Implementasi *Hypergraph Partitioning* pada Paralelisasi Perkalian Matriks-Vektor

Murni dan Tri Handhika
Pusat Studi Komputasi Matematika
Universitas Gunadarma, Depok
{murnipskm, trihandika}@staff.gunadarma.ac.id

Abstrak

Proses perhitungan perkalian matriks-vektor dalam penyelesaian masalah di dunia nyata seringkali melibatkan matriks dengan ukuran sembarang dan besar. Oleh sebab itu, paralelisasi dilakukan untuk mempercepat proses perhitungan tersebut yang biasanya memerlukan waktu lama. Makalah ini membahas paralelisasi yang efisien untuk perkalian matriks-vektor melalui teknik graf. Teknik *graph partitioning* yang telah dibahas pada penelitian sebelumnya tidak dapat digunakan dalam menyelesaikan perhitungan perkalian matriks-vektor dengan ukuran matriks sembarang. Hal ini disebabkan oleh asumsi dari teknik *graph partitioning* yang hanya dapat menyelesaikan matriks persegi dan simetris saja. Adapun implementasi yang ditunjukkan pada makalah ini adalah perkalian matriks dengan ukuran 4×3 dan vector berukuran 3×1 dimana matriks tersebut bukan matriks persegi ataupun simetris sehingga teknik *graph partitioning* tidak dapat digunakan. Dengan demikian, teknik *hypergraph partitioning* dipilih untuk mengatasi kekurangan dari teknik *graph partitioning* tersebut.

Kata kunci: *graph partitioning, hypergraph partitioning, paralelisasi, matriks-vektor*

Pendahuluan

Perhitungan perkalian matriks-vektor sangat dibutuhkan dalam menyelesaikan berbagai permasalahan perhitungan di dunia nyata, contohnya dalam menentukan solusi sistem persamaan linier. Ukuran matriks yang besar memunculkan kendala waktu yang lama dari proses perhitungan tersebut. Hal ini berakibat pada akurasi hasil perhitungan yang tidak *real-time.* Beberapa penelitian terdahulu mencoba melakukan paralelisasi terhadap perhitungan perkalian matriks-vektor salah satunya dengan mengadopsi konsep graf, yaituteknik*graph partitioning* [4, 6, 9]. Namun, terdapat asumsi yang harus dipenuhi dalam mengimplementasikan teknik *graph partitioning* tersebut dimana matriks harus berukuran persegi dan simetris. Padahal, pada permasalahan dunia nyata ukuran matriks tidak hanya terbatas pada persegi dan simetris saja.

Pada makalah ini penulis menggunakan teknik *hypergraph partitioning* untuk mengatasi kekurangan dari teknik *graph partitioning* dimana teknik ini dapat digunakan untuk sembarang ukurang matriks. Berbeda dengan teknik *graph partitioning*, pada teknik *hypergraph partitioning* ini setiap*edge* dapat menghubungkan lebih dari dua verteks atau biasa disebut sebagai *net* (*hyperedge*). Dengan demikian, selanjutnya ditunjukkan proses perhitungan perkalian matriks-vektor secara parallel disertai dengan contoh melaluiteknik*hypergraph partitioning*.

Hypergraph Partitioning

Misalkan *hypergraph* tidak berarah dinotasikan sebagai $\mathcal{H} = (V, N)$ dengan V adalah himpunan tak kosong dari verteks-verteks dimana$n = |V|$ menyatakan banyaknya verteks. Sedangkan, N adalah himpunan *net* yang menghubungkan verteks-verteks tersebut. Setiap *net*, $n_j \epsilon N$, merupakan subset dari verteks ($n_j \subseteq V$). Verteks-verteks pada n_j disebut dengan *pins* dan dinotasikan sebagai $pins[n_j]$. Ukuran dari *net* sama dengan banyaknya *pins*, dinotasikan dengan $s_j = |pins[n_j]|$. Himpunan *net* yang terhubung dengan suatu verteks, misal v_i, dinotasikan dengan $nets[v_i]$. Selain itu, derajat dari suatu verteks merupakan banyaknya *net* yang terhubung dengan verteks tersebut, dinotasikan dengan $d_i = |nets[v_i]|$. Graf merupakan kasus khusus dari *hypergraph* sedemikian sehingga setiap *net* memiliki tepat 2

pin. Seperti halnya pada graf, bobot verteks dan *net* juga didefinisikan pada *hypergraph*. Misalkan $w_i$menotasikan bobot dari $v_i \epsilon V$ dan c_j menotasikan *cost* dari $n_j \epsilon N$.

Selanjutnya, misalkan pula partisi$\Pi = \{\wp_1, \wp_2, \dots, \wp_{\mathrm{K}}\}$adalah himpunan *K* partisi pada $\mathcal{H}$. *Net* yang memiliki paling sedikit satu *pin* pada suatu partisi dikatakan *connect* dengan partisi tersebut. Himpunan *connectivity*Λ_j dari n_j didefinisikan sebagai himpunan partisi yang dihubungkan oleh n_j. Adapun banyaknya partisi yang dihubungkan oleh n_j dinotasikan dengan $\lambda_j = |n_j|$. n_j dikatakan *cut* jika terhubung lebih dari satu partisi $(\lambda_j > 1)$ dan sebaliknya dikatakan *uncut*$(\lambda_j = 1)$. Adapun masing-masing *cut* dan *uncutnet* pada *hypergraph* ini secara berurutan disebut pula dengan *net* eksternal dan internal. Himpunan *net* eksternal pada partisi Π dinotasikan dengan N_E. Selanjutnya, diberikan dua definisi *cutsize* untuk merepresentasikan total *cost* pada partisi Π yaitu:

$$\chi(\Pi) = \sum_{n_j \epsilon N_E} c_j \quad (1)$$

dan

$$\chi(\Pi) = \sum_{n_j \epsilon N_E} c_j \left(\lambda_j - 1\right) \quad (2)$$

Pada persamaan (1), setiap *cut net*n_j berkontribusi pada *cutsize* sebesar c_j. Sedangkan, pada persamaan (2), setiap *cutnet*n_j memberikan kontribusi sebesar $c_j(\lambda_j - 1)$ pada *cutsize*. Dengan demikian, permasalahan *hypergraph partitioning* dapat didefinisikan sebagai teknik untuk mempartisi suatu *hypergraph* menjadi dua bagian atau lebih sedemikian sehingga diperoleh *cutsize* minimum dengan *balance criterion* sebagai berikut [1, 2]:

$W_k \leq W_{avg}(1 + \varepsilon)$, untuk $k = 1,2,\cdots,\mathrm{K}$

dimana

$$W_k = \sum_{v_i \epsilon \wp_{\mathrm{k}}} w_i \text{ dan } W_{avg} = \frac{\sum_{v_i \epsilon V} w_i}{\mathrm{K}}$$

serta ε (maksimum rasio *imbalance* yang ditentukan) merupakan suatu bilangan kecil nonnegatif.

Adapun algoritma optimisasi yang digunakan dalam teknik *hypergraph partitioning* pada makalah ini adalah algoritma Fiduccia-Mattheyses (atau dikenal dengan algoritma FM)[3]. Konsep algoritma ini hampir sama dengan algoritma Kernighan-Lin [8] yang terbatas hanya dapat digunakan untuk kasus dua partisi saja. Perbedaannya adalah jika pada algoritma Kernighan-Lin menerapkan konsep penukaran sepasang verteks pada setiap iterasinya maka tidak demikian halnya dengan algoritma FM. Konsep dari algoritma FM ini adalah menukar satu per-satu verteks pada setiap iterasi. Pertama-tama, definisikan *initial state,* yakni himpunan verteks yang secara acak dipartisi menjadi dua bagian. Selanjutnya, setiap verteks akan berpindah posisi dari partisi yang satu ke partisi lainnya dengan konsep *gain*, yaitu nilai yang menandakan pengurangan *cutsize*. *Gain* yang bertanda positif akan mengurangi *cutsize* sedangkan *gain* bertanda negatif akan meningkatkan *cutsize*. Proses ini dilakukan sehingga verteks dengan *gain* terbesar akan dipindah posisi partisinya. Kemudian, verteks yang telah berpindah posisi partisinya tidak diikutsertakan kembali dalam perhitungan *gain* pada iterasi selanjutnya. Selain itu, perpindahan tersebut menyebabkan perubahan nilai *gain* pada verteks-verteks yang lain. Proses ini dilakukan terus berulang kali sedemikian sehingga tidak terdapat verteks yang berpindah posisi lagi. Dengan demikian, solusi optimal dari algoritma FM adalah partisi dengan *cutsize* terkecil.

Paralelisasi Matriks-Vektor

Terdapat dua model untuk mendekomposisikan matriks dengan menggunakan *hypergraph partitioning* yaitu model *column-net* dan *row-net.* Misalkan diberikan matriks sembarang berukuran $m \times n$ dan vektor $n \times 1$ sebagai berikut:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, \bar{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad (3)$$

Pada model *column-net*, matriks *A* direpresentasikan sebagai *hypergraph* $\mathcal{H}_R = (V_R, N_C)$dimana setiap baris matriks *A* diwakili oleh verteks-verteks di dalamV_R pada *hypergraph* dan setiap kolom matriks *A* diwakili oleh *net-net* di dalam N_C. Sedangkan, pada model *row-net*, matriks *A* direpresentasikan sebagai *hypergraph* $\mathcal{H}_C = (V_C, N_R)$dimana setiap baris matriks *A* diwakili oleh verteks-verteks di dalamV_C pada *hypergraph* dan setiap kolom matriks *A* diwakili oleh *net-net* di dalam N_R.Dengan demikian, baik model *column-net* maupun *row-net* terdapat satu verteks v_i dan satu *net*n_j untuk setiap baris ke-*i* dan kolom ke-*j*.

Matriks pada persamaan (3) selanjutnya dibentuk menjadi blok-blok matriks sebanyak $K \times K$ yang dapat dinyatakan pada persamaan (4) berikut ini.

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1K} \\ A_{21} & A_{22} & \cdots & A_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ A_{K1} & A_{K2} & \cdots & A_{KK} \end{bmatrix}, \qquad (4)$$

Dimana K merupakan banyaknya prosesor dengan setiapmatriks A_{ij}berukuran $\mathrm{m_i} \times \mathrm{n_j}$, sedemikian sehingga

$$\sum_{i=1}^{k} m_i = m \text{ dan } \sum_{j=1}^{k} n_j = n.$$

Sama halnya seperti matriks, vektor $\bar{x}$ pun juga dibagi menjadi beberapa sub-vektor yang bersesuaian sebagai berikut:

$$\bar{x} = \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \vdots \\ \bar{x}_K \end{bmatrix}, \qquad (5)$$

Dimana $\bar{x}_i$ terdiri dari $n_i$elemen $\left(\sum_{i=1}^{k} n_i = n\right)$.

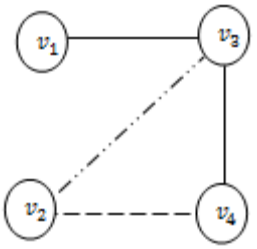
Pada teknik *hypergraph partitioning* ini, blok-blok matriks yang tidak terletak pada diagonal utama disusun sedemikan sehingga mayoritas elemen-elemen blok matriks tersebut banyak yang bernilai nol. Dengan demikian berarti bahwa komunikasi antar prosesor menjadi berkurang. Selanjutnya sub-matriks dan sub-vektor pada persamaan (4) dan (5) dikalikan secara paralel. Adapun algoritma yang digunakan dalam perkalian matriks-vektor adalah algoritma *row-based partitioning* [5, 7] dimana setiap prosesor mengerjakan masing-masing baris blok matriks pada persamaan (4). Tahap awal algoritma *row-based partitioning* ini menyatakan bahwa prosesor *i* yang menyimpan sub-vektor $\bar{x}_i$, mengerjakan blok baris ke-*i* dari matriks *A*, yaitu $[A_{i1}\ A_{i2}\ \cdots\ A_{ik}]$. Selanjutnya, $\bar{x}_i$ dikirim ke setiap prosesor$j \neq i$ dengan $A_{ji} \neq 0$. Selama pesan dari prosesor $i \neq j$ belum diterima oleh prosesor *j*, setiap prosesor menghitung $A_{jj}\bar{x}_j$. Adapun ketika $\bar{x}_i$ telah diterima oleh prosesor$j \neq i$, untuk elemenA_{ji}tidak nol, selanjutnya prosesor *j* menghitung $A_{ji}\bar{x}_i$. Pada proses pengiriman pesan tersebut diasumsikan bahwa masing-masing prosesor telah mengetahui pesan yang akan dikirim oleh prosesor lainnya. Dengan demikian, prosesor $j, 1 \leq j \leq k$, menghasilkan perkalian matriks-vektor yang terdiri dari n_j elemen, yaitu $\sum_{i=1}^{k} A_{ji}\bar{x}_i$.

Implementasi

Salah satu penerapan paralelisasi perhitungan perkalian matriks-vektor dengan mengimplementasikan *hypergraph partitioning* diberikan pada contoh berikut ini. Misalkan *A*adalah matriks berukuran 4×3 dan $\bar{x}$ adalah vektor berukuran 3×1 pada persamaan (6).

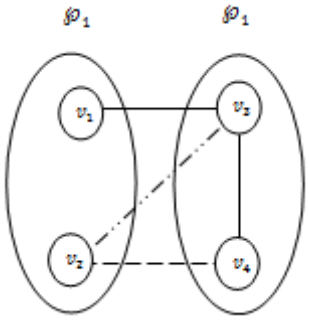
$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \ \bar{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}. \qquad (6)$$

Matriks *A* pada persamaan (6) dapat direpresentasikan sebagai *hypergraph*dengan model *column-net* yang memiliki 4 verteks yaitu $v_1, v_2, v_3,$ dan $v_4$serta 3 *net* yaitu,n_1, $n_2$dann_3 yang masing-masing menghubungkan$\{v_1, v_3, v_4\}$, $\{v_2, v_3\}$, dan $\{v_2, v_4\}$. Vertex-verteks padamasing-masing net disebut dengan *pins* sehingga untuk permasalahan tersebut terdapat 3 *pins* yaitu, $pins\ [n_1] = \{v_1, v_3, v_4\}$, $pins\ [n_2] = \{v_2, v_3\}$, dan $pins\ [n_3] = \{v_2, v_4\}$. Dengan demikian, permasalahan model *column-net* pada persamaan (6) dapat digambarkan sebagai *hypergraph* sebagai berikut:



Gambar 1. *Hypergraph*

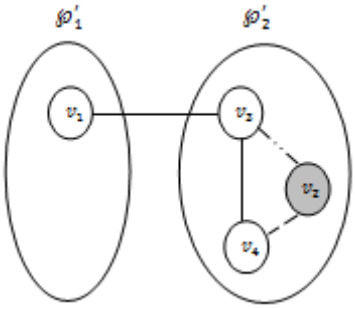
Seperti telah dijelaskan pada bab sebelumnya bahwa pada makalah ini algoritma FM dilakukan pada tahapan optimisasi dari teknik *hypergraph partitioning*. Oleh sebab itu, *hypergraph* pada Gambar 1 dipartisi menjadi dua bagian dimana *initial state* dari masing-masing partisi pertama ($\wp_1$) dan kedua ($\wp_2$) misalnya beranggotakan $\{v_1, v_2\}$ dan $\{v_3, v_4\}$, seperti ditunjukkan pada Gambar 2 berikut ini:



Gambar 2. Inisial Partisi

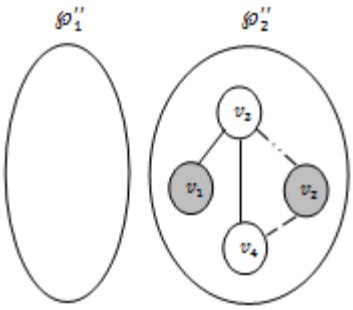
Terlihat pada Gambar 2 bahwa *cutsize* awal adalah sebesar 3. Dengan menggunakan *software* Matlab 7.01, diketahui bahwa v_2 memiliki *gain*

terbesar sehingga posisinya perlu dipindah dari sebelumnya pada partisi pertama menjadi partisi kedua pada iterasi pertama sedemikian sehingga $\wp_1' = \{v_1\}$ dan $\wp_2' = \{v_2, v_3, v_4\}$, seperti direpresentasikan pada Gambar 3. Dengan demikian, diperoleh *cutsize* baru, yaitu sebesar 1.



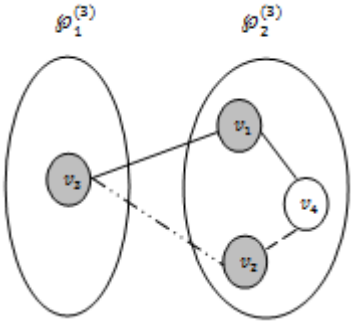
Gambar 3. Partisi Pertama

Algoritma FM kemudian dilanjutkan sedemikan sehingga diketahui bahwa v_1 memiliki *gain* terbesar sehingga posisinya perlu dipindah kembali pada iterasi kedua sedemikian sehingga $\wp_1'' = \{\ \}$ dan $\wp_2'' = \{v_1, v_2, v_3, v_4\}$ (lihat Gambar 4). Pada iterasi kedua ini *cutsize* yang dihasilkan adalahsebesar 0.



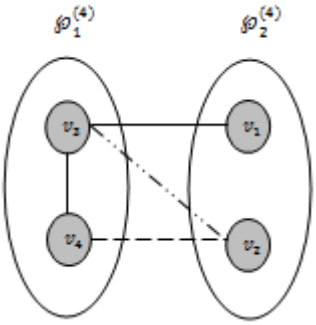
Gambar 4. Partisi Kedua

Adapun *gain* terbesar pada iterasi kedua ini ditempati oleh v_3 sehingga posisinya perlu dipindah pada iterasi ketiga, seperti ditunjukkan pada Gambar 5 dimana $\wp_1^{(3)} = \{v_3\}$ dan $\wp_2^{(3)} = \{v_1, v_2, v_4\}$. *Cutsize* pada iterasi ketiga ini bernilai 2.



Gambar 5. Partisi Ketiga

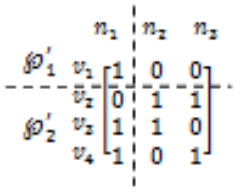
Terakhir, posisi v_4 kemudian dipindah juga pada iterasi keempat dimana $\wp_1^{(4)} = \{v_3, v_4\}$ dan $\wp_2^{(4)} = \{v_1, v_2\}$ sehingga diperoleh *cutsize* sebesar 3 seperti direpresentasikan pada Gambar 6.



Gambar 6. PartisiKeempat

Solusi optimum dari penerapan algoritma FM ini diperoleh pada iterasi pertama dengan *cutsize* sebesar 1. Selanjutnya, matriks *A* pada persamaan (1) juga dapat direpresentasikan sebagai *hypergraph* dengan model *row-net* yang memiliki 3 verteks yaitu $v_1', v_2', \text{dan} v_3'$serta 4 *net* yaitu,$n_1'$, n_2', n_3'dan n_4' yang masing-masing menghubungkan$\{v_1'\}$, $\{v_2', v_3'\}$, $\{v_1', v_2'\}$ dan$\{v_1', v_3'\}$ sehingga terdapat 4 *pins*y yaitu, $pins\ [n_1'] = \{v_1'\}$, $pins\ [n_2'] = \{v_2', v_3'\}$, $pins\ [n_3'] = \{v_1', v_2'\}$ dan $pins\ [n_4'] = \{v_1', v_3'\}$. Dengan cara yang samaseperti model *column-net*, maka partisi juga dilakukan untuk model *row-net* sedemikian sehingga diperoleh solusi optimum dimana $\wp_1 = \{v_1'\}$ dan $\wp_2 = \{v_2', v_3'\}$.

Dengandemikian, hasil yang diperoleh pada teknik *hypergraph partitioning* tersebut dapat direpresentasikan sebagai blok matriks seperti diberikan pada persamaan (7).



(7)

Selanjutnya untuk memudahkan pemahaman, maka masing-masing blok matriks pada persamaan (5) dinotasikan kembali ke dalam beberapa sub-matriks berikut ini:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

dimana

$$A_{11} = [1], A_{12} = [0 \quad 0],$$
$$A_{21} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \text{dan } A_{22} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Proses perhitungan perkalian matriks-vektor dapat dilakukan secara paralel menggunakan algoritma *row-based partitioning* dengan menerapkan hasil dari *hypergraph partitioning* tersebut. Pertama-tama masing-masing prosesor pertama dan kedua melakukan perkalian matriks A_{11} dengan x_1 dan A_{22} dengan $[x_2, x_3]^T$ secara paralel. Selanjutnya, prosesor pertama mengirimkan x_1 kepada prosesor kedua dan mengalikannya dengan matriks A_{21} untuk kemudian dijumlahkan dengan hasil perkalian yang telah dilakukan sebelumnya dalam prosesor kedua, begitu pun sebaliknya. x_i yang dikirimkan oleh prosesor-*i* kepada prosesor-*j* merupakan komunikasi yang terjadi dalam perkalian matriks-vektor. Dengan demikian, proses perhitungan perkalian matriks-vektor secara paralel dengan mengadopsi teknik *hypergraph partitioning* dapat tercapai.

Penutup

Berdasarkan hasil dan pembahasan pada makalah ini, dapat disimpulkan bahwa teknik *hypergraph partitioning* dapat digunakan dalam melakukan paralelisasi perkalian matriks ukuran sembarang dengan sebuah vektor dimana setiap prosesor hanya mengalikan baris-baris matriks yang terdapat pada hasil partisi dari teknik *hypergraph partitioning* saja. Paralelisasi dengan teknik *hypergraph partitioning* ini berhasil mengurangi komunikasi yang terjadi antar prosesor sehingga proses komputasi yang dilakukan menjadi lebih efisien. Kedua teknik *graph partitioning* dan *hypergraph partitioning* dalam paralelisasi perhitungan perkalian matriks-vektor yang telah dijelaskan pada makalah ini baru dapat diimplementasikan dengan model pemrograman multi-prosesor pada arsitektur *distributed memory*. Padahal, sejak intel menemukan teknologi *hyperthreading*, teknik paralelisasi cenderung beralih dari pemrograman multi-prosesor menjadi pemrograman *multi-thread* pada arsitektur *shared memory*. Pada pemrograman *multi-thread* ini, memori dapat diakses oleh beberapa prosesor secara bersamaan dimana setiap prosesor dapat mengeksekusi lebih dari satu rangkaian kode program yang independen selama periode waktu tertentu secara bersama-sama pula. Oleh sebab itu, implementasi kedua teknik *graph partitioning* dan *hypergraph partitioning* tersebut dalam memparalelkan perhitungan perkalian matriks-vektor dapat dikembangkan pada arsitektur *shared memory*.

Daftar Pustaka

[1] Catalyürek, U.V. dan Aykanat, C. (1996). Decomposing Irregularly Sparse Matrices for Parallel Matrix-Vector Multiplication. In *Parallel Algorithms for Irregularly Structured Problems*, Irregular '96, Number 1117 In Lecture Notes in Computer Science: 75-86. Berlin: Springer.

[2] Catalyürek, U.V. dan Aykanat, C. (1999). Hypergraph-Partitioning-Based Decomposition for Parallel Sparse-Matrix Vector Multiplication. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 10, No.7: 673-693.

[3] C.M. Fiduccia and R.M. Mattheyses. (1982). A Linier-Time Heuristic for Improving Network Partitions. *Proc. 19th ACM/EEE Design Automation* Conf.:175-181.

[4] Hendrickson, B. (1998). Graph Partitioning and Parallel Solvers: Has the Emperor No Clothes?. In *Solving Irregularly Structured Probem in Parallel*, Irregular '98, Number 1457 In Lecture Notes in Computer Science: 218-225. Berlin: Springer.

[5] Hendrickson, B. dan Kolda, T.G. (1998). Partitioning Sparse Rectangular Matrices for Parallel Computations of Ax and $A^T v$. In *Applied Parallel Computing in Large Scale Scientific and Industrial Problems*, PARA '98, Number 1541 In Lecture Notes in Computer Science: 239-247. Berlin: Springer.

[6] Hendrickson, B. dan Kolda, T.G. (2000). Graph Partitioning Models for Parallel Computing. *Parallel Computing*, Vol. 26, No. 12: 1519-1534.

[7] Hendrickson, B. dan Kolda, T.G. (2000). Partitioning Rectangular and Structurally Unsymmetric Sparse Matrices for Parallel Processing. *SIAM J.*, Vol. 21, No. 6: 2048-2072.

[8] Kernighan, B. W. dan Lin, S. (1969). An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell System Technical Journal*: 291-307.

[9] Murni, Handhika, T., Sari, I., dan Indarti, D. (2016). Implementasi Graph Partitioning pada Paralelisasi Perkalian Matriks-

Vektor.*ProsidingSeminar Nasional Matematika dan Statistika (SEMASTAT) Universitas Negeri Padang*: 194-199.