *scalable*, artinya program harus berjalan lebih cepat sejalan dengan bertambahnya jumlah prosesor yang digunakan. Pada gambar 4 terlihat bahwa program persamaan difusi dengan iterasi Jacobi memenuhi kriteria tersebut. Jika dijalankan pada *single-core* nilai *speedup* yang dicapai adalah 0,99, sedangkan pada 2-*core* dan 4-*core* masing-masing adalah 1,51 dan 3,09.



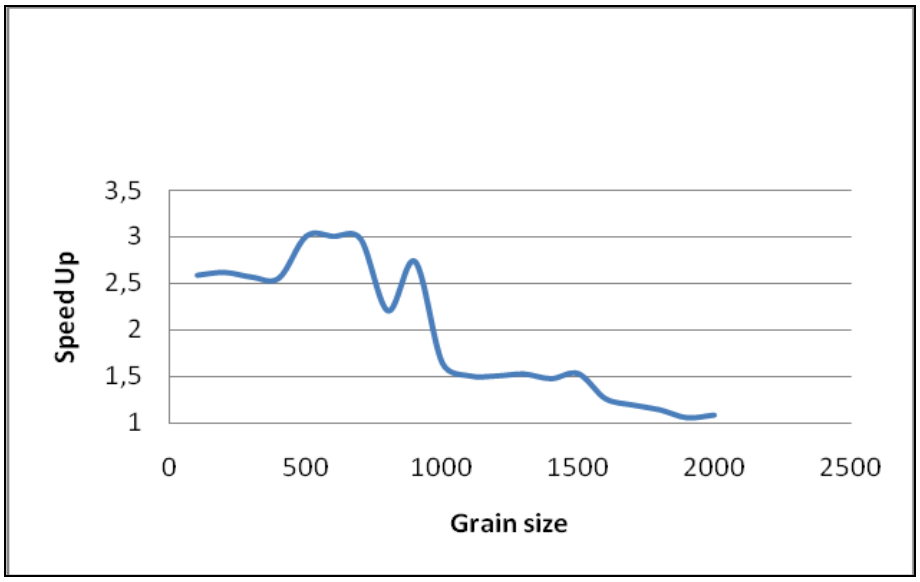
Gambar 1. Distribusi Fluks ke arah sumbu-z untuk grup neutron ke-8 (grup neutron cepat)



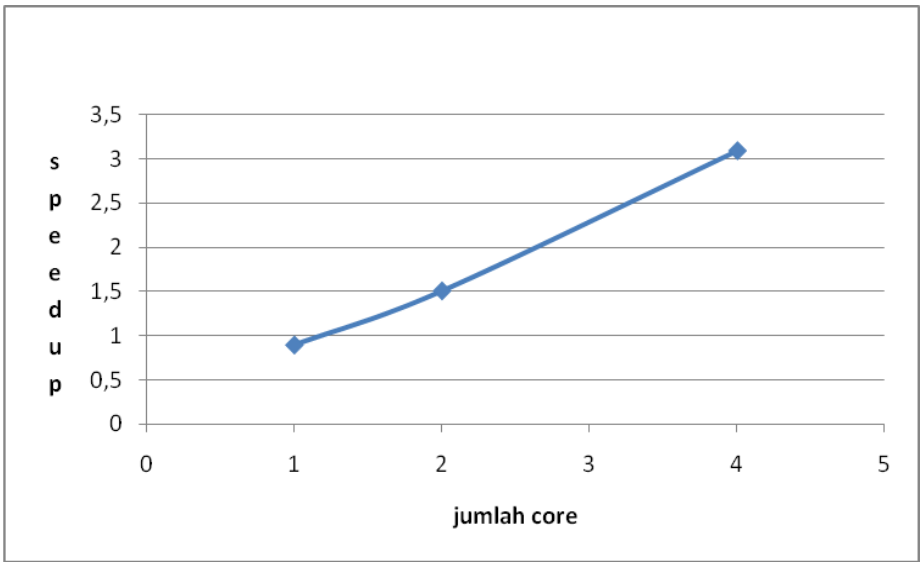
Gambar 2. Pola osilasi nilai keff hingga mencapai nilai konvergennya.

Tabel 1. Hubungan nilai *grain size* dan *speedup* ketika dijalankan pada intel i5 2320 3,0 GHz (4 *core*)

No	*Grain Size*	Komputer 2		*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



Gambar 3. Hubungan Speedup dengan Grain Size pada komputer 1



Gambar 4. Skalabilitas program yang ditulis dengan intel TBB.

4. KESIMPULAN

Kode program penyelesaian persamaan difusi neutron 2-dimensi dengan metode Jacobi paralel yang ditulis dalam bahasa C++ dan intel TBB telah berhasil dibangun dan memiliki keunggulan sebagaimana diharapkan pada sebuah program paralel. *Speedup* tertinggi dicapai sebesar 3,09 yang dijalankan pada prosesor intel i5 2320 3GHz (4-*core*). Program dengan menggunakan Intel TBB ini bersifat *scalable*, sehingga jika ada prosesor dengan jumlah *core* lebih banyak, maka dapat diharapkan program dapat berjalan lebih cepat lagi. Di masa depan semua program paralel harus mempunyai sifat skalabilitas yang baik agar dapat meningkat kinerjanya bersama dengan kemajuan perangkat keras komputer.

DAFTAR PUSTAKA

1. Ariani, I, Dody K., "Determination Of Optimum Parameters In Solving Neutron Diffusion Equation Using Finite Difference Method**"**, BATAN.
2. Duderstadt, J.J., 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.
3. El-Rewini, Hesyam, 2005, "Advanced Computer Architecture and Parallel Processing", Jhon Wiley & Son Inc, Canada.
4. Reinders, J. 2007, " Intel Threading Building Bloks ", O'Reilly Media, Inc, United States of America.
5. Stacey, W. M., 2001, "Nuclear Reactor Physics", Jhon Wiley & Son Inc, Canada.
6. Su'ud, Z., (2001), Komputasi Paralel dalam Analisa Reaktor Nuklir, Seminar Komputasi 2001, Bandung.
7. Taufiq, I., 2010, Komputasi Paralel Persamaan Burnup Pada Reaktor Cepat dengan Pendingin Pb-Bi Menggunakan Pemrograman Multicore, Disertasi S3, Jurusan Fisika FMIPA ITB, Bandung.
8. Varga, R. S., 2009, " Matrix Iterative Analysis", Springer Heidelberg Dordrecht London, New York.
9. Vesely, F. J., 1994, "Computational Physics An Introduction", Plenum Press. New York.