Jurnal Sains dan Pendidikan Fisika (JSPF)
Jilid 16, No 03, Desember 2020
Hal. 259 - 269

pISSN :1858 330X
eISSN : 2548-6373
Laman Webiste : http://ojs.unm.ac.id/jsdpf

FUNGSIONALITAS PEMROGRAMAN BERBASIS CPU-GPU DALAM SIMULASI PENJALARAN TSUNAMI

***Dewi Anggraeni**
Universitas Brawijaya
dewianggraeni.x@ub.ac.id

*koresponden author

Abstrak - Indonesia merupakan daerah rawan gempa yang memicu timbulnya tsunami. Salah satu cara menganalisis gelombang tsunami adalah dengan membangun sistem simulasi penjalarannya. Penjalaran gelombang tsunami dapat dimodelkan secara matematis dengan *Shallow Water Equation* (SWE) sebagai gelombang *non-dispersive*, yang berarti bahwa gelombang dengan panjang gelombang berbeda menjalar dengan kecepatan yang sama. Pengembangan GPU (*Graphic Processing Unit*) dalam aplikasi numerik menjadi salah satu solusi dalam penyelesaian persamaan penjalaran tsunami tersebut. *Library* CUDA dalam GPU dapat digunakan sebagai komputasi parallel sehingga mempercepat proses perhitungan. Dalam penelitian ini berhasil dibangun program untuk memodelkan penjalaran gelombang tsunami dengan bahasa pemrograman C berbasis CPU-GPU, dengan penyelesaian yang fungsional dan 9 kali lebih cepat dari pemrograman berbasis CPU.

Kata Kunci : Tsunami, CPU, GPU

Abstract *– Indonesia is one of the most tsunami prone regions in the world. One way to analyse tsunami waves is to build a simulation system for their propagation. The propagation of a tsunami wave can be mathematically modelled using the Shallow Water Equation (SWE) as a non-dispersive wave, which means that waves of different wavelengths travel at the same speed. The development of GPU (Graphic Processing Unit) in numerical applications is one solution in solving the tsunami propagation equation. The CUDA library on the GPU can be used for parallel computing thereby speeding up the calculation process. In this research, a program to model tsunami wave propagation was successfully built using the CPU-GPU-based C programming language, with a functional solution and 9 times faster than CPU-based programming.*

Keywords *: Tsunami, CPU, GPU.*

A. PENDAHULUAN

Indonesia adalah daerah rawan gempa, yang memicu timbulnya tsunami. Tsunami adalah gelombang laut yang terjadi akibat gempa, longsoran, letusan gunung api, yang terjadi di dasar laut, yang ditimbulkan oleh gaya impulsif akibat deformasi bawah laut, karena itulah, diperlukan sistem simulasi penjalaran tsunami yang memadai akan hal itu. Salah satu cara menganalisis gelombang tsunami ini adalah dengan membangun sistem simulasi penjalarannya pada area laut.

Persamaan penjalaran gelombang tsunami sendiri telah banyak dikembangkan, salah satunya dengan pendekatan tsunami sebagai gelombang *non-dispersif,* yaitu *Shallow Water Equation* (SWE) (Erwina et al., 2020)(Supian & Rusli, 2018). Hal ini berarti bahwa gelombang dengan panjang gelombang berbeda diasumsikan merambat dengan kecepatan yang sama. Penyelesaian persamaan SWE tersebut lebih mudah diselesaikan secara numerik, dan salah satu keuntungannya, hasil numerik tersebut dapat divisualisasikan, sehingga sangat tepat jika dimanfaatkan sebagai *Tsunami Warning System*. Karena dengan demikian, kita dapat mengetahui daerah yang diperkirakan rawan dan parah.

Fenomena fisis penjalaran gelombang tsunami dapat dimodelkan secara matematis menggunakan persamaan diferensial (*Partial Differential Equation atau* PDE). Salah satu model persamaannya adalah *Shallow Water Equation (*SWE) (Erwina et al., 2020). Persamaan ini menggambarkan model penjalaran gelombang tsunami dengan pendekatan sebagai gelombang *non-dispersif*. Penjalaran gelombang *non-dispersif* berarti bahwa gelombang dengan panjang gelombang berbeda menjalar dengan kecepatan rambat yang sama. SWE yang menggambarkan persamaan penjalaran tsunami di daerah perairan adalah sebagai berikut :

$$\partial_t \eta(\Omega,t) = -\nabla\ [h(\Omega)\nabla\ \varphi(\Omega,t)] \tag{1}$$

$$\partial_t \varphi(\Omega,t) = -g\,\eta(\Omega,t) \tag{2}$$

dimana:

Ω adalah posisi tiap titik di dalam domain yang berada pada koordinat (x,y) (meter)

∇ adalah operator gradien dari sebuah fungsi (pada kasus ini 2 dimensi) :

$$\nabla = \frac{\partial}{\partial} + \frac{\partial}{\partial}$$

$\eta(\Omega, t)$ = ketinggian muka air laut pada waktu t dan posisi Ω (meter)

$\varphi(\Omega,t)$ = potensial kecepatan rambatan gelombang pada waktu t dan posisi Ω, yang berarti bahwa gradien dari potensial kecepatan tersebut, menunjukkan kecepatan gelombang pada posisi Ω (meter/detik)

$h(\Omega)$ = kedalaman laut pada domain Ω, diasumsikan tidak terjadi perubahan sepanjang simulasi (meter)

g = konstanta percepatan gravitasi (m/s^2)

Pada persamaan tersebut, dapat dilihat bahwa persamaan SWE merupakan persamaan penjalaran gelombang *Spatial-Time Dependent Partial Differential Equation (PDE),* persamaan diferensial yang merupakan fungsi dari domain spasial Ω dan waktu t. Dengan penerapan metode elemen hingga pada persamaan SWE tersebut, maka persamaan SWE dapat direduksi menjadi sebuah sistem persamaan dalam bentuk matrik yang hanya merupakan fungsi terhadap waktu (*Time Dependent Ordinary Defferential Equation* (ODE), seperti berikut ini :

$$M\partial_t \overline{\eta} = S_h \overline{\varphi} + B\overline{\eta} \tag{3}$$

$$M\partial_t \overline{\varphi} = -gM\overline{\eta} \tag{4}$$

Untuk selanjutnya, sistem persamaan tersebut disebut sebagai sistem matrik reduksi.

Matrik *M*, *Sh* dan *B* merupakan hasil dari penerapan metode hingga pada persamaan SWE untuk mereduksi kebergantungannya terhadap domain spasial. Nilai dari matrik *M* (*Mass Matrix*), *Sh* (*Stiffness Matrix*), dan *B* (*Boundary Matrix*) pada sistem matrik reduksi, merupakan operasi matematis yang berkaitan dengan data elemen dan data kedalaman laut (*bathymetry*) pada domain spasial yang digunakan, pada penelitian ini merupakan data riil dari pangandaran pada koordinat lokasi 105^0 BT – 110^0 BT dan 11^0 LS – 7^0 LS.

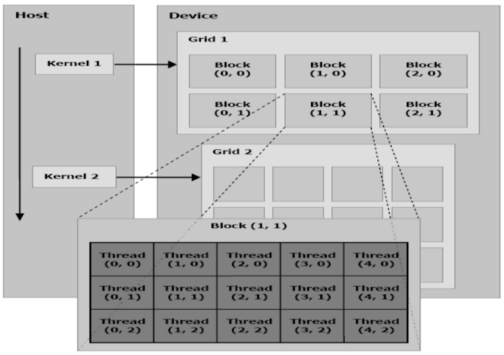
Pengembangan GPU (*Graphic Processing Unit*) dalam aplikasi numerik, diharapkan menjadi salah satu solusi dalam kasus penyelesaian persamaan penjalaran tsunami tersebut. GPU adalah salah satu *hardware* komputer dengan memori dan *processor* internal yang instruksi maupun akses datanya dapat berjalan secara parallel (Vázquez et al., 2013). Komputasi numerik di dalam GPU dapat dilakukan dengan memanfaatkan *library* CUDA (*Compute Unified Device Architecture*) (Ueng et al., 2008). Bahasa pemrograman C merupakan bahasa pemrograman CPU (C*entral Processing Unit*) yang dapat diterapkan (*compatible*) dengan GPU, sehingga hal ini dapat menjadi langkah awal dalam memanfaatkan GPU untuk simulasi penjalaran tsunami. Dengan demikian, langkah membangun sistem matrik SWE berbasis CPU-GPU menjadi hal yang penting untuk dikembangkan. Karena itulah penelitian ini ditekankan pada pembangkitan sistem matrik persamaan SWE, sebagai hasil reduksi dengan menggunakan metode elemen hingga, dengan menggunakan bahasa pemrograman C berbasis pada CPU - GPU.

CUDA (*Compute Unified Device Architecture*) merupakan *library* keluaran NVIDIA untuk komputasi paralel pada GPU (Ueng et al., 2008). Dengan adanya CUDA pengembang *software* bisa menggunakan kemampuan GPU untuk mempercepat proses komputasi. CUDA didesain khusus hanya untuk kartu grafik (GPU) keluaran NVIDIA. CUDA menyembunyikan detail dari perangkat keras GPU di bawah API (*Aplication programming Interface*- antar-muka aplikasi pemrograman). Penyembunyian detail perangkat keras mempunyai dua keuntungan. Pertama, menyederhanakan model perograman tingkat tinggi yang mengisolasi programer dari detail perangkat keras GPU. Kedua, kompatibilitas perangkat lunak, artinya perangkat lunak yang telah dibuat bisa berjalan pada berbagai macam artisetur

GPU atau hanya mengalami perubahan sedikit untuk alasan optimasi jika pengembang perangkat keras (NVIDIA) merubah / mengembangkan arsitektur GPU-nya (Boyer & El Baz, 2013).

Dalam konteks CUDA, GPU disebut *device* sedangkan CPU disebut *host.* Device hanya bisa mengakses memorinya sendiri. Sebuah fungsi yang dieksekusi di GPU disebut *fungsi kernel. Kernel* dieksekusi dengan model SPMD (*Single Program Multiple Data*), artinya programer menentukan jumlah *thread* yang akan meneksekusi fungsi kernel (Breitbart, 2008).

Thread yang mengeksekusi *kernel* dikelompokkan sebagai sebuah grid dari blok-blok *thread* seperti yang ditunjukkan pada **Gambar 1**. Blok *thread* merupakan kumpulan *thread*. *Thread* dalam satu blok dapat berbagi data melalui memori yang disebut memori bersama (*shared memory*). Masing-masing *thread* dalam satu blok diidentifikasi melalui indeks *thread*-nya yang merupakan nomor *thread* dalam blok tersebut. Untuk membantu dalam pengelamatan komplek yang menggunakan indeks *thread*, sebuah aplikasi dapat menspesifikasikan sebuah blok dalam array 2 atau 3 dimensi dengan ukuran yang sembarang. Untuk blok berukuran 2 dimensi dengan besar (D_x, D_y) ,id *thread* dari *thread* dengan index(x,y) adalah ($x + yD_x$) dan untuk blok berukuran 3 dimensi dengan ukuran dimensi (D_x, D_y, D_z), id *thread* dengan indeks (x, y, z) adalah ($x + yD_x + zD_xD_y$) (Breitbart, 2008).



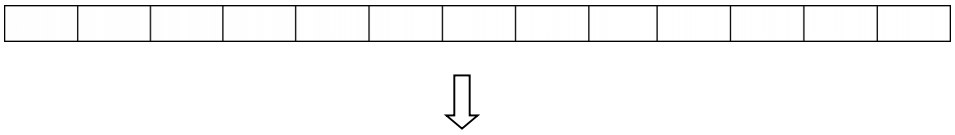
Gambar 1 Model *Software* CUDA (Breitbart, 2008)

Sebuah grid dieksekusi di dalam GPU dengan menjadwalkan blok *thread* ke dalam multiprosesor. Jadi, tiap-tiap blok dipetakan ke satu buah multiprosesor. Multiple blok *thread* dapat dipetakan pada multiprosesor yang sama dan kemudian dieksekusi secara bersamaan. Jika multiple blok *thread* dieksekusi pada satu multiprosesor yang sama, maka sumberdaya-nya, seperti register dan shared memory akan dibagi ke semua blok *thread*. Ini membatasi jumlah blok yang bisa dipetakan pada multiprosesor yang sama. Sebuah blok akan tetap tinggal di dalam multiprosesor sampai eksekusi kernel-nya selesai (Breitbart, 2008).

B. METODE

Pada penelitian ini dilakukan pembuatan program untuk menghitung nilai data matrik reduksi, dari persamaan penjalaran gelombang tsunami (*Shallow Water Equation*) dengan metode elemen hingga, menggunakan bahasa pemrograman C berbasis CPU-GPU.

Analisis data hasil penelitian ini dilakukan dalam beberapa tahap, diantaranya kalibrasi data, perhitungan waktu eksekusi program dan penyelesaian dengan ODESOLVER Matlab 7.4. Kalibrasi data



Gambar 2. Perubahan Indek Array 1 Dimensi Menjadi Matrik

dilakukan untuk menghitng selisih data dari hasil penelitian yang diperoleh melalui bahasa pemrograman C dan CUDA berbasis CPU-GPU dibandingkan dengan pemrograman menggunakan Matlab berbasis CPU. Karena hasil penelitian berupa 3 kumpulan data matrik, yaitu matrik M, Sh dan B, dalam bentuk array 1 dimensi berukuran N x N (4089 x 4089), dimana N adalah banyaknya titik-titik data, maka perlu diubah dahulu indek data supaya tepat dapat dimasukan dalam indek matrik di dalam Matlab. Hal itu dapat dilakukan dengan beberapa langkah dalam operasi program C terhadap data yang bernilai tidak nol. Program menghitung hasil bagi indek data array 1 dimensi dengan N, dituliskan untuk mengetahui baris indeknya di dalam matrik, dan program menghitung sisa hasil baginya, untuk mengetahui indek kolom matrik. Langkah tersebut digambarkan pada **Gambar 2**.

Perhitungan waktu dalam membangun matrik dalam bahasa C berbasis CPU-GPU dengan Matlab 7.4 (berbasis CPU) dapat diketahui dengan menuliskan perintah pada program untuk mengaktifkan *timer* penghitung waktu selama program membangun matrik berjalan, kemudian menghentikan *timer* setelah selesai membangun ketiga matrik.

Analisis data juga dilakukan dengan cara menyelesaikan *Timing Dependent* ODE dari sistem data matrik M, Sh dan B yang dibangun, menggunakan ODESOLVER Matlab 7.4. Hasil dari analisis data ini, ditujukan untuk memastikan bahwa program yang dibuat dengan bahasa C berbasis CPU-GPU dalam membangun matrik reduksi SWE dengan metode elemen hingga.

C. HASIL DAN PEMBAHASAN

Data elemen yang digunakan dalam penelitian ini berupa nomor indek segitiga (elemen), nomor global titik, dan juga posisi setiap titik, yang akan dicari solusi persamaan SWE-nya. Jika jumlah titik data sebanyak N, maka ukuran ketiga matrik (masing-masing) sebesar N x N. Dalam penelitian ini digunakan sebanyak 4089 titik data pada domain spasial, sehingga masing-masing matrik yang dibangun berukuran 4089 x 4089 = 16.719.921. Ketiga matrik tersebut termasuk matrik *sparse,* yaitu matrik yang sebagian besar berisi nol, kecuali nilai pada indek matrik tertentu yang tidak nol, sebagai

data hasil dari penerapan metode elemen hingga (akan dijelaskan selanjutnya untuk data masing-masing matrik).

Operasi matematis pada data penelitian dilakukan untuk mencari indek matrik yang bernilai tidak nol dan mengisinya dengan nilai sesuai persamaan (3) dan (4). Dengan demikian, perhitungan nilai matrik yang bernilai tidak nol dapat dilakukan dari operasi data tiap elemen segitiga, yang meliputi nomor global dan lokal tiap titik yang membentunya dan juga kedalaman laut tiap titik tersebut. Dalam operasi numerik di CPU saja, operasi tiap elemen ini hanya dapat dilakukan secara serial. Namun, dengan operasi di CPU-GPU, operasi data elemen tersebut dapat dilakukan secara semi paralel. Dikatakan semi paralel karena penggunaan operasi numerik pada penelitian ini masih menggunakan CPU-GPU, yang dilakukan dengan bahasa pemrograman C dan menggunakan *library* CUDA, tidak secara utuh di dalam GPU saja. Karena itulah untuk selanjutnya, yang dimaksud operasi di CPU-GPU adalah penggunaan bahasa pemrograman C dengan menggunakan *library* CUDA, begitu pula sebaliknya.

Dengan penerapan elemen hingga, solusi persamaan disetiap titik didekati dengan kombinasi fungsi basis lokal. Fungsi basis lokal akan bernilai 1 pada titik yang dicari solusinya, dan bernilai nol pada titik lain. Dengan demikian untuk menghitung matrik M, Sh dan juga matrik B, perlu kita perhatikan dahulu diskripsi awal ketiga matrik tersebut. Matrik M didefinisikan sebagai berikut (Kerkeni et al., 2016):

$$\mathrm{M} = \left(m_{ij}\right) = \int_{\Omega} T_{\Omega_i} T_{\Omega_j} \, d\Omega \tag{5}$$

Matrik M merupakan integral setiap fungsi basis dikali dengan fungsi basis lain ($i \neq j$)dan juga dirinya sendiri (i = j). Untuk solusi $\eta_1$misalanya, maka kita harus mencari fungsi basis lokal yang tumpang tindih dengan fungsi basis lokal pada titik 1 di dalam domain segitiga. Dengan demikian nilai matrik M untuk solusi η_1, merupakan resultan dari integral fungsi basis lokal pada titik 1 terhadap dirinya sendiri, bernilai 1/6 luas elemen, dan juga integral dari fungsi basis lokal titik 1 terhadap fungsi basis lokal elemen segitiga disekitarnya(bernilai 1/12 luas elemen sgitiga). Oleh karena itu, untuk menghitung indek matrik M_{ij} yang berisi tidak nol, maka dicari terlebih dahulu nomer global dari setiap titik (sebagai indek i) dan juga setiap titik yang berdekatan (sebagai indek j), kemudian diisi dengan nilai1/6 x luas elemen segitiga (untuk i = j) dan 1/12 x luas elemen segitiga (untuk $i \neq j$). Dimana luas setiap elemen adalah sebesar:

$$A = {}^{1}/_{2} \left[(x_2 - x_1)(y_2 - y_3) - (y_2 - y_1)(x_3 - x_2)\right] \tag{6}$$

Data setiap elemen tidak tergantung dari data elemen lain, karena itulah operasi elemen dapat dilakukan secara paralel di dalam GPU. GPU merupakan *hardware multiprocessor* yang dapat bekerja secara paralel terhadap banyak data. GPU akan membagi data ke dalam sejumlah blok dan akan memebrikan indek *thread* pada setiap data di dalam blok. Dengan menggunakan *library* CUDA, data

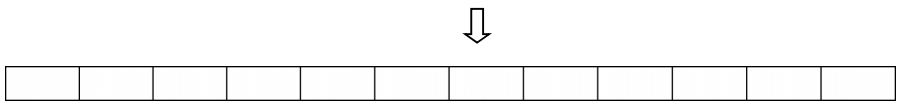
domain segitiga tersebut dibagi dalam 16 blok, masing-masing blok mengolah 256 *thread* data elemen segitiga.

Setiap data di dalam blok dapat dikenali dari indek x dan indek y pada *thread* datanya. Dengan demikian operasi *thread* untuk semua blok dapat dilakukan secara paralel. Karena setiap elemen dibentuk oleh 3 buah titik, yang dapat dikenali dengan nomer lokal titik i, j, dan titik k, maka operasi setiap elemen menghasilkan 3^2 anggota matrik M (untuk indek matrik M (baris,kolom) diantaranya (i,i), (i,j), (i,k), (j,i), (j,j), (j,k), (k,i), (k,j), dan (k,k)) yang bernilai tidak nol. Hal ini dapat dilihat pada cuplikan program dalam **Gambar 3** berikut :

```
#program operasi thread untuk indek matrik
#define blocksize 16
//parameter global untuk aplikasi di GPU
__global__ void input_data (…)
{       int N=4089;
for(int k=0; k<blockDim.x; k++){
        int i = (blockIdx.x * blockDim.x+ k);
        int data_t1=g_tt1[i];                   %titik i
        int data_t2=g_tt2[i];                    %titik j
        int data_t3=g_tt3[i];                   %titik k
        int index1= (data_t1-1)*N + data_t2;    %(i,j)
```

Gambar 3. Listing operasi *thread* data secara paralel

Dari program tersebut, dapat dilihat bahwa dalam bahasa pemrograman CUDA, indek data matrik yang seharusnya berupa baris dan kolom (2 dimensi) harus diubah untuk dapat dikenali sebagai data dalam array 1 dimensi. Hal ini dapat dilakukan sesuai program tersebut dan dapat dilihat pada **Gambar 4**.



Gambar 4. Perubahan Susunan Matrik Menjadi Array 1 Dimensi

Indek data matrik dalam array 1 dimensi ini, kemudian disebut sebagai indek koreksi matrik. Data indek koreksi matrik yang bernilai tidak nol tersebut disalin ke CPU, dan proses pengisian nilai matrik pada indek tersebut masih dilakukan secara serial dengan menggunakan bahasa pemrograman C.

Hal yang sama juga dilakukan dalam menghitung matrik Sh. Diskripsi matrik Sh adalah sebagai berikut (Kerkeni et al., 2016):

$$S_h = \left(S_{hij}\right) = \int_{\Omega} [h(\Omega)] \nabla T_{\Omega_i} . \nabla T_{\Omega_j} \, d\Omega \tag{7}$$

Matrik Sh mengandung integral dari gradien fungsi basis lokal (∇T_{Ω_i}) dikali gradien fungsi basis lokal lain ($i \neq j$) ataupun dikali gradien dirinya sendiri ($i = j$). dengan demikian maka indek matrik Sh yang bernilai tidak nol, sama dengan indek matrik M yang tidak nol. Namun, untuk ($i \neq j$) maupun ($i = j$), memiliki nilai yang sama.

Berbeda dengan perhitungan matrik M dan Sh, matrik B yang menggambarkan koreksi terhadap kondisi batas (*Boundary Condition*), dilakukan secara utuh dengan cara serial. Matrik B hanya bernilai tidak nol untuk koreksi solusi persamaan pada titik didaerah batas, sehingga jumlah anggota matrik B yang bernilai tidak nol lebih sedikit dari pada matrik M maupun matrik Sh. Karena hanya titik-titik pada batas yang memiliki nilai koreksi matrik B, maka perlu dibangun program untuk mengenali posisi titik yang berada pada batas. Hal ini dilakukan dengan mencari apakah titik berada pada posisi minimum (x atau y) atau maksimum (x atau y) dari domain keseluruhan,

Dalam penelitian ini didapatkan bahwa untuk menyusun data ketiga matrik tersebut dibutuhkan waktu 486.75 ms, sedangkan operasi di dalam CPU saja membutuhkan waktu 4220.98 ms. Hal ini berarti bahwa, dalam membangun matrik reduksi SWE dengan metode elemen hingga, menggunakan bahasa pemrograman C berbasis pada CPU-GPU, dapat 9 kali lebih cepat dari pada menggunakan Matlab 7.4 (operasi di dalam CPU saja).

Untuk menganalisis nilai matrik yang dibangunkan dalam penelitian ini, maka dilakukan kalibrasi terhadap data referensi yang telah dibangun di dalam CPU (*software* Matlab 7.4). Hasil kalibrasi ditunjukkan pada Tabel 1 dan Tabel 2 berikut.

Table 1. Data Matrik Referensi

Data	Ukuran	Total yang Tidak Nol
Matrik M	16.719.921	28039
Matrik Sh	16.719.921	28039
Matrik B	16.719.921	340

Table 2. Data Hasil Kalibrasi

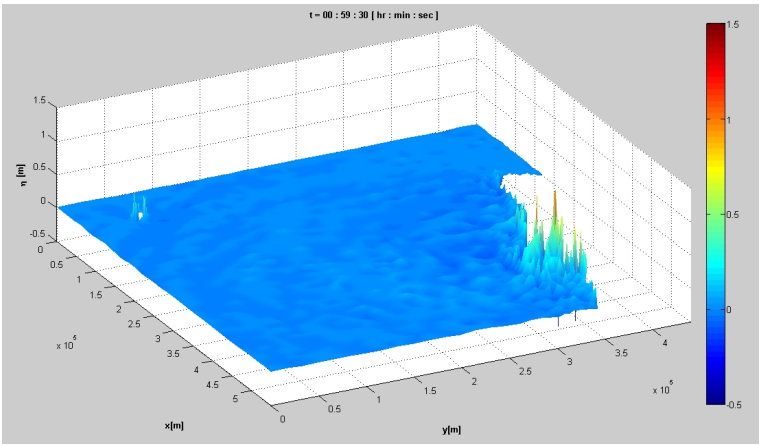
Data	Ukuran	Total yang Tidak Nol	Total Data yang Selisih	% Error data
Matrik M	16.719.921	28039	3947	0,000025
Matrik Sh	16.719.921	28039	137	0,329365
Matrik B	16.719.921	340	15	0,420745

Dari **Tabel 2**, didapatkan bahwa terdapat 28.039 anggota data matrik M yang bernilai tidak nol. Hal ini sesuai dengan matrik referensi (**Tabel 1**), baik jumlah maupun indek koreksi matriknya. Namun, terdapat 3947 data dari data yang tidak bernilai nol tersebut, yang memiliki selisih dengan data matrik referensi M. Nilai rata-rata selisih tersebut adalah sebesar 0,000025% dari data referensi. Meskipun jumlah data yang tidak nol untuk matrik Sh sama dengan jumlah data yang tidak nol matrik M, namun hanya terdapat 137 data matrik Sh yang memiliki selisih dari matrik referensi Sh, dengan rata-rata selisih sebesar 0,329365%. Perbedaan ini disebabkan karena proses data yang diisikan pada kedua matrik berbeda. Untuk matrik M, nilainya hanya tergantung dari luas tiap elemen segitiga, dimana luas elemen (A) hanya ditentukan oleh posisi tiap titik yang membentuk elemen tersebut.

Nilai matrik Sh lebih komplek dari matrik M, sehingga dimungkinkan lebih banyak pembulatan pada saat proses pengisian data. Hal ini yang menyebabkan rata-rata selisih matrik Sh terhadap referensi lebih besar dari pada matrik M terhadap referensinya.

Berbeda dengan kedua matrik sebelumnya, di dalam matrik B yang dibangun dalam penelitian ini hanya terdapat 340 data anggota matrik B yang bernilai tidak nol, dan 15 diantaranya memiliki selisih dengan matrik B referensi dengan rata-rata selisih sebesar 0,420745%. Dengan rata-rata selisih data ketiga matrik yang dibangun cukup kecil (kurang dari 1%,) dapat disimpulkan bahwa matrik yang dibangun dengan bahasa C berbasis CPU-GPU sesuai dengan data referensi yang dibangun di dalam Matlab 7.4 (berbasis CPU).

Penyelesaian *Time Dependent* ODE, dengan menggunakan ODESOLVER Matlab 7.4, dari sistem matrik yang dibangun, dilakukan untuk memastikan bahwa data matrik yang dibangun dapat diselesaikan (fungsional). Penyelesaian dilakukan untuk selang waktu 30 detik selama 1 jam. Dengan demikian didapatkan cuplikan gambar pada detik terakhir seperti yang terlihat pada **Gambar 5**.



Gambar 5. Penyelesaian Sistem Data Matrik Hasil dengan ODESOLVER Matlab 7.4

Penyelesaian dengan ODESOLVER Matlab 7.4 ini untuk memastikan bahwa data matrik yang dibangun dengan bahasa pemrograman C berbasis CPU-GPU berjalan baik dan dapat diselesaikan

(fungsional). **Gambar 5** menunjukan bahwa matrik yang dibangun dengan C memberikan hasil simulasi yang sesuai dengan matrik referensi berbasis CPU. Hal ini juga menjadi kunci untuk penelitian selanjutnya dalam menyelesaikan *Time Dependent* ODE matrik SWE dengan metode elemen hingga, menggunakan CPU-GPU.

D. SIMPULAN

Dari penelitian ini dapat disimpulkan bahwa :

a. Bahasa pemrograman C berbasis CPU-GPU, dapat digunakan untuk membangun data matrik reduksi persamaan penjalaran tsunami (*Shallow Water Equation*) dengan metode elemen hingga.
b. Data matrik reduksi yang dibangun sesuai dengan data referensi yang dibangun dengan *software* Matlab 7.4 berbasis CPU.
c. Waktu yang dibutuhkan bahasa pemrograman C (berbasis CPU-GPU) untuk membangun data matrik reduksi adalah 486.75 ms, 9 kali lebih cepat dari data referensi yang dibangun dengan *software* Matlab 7.4 berbasis CPU (4220.98 ms).

DAFTAR RUJUKAN

Boyer, V., & El Baz, D. (2013). Recent advances on GPU computing in operations research. *Proceedings - IEEE 27th International Parallel and Distributed Processing Symposium Workshops and PhD Forum, IPDPSW 2013*, *May 2013*, 1778–1787. (https://doi.org/10.1109/IPDPSW.2013.45, diakses 4 2020)

Breitbart, J. (2008). A framework for easy CUDA integration in C ++ applications. *Diplome Thesis Universität Kassel Kassel*. (http://www.plm.eecs.uni-kassel.de/plm/fileadmin/pm/publications/breitbart/framework-_for_easy_cuda_integration_c___applications.pdf, diakses 4 Oktober 2020)

Erwina, N., Adytia, D., Pudjaprasetya, S. R., & Nuryaman, T. (2020). Staggered conservative scheme for 2-dimensional shallow water flows. *Fluids*, *5*(3), 1–18. (https://doi.org/10.3390/fluids5030149, diakses 4 Oktober 2020)

Kerkeni, L., Ruano, P., Delgado, L. L., Picco, S., Villegas, L., Tonelli, F., Merlo, M., Rigau, J., Diaz, D., & Masuelli, M. (2016). The Discontinuous Galerkin Finite Element Method for Ordinary Differential Equations. In *Intech* (Issue tourism, p. 13). (https://www.intechopen.com/books/advanced-biometric-technologies/liveness-detection-in-biometrics, diakses 5 Oktober 2020)

Supian, N. M., & Rusli, N. (2018). Approximating one-dimensional coupled shallow-water equations for predicting tsunami wave propagation using finite difference method. *AIP Conference Proceedings*, *2013*(December).

(https://doi.org/10.1063/1.5054223, diakses 5 Oktober 2020)

Ueng, S. Z., Lathara, M., Baghsorkhi, S. S., & Hwu, W. M. W. (2008). CUDA-Lite: Reducing GPU programming complexity. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *5335 LNCS*(May 2014), 1–15. (https://doi.org/10.1007/978-3-540-89740-8_1, diakses 5 Oktober 2020)

Vázquez, F., Martínez, J. A., & Garzón, E. M. (2013). GPU Computing. *Encyclopedia of Systems Biology*, *1978*, 845–849. (https://doi.org/10.1007/978-1-4419-9863-7_998, diakses 5 Oktober 2020)