OPEN ACCESS
Indo-JC
Indonesian Journal of Computing
ISSN 2460-9056
socj.telkomuniversity.ac.id/indojc

IND. JOURNAL ON COMPUTING
VOL. 3, ISSUE. 1, MARCH 2018. PP. 97-102
DOI:10.21108/INDOJC.2018.3.1.218

Kinerja OpenMP pada Pengolahan Citra dengan Model Curvature Motion

Hasbi Rabbani [#1], P. H. Gunawan [*2]

[#] *School of Computing, Telkom University*
Jalan Telekomunikasi No. 1 Terusan Buah Batu, Bandung 40257, Indonesia.

[1] hasbirabb@student.telkomuniversity.ac.id

[2] phgunawan@telkomuniversity.ac.id

Abstract

The evolution of a geometric form can be generated by its curvature. The changing curvature is a displacement of its points and it is called as Mean Curvature Motion (MCM). MCM has been studied in depth to solve one of Jordan's curves in physical modeling. In this paper, the MCM solution is approximated by using a finite difference scheme and simulated into OpenMP parallel. To compute the parallel performance, 10 n_{iter} times simulations are elaborated using 2, 4, and 8 threads. From the numerical simulations, the results show that parallel performance is obtained has a lower computation time than the serial. In addition, the average efficiency of parallel code using 2 threads are observed higher than using 4 and 8 threads. For example on the size of n_{iter} 50000, the time speed of 2, 4, and 8 threads are 180.422 s, 156.002 s, and 333.243 s respectively. Moreover, the efficiency using 2, 4, and 8 threads are found 113%, 66%, and 34,8% respectively.

Keywords: Finite Difference Schemes, Curvature Motion Model, Parallel.

Abstrak

Evolusi dari sebuah bentuk geometri meliputi perubahan *curvature* yang terdapat dalam bentuk tersebut. Perubahan *curvature* ini tidak lepas dari perpindahan titik-titik pembentuknya dan diformulakan sebagai *Mean Curvature Motion* (MCM). MCM telah dipelajari secara mendalam untuk menyelesaikan salah satunya kurva Jordan pada pemodelan fisis. Pada jurnal ini, solusi MCM diaproksimasi menggunakan skema *finite difference* dan disimulasikan ke dalam paralel OpenMP. Untuk menghitung performansi paralel, dilakukan simulasi sebanyak 10 n_{iter} berbeda pada thread sejumlah 2, 4, dan 8. Dari simulasi yang telah dilakukan, didapatkan hasil bahwa performa paralel lebih membutuhkan waktu komputasi yang lebih rendah daripada serial. Selain itu, didapat pula rata-rata efisiensi kode paralel menggunakan 2 *Thread* lebih tinggi daripada menggunakan 4 *Thread* dan 8 *Thread*. Sebagai contoh pada ukuran n_{iter} 50000, kecepatan masing-masing 2, 4, dan 8 *Thread* adalah 180.422, 156.002, dan 333.243 s, serta efisiensi masing-masing 2, 4, dan 8 *Thread* adalah 113%, 66%, dan 34,8%.

Kata Kunci: Skema *Finite Difference*, Model *Curvature Motion*, Paralel.

I. PENDAHULUAN

MENURUT (Witkin [1]), semua teori *image multiscale* (gambar yang mempunyai skala yang banyak) menunjukkan bahwa semua ruang skala kausal, lokal, isometrik, dan kontras invarian didapat dari persamaan *curvature evolution* dari tipe

$$u_t = g(curv(u), t)|Du|, \tag{1}$$

RECEIVED ON FEB 2018. ACCEPTED ON MARCH 2018

dimana $u \in C^2$, dan g adalah fungsi bernilai riil yang tidak turun sehubungan dengan $curv(u)$ yang memenuhi kondisi batas $g(0,t) = 0$ dan $curv(u)$ menunjukkan *scalar curvature*, yang didefinisikan sebagai

$$curv(u) = \frac{1}{|Du|^3} D^2u(Du^\perp, Du^\perp) = \frac{u_{xx}u_y^2 - 2u_{xy}u_xu_y + u_{yy}u_x^2}{(u_x^2 + u_y^2)^{\frac{3}{2}}}. \quad (2)$$

Persamaan paling sederhana di kelas ini adalah *Mean Curvature Motion*:

$$u_t = curv(u)|Du|, \quad (3)$$

untuk itu eksistensi dan keunikan hasilnya diberikan secara independen oleh Evans dan Spruck [10] dan Chen, Giga dan Goto [6] dalam pengaturan teori solusi viskositas. Persamaan ini dihubungkan dengan kurva *shortening* dari *Jordan curves*

$$\frac{\partial x}{\partial t} = \kappa(x). \quad (4)$$

Untuk mempercepat waktu komputasi nya, teknik paralel akan diuraikan. Dalam referensi [4], [8], [9], [11] dan [5], komputasi paralel ditunjukkan sebagai ide bagus untuk menghitung perhitungan numerik. Dengan demikian tujuan jurnal ini adalah untuk mendapatkan kinerja komputasi paralel pada pengolahan citra dengan model *Curvature Motion*. Selain itu, arsitektur paralel menggunakan platform OpenMP dengan spesifikasi *processor* Intel Core i7-3770 @3.4GHz.

II. PENDEKATAN SOLUSI *MCM* MENGGUNAKAN *FDM*

Skema numerik mengambil keuntungan dari interpretasi difusi dari persamaan: jika $Du \neq 0$, skalar *curvature* dapat dinyatakan sebagai turunan kedua u dalam arah ξ ortogonal ke gradien, yaitu $u_t = u_{\xi\xi}$ [15]. Seperti yang disarankan oleh Alvarez dan Morel [3] bahwa turunan dievaluasi pada stensil 3x3, sebagai sebuah kombinasi linier dari nilai yang sesuai

$$\begin{aligned}(u_{\xi\xi})_{i,j} = \tfrac{1}{\Delta x^2}(-4\lambda_0 u_{i,j} + \lambda_1 \cdot (u_{i,j+1} + u_{i,j-1}) + \lambda_2 \cdot (u_{i+1,j} + u_{i-1,j}) \\ +\lambda_3 \cdot (u_{i-1,j-1} + u_{i+1,j+1}) + \lambda_4 \cdot (u_{i-1,j+1} + u_{i+1,j-1}))\end{aligned}$$

dan dengan demikian urutan iteratif didefinisikan oleh

$$u_{i,j}^{n+1} = u_{i,j}^n + \Delta t \cdot (u_{\xi\xi}^n)_{i,j}. \quad (5)$$

Sebagai gantinya, jika $|Du| = 0$, *Mean Curvature Motion* tidak terdefinisi dengan baik. Apabila gradien tersebut kecil, arahnya menjadi sangat acak yang disebabkan oleh kesalahan pembulatan dan penaksiran, sehingga tidak dapat dihindari bila berhadapan dengan skema diskrit. Jadi, ketika $|Du| < T_g$, maka menggunakan *half Laplacian*, yang merupakan ekspektasi dari $u_{\xi\xi}$ untuk distribusi uniform,

$$u_{i,j}^{n+1} = u_{i,j}^n + \frac{1}{2}\Delta t \cdot (\Delta u^n)_{i,j}, \quad (6)$$

dengan

$$(\Delta u^n)_{i,j} = u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}.$$

Rumus $(u_{\xi\xi})$ dapat ditulis ulang dalam bentuk yang lebih sintetis,

$$(u_{\xi\xi})^n = \tfrac{1}{\Delta x^2} \cdot (A \star u^n)\ ,\ A = \begin{pmatrix} \lambda_3(\theta) & \lambda_2(\theta) & \lambda_4(\theta) \\ \lambda_1(\theta) & -4\lambda_0(\theta) & \lambda_1(\theta) \\ \lambda_4(\theta) & \lambda_2(\theta) & \lambda_3(\theta) \end{pmatrix}$$

dimana $\star$ singkatan dari konvolusi diskrit dengan variabel kernel A, sedangkan θ adalah sudut antara gradien dan sumbu horizontal.

Berdasarkan referensi [15], bahwa untuk memperoleh konsistensi koefisien kombinasi linier harus memenuhi:

$$\begin{aligned}\lambda_1(\theta) &= 2\lambda_0(\theta) - cos^2\theta \\ \lambda_2(\theta) &= 2\lambda_0(\theta) - sin^2\theta \\ \lambda_3(\theta) &= -\lambda_0(\theta) + 0.5 \cdot (1 - sin^2\theta cos^2\theta) \\ \lambda_4(\theta) &= -\lambda_0(\theta) + 0.5 \cdot (1 + sin^2\theta cos^2\theta)\end{aligned} \tag{7}$$

dimana θ adalah arah antara gradien dan semi sumbu x positif. Perhatikan bahwa $sin\theta$ dan $cos\theta$ dapat dihitung dari u_x dan u_y , yang pada gilirannya akan dihitung secara numerik sebagai

$$(u_x)_{i,j} = \frac{2(u_{i+1,j} - u_{i-1,j}) + u_{i+1,j+1} - u_{i-1,j+1} + u_{i+1,j-1} - u_{i-1,j-1}}{8\Delta x} \tag{8}$$

$$(u_y)_{i,j} = \frac{2(u_{i,j+1} - u_{i,j-1}) + u_{i+1,j+1} - u_{i+1,j-1} + u_{i-1,j+1} - u_{i-1,j-1}}{8\Delta x} \tag{9}$$

Untuk MCM, kisaran nilai n_{iter} untuk simulasi numerik dihitung menggunakan rumus teoritis,

$$n_{iter} = \frac{3}{4\Delta t} R^{\frac{4}{3}}$$

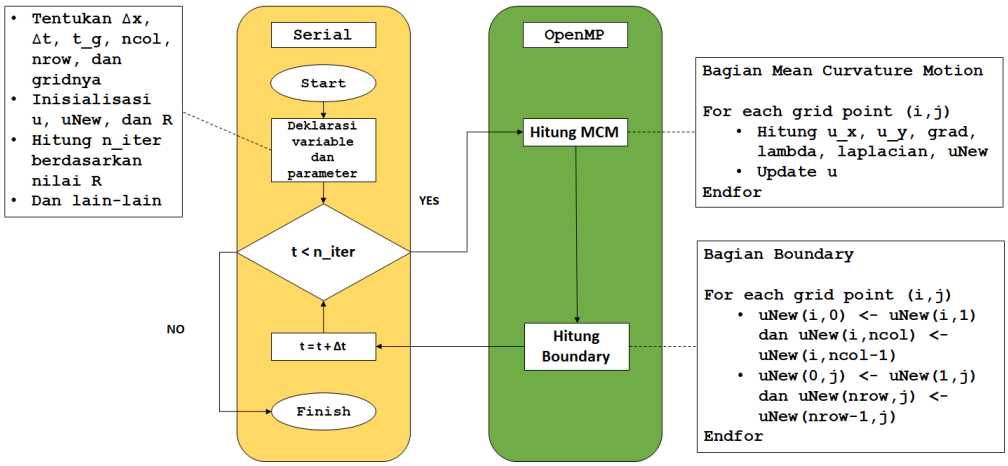
dengan R adalah skala yang dinormalisasi.

III. ARSITEKTUR PARALEL

Untuk menurunkan biaya waktu komputasi, dilakukan penjabaran masalah dengan menggunakan pemrosesan secara paralel. Karena setiap variabel yang digunakan independen, perhitungan fungsi dari masing-masing grid dapat dipisahkan dan dihitung pada prosesor yang berbeda. Pada tulisan ini, algoritma paralel akan diimplementasikan dengan menggunakan program OpenMP.

OpenMP adalah API yang menyediakan memori multithreading dan shared [2], [7], [13], [14], [16], [17]. Multithreading adalah kemampuan CPU untuk menjalankan beberapa proses dalam waktu yang bersamaan. Kemampuan ini dapat mengurangi proses runtime dan meningkatkan efisiensi algoritma. Selain itu, memori bersama adalah arsitektur paralel yang memungkinkan setiap proses berjalan untuk mendapatkan sumber memori secara proporsional. Dengan fitur ini, waktu akses ke penyimpanan bisa beminimum dan mempercepat proses runtime dapat ditingkatkan.

Dalam algoritma pengolahan citra dengan model *Curvature Motion*, paralelisasi dapat diimplementasikan pada bagian *Mean Curvature Motion* dan batas-batasnya untuk setiap grid. Lebih detail tentang algoritma pengolahan citra dengan model *Curvature Motion*, ditunjukkan pada Gambar 1.



Gambar 1: Konfigurasi algoritma paralel untuk mendekati solusi FDM untuk model *curvature motion* dalam mempercepat waktu komputasi.

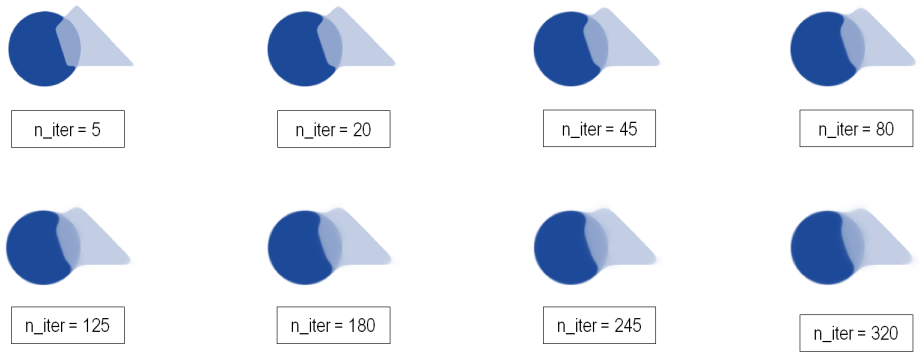
Dalam proses serial, semua parameter dinyatakan dan menyatakan kondisi matriks u sebelum di proses. Sementara dalam proses paralel, prosedur untuk menghitung dan memperbarui matriks u menggunakan parameter yang sudah di deklarasikan.

IV. HASIL NUMERIK DAN KINERJA PARALEL

Berikut adalah inisiasi awal untuk bentuk geometri yang akan diproses menggunakan FDM pada MCM dengan $\Delta t = 0.1$, *threshold* $T_g = 4$ dan R dalam *ranges* [1,8].

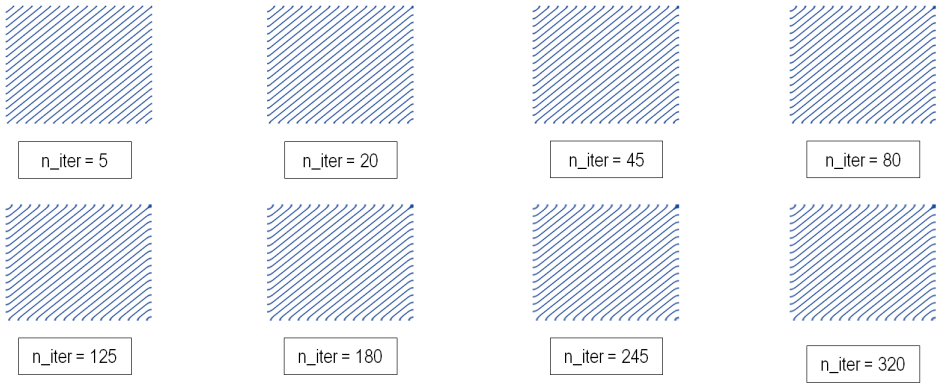
Gambar 2: Kiri, dengan ukuran matriks u 200×270; Kanan, dengan ukuran matriks u 450×450.

Hasil dari pengolahan citra dapat dilihat pada Gambar 3 dan 4. Untuk Gambar 3, pada saat $n_{iter} = 45$ belum mengalami perubahan yang cukup signifikan akan tetapi, perubahan kelengkungan mulai terlihat jelas pada saat $n_{iter} = 125$ sampai $n_{iter} = 320$ dengan SNR(*Signal-to-Noise Ratio*) 21.0518.



Gambar 3: (dari kiri ke kanan dan dari atas ke bawah): Evolusi MCM dari beberapa garis level

Sedangkan untuk Gambar 4, perubahan kelengkungan terjadi di sekitar tepi gambar pada saat $n_{iter} = 80$ sampai $n_{iter} = 320$ dengan SNR 15.7863.



Gambar 4: (dari kiri ke kanan dan dari atas ke bawah): Evolusi MCM dari beberapa garis level

Tulisan ini fokus pada kinerja paralel dari pengolahan citra sebelumnya. Hasil kinerja paralel dapat ditemukan pada Tabel I dan Tabel II. Di sini, tiga percobaan diberikan, dengan menggunakan 2, 4, dan 8 *Threads*. Sepuluh macam jumlah n_{iter} diberikan untuk mengevaluasi kinerja paralel. Dari Tabel I, platform OpenMP terbukti mampu mempercepat waktu komputasi untuk berbagai ukuran n_{iter}. Ketika digunakan nilai n_{iter} yang besar $n_{iter} = 50000$, waktu CPU dari kode serial diperoleh 409.380 s. Sedangkan waktu CPU dalam kode paralel menggunakan 2, 4, dan 8 *Threads* masing-masing teramati 180.422, 156.002, dan 93.302.

n_iterasi	Execution Time(s)			
	Serial	2 Threads	4 Threads	8 Threads
500	49.781	20.991	17.043	10.155
2000	102.671	43.275	22.872	20.918
4500	155.026	65.347	40.944	31.542
8000	204.490	86.235	45.822	41.966
12500	250.509	105.685	56.771	52.145
18000	292.458	123.952	67.051	61.764
24500	330.815	139.998	77.814	70.859
32000	365.567	155.080	86.817	79.253
40500	392.789	166.926	91.819	85.841
50000	409.380	180.422	156.002	93.302

Tabel I: Waktu eksekusi kinerja paralel algoritma FDM untuk gerak *curvature* menggunakan OpenMP.

Dari Tabel II, menghasilkan *speedup* 2.269, 2.624, dan 4.388 kali dengan menggunakan 2, 4, dan 8 *Threads* untuk $n_{iter} = 50000$. Selain itu, efisiensi menggunakan 2 *Thread* diperoleh 113%, 4 *Thread* 66% dan menggunakan 8 *Thread* dihitung 55%. Pada saat menggunakan 8 *Threads*, kinerja dari paralel mengalami *sublinear speedup* dimana *speedup* lebih kecil dari prosesor yang digunakan. Salah satu penyebab nya adalah batas *speedup* dari program paralel yang bergantung pada proporsi program yang dapat diparalelkan [12].

Speedup			Efficiency(%)		
2 Threads	4 Threads	8 Threads	2 Threads	4 Threads	8 Threads
2.372	2.921	4.902	119	73	61
2.373	4.489	4.908	119	112	61
2.372	3.786	4.915	119	95	61
2.371	4.463	4.873	119	112	61
2.370	4.413	4.804	119	110	60
2.359	4.362	4.735	118	109	59
2.363	4.251	4.669	118	106	58
2.357	4.211	4.613	118	105	58
2.353	4.278	4.576	118	107	57
2.269	2.624	4.388	113	66	55

Tabel II: Kecepatan dan efisiensi kinerja paralel algoritma FDM untuk gerak *curvature* menggunakan OpenMP.

Pada tulisan ini, dengan percobaan menggunakan 2, 4, dan 8 *Threads*, *speedup* untuk semua simulasi dengan menggunakan 8 *Thread* diamati mendapatkan *speedup* terbaik dibandingkan dengan menggunakan 2 *Thread* dan 4 *Thread*. Namun, berbeda dengan *speedup*, efisiensi menggunakan 2 *Thread* diperoleh lebih baik daripada menggunakan 4 *Thread* dan 8 *Thread*.

V. KESIMPULAN

Berdasarkan hasil yang telah didapatkan, performa paralel pada pengolahan citra dengan model *Curvature Motion* membutuhkan waktu komputasi yang lebih rendah daripada algoritma serial. Ini berarti algoritma paralel berhasil diterapkan dalam masalah ini. Selanjutnya, kode paralel dengan 2 *Thread* memiliki kecepatan rata-rata yang lebih rendah daripada menggunakan 4 *Thread* dan 8 *Thread*. Namun, rata-rata efisiensi kode paralel menggunakan 2 *Thread* lebih tinggi daripada menggunakan 4 *Thread* dan 8 *Thread*. Misalnya pada ukuran n_{iter} 50000, kecepatan masing-masing 2, 4, dan 8 *Thread* adalah 180.422, 156.002, dan 333.243 s, serta efisiensi masing-masing 2, 4, dan 8 *Thread* adalah 113%, 66%, dan 34,8%.

Pustaka

[1] *IJCAI'95: Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[2] M N A Alamsyah, C A Simanjuntak, B A R H Bagustara, W A Pradana, and P H Gunawan. Openmp analysis for lid driven cavity simulation using lattice boltzmann method. In *Information and Communication Technology (ICoIC7), 2017 5th International Conference on*, pages 1–6. IEEE, 2017.

[3] Luis Alvarez and Jean Michel Morel. Formalization and computational aspects of image analysis. *Acta numerica*, 3:1–59, 1994.

[4] André R Brodtkorb, Martin L Sætra, and Mustafa Altinakar. Efficient shallow water simulations on gpus: Implementation, visualization, verification, and validation. *Computers & Fluids*, 55:1–12, 2012.

[5] D Castillo, AM Ferreiro, José A García-Rodríguez, and Carlos Vázquez. Numerical methods to solve pde models for pricing business companies in different regimes and implementation in gpus. *Applied Mathematics and Computation*, 219(24):11233–11257, 2013.

[6] Yun Gang Chen, Yoshikazu Giga, Shun'ichi Goto, et al. Uniqueness and existence of viscosity solutions of generalized mean curvature flow equations. *Journal of differential geometry*, 33(3):749–786, 1991.

[7] Leonardo Dagum and Ramesh Menon. Openmp: an industry standard api for shared-memory programming. *IEEE computational science and engineering*, 5(1):46–55, 1998.

[8] M de la Asunción, MJ Castro, JM Mantas, and S Ortega. Numerical simulation of tsunamis generated by landslides on multiple gpus. *Advances in Engineering Software*, 99:59–72, 2016.

[9] Marc De La Asunción, José M Mantas, and Manuel J Castro. Simulation of one-layer shallow water systems on multicore and cuda architectures. *The Journal of Supercomputing*, 58(2):206–214, 2011.

[10] L. C. Evans and J. Spruck. Motion of level sets by mean curvature. i. *J. Differential Geom.*, 33(3):635–681, 1991.

[11] Putu Harry Gunawan. Scientific parallel computing for 1d heat diffusion problem based on openmp. In *Information and Communication Technology (ICoICT), 2016 4th International Conference on*, pages 1–5. IEEE, 2016.

[12] Mark D Hill and Michael R Marty. Amdahl's law in the multicore era. *Computer*, 41(7), 2008.

[13] Iryanto and P H Gunawan. An openmp parallel godunov scheme for 1d two phase oil displacement problem. In *Information and Communication Technology (ICoIC7), 2017 5th International Conference on*, pages 1–5. IEEE, 2017.

[14] S Juliati and P H Gunawan. Openmp architecture to simulate 2d water oscillation on paraboloid. In *Information and Communication Technology (ICoIC7), 2017 5th International Conference on*, pages 1–5. IEEE, 2017.

[15] M Mondelli and A Ciomaga. On finite difference schemes for curvature motions. In *International Student Conference on Pure and Applied Mathematics*, page 137, 2011.

[16] Mulyani, Novella D Putri, and P H Gunawan. The performance of openmp architecture for simulating fire spreading in forest area by cellular automata. In *Information and Communication Technology (ICoIC7), 2017 5th International Conference on*, pages 1–5. IEEE, 2017.

[17] M R Pahlevi and P H Gunawan. Parallel processing for simulating 2d radial dambreak using fvm hlle flux on openmp. In *Information and Communication Technology (ICoIC7), 2017 5th International Conference on*, pages 1–4. IEEE, 2017.