JURNAL REKAYASA GEOFISIKA INDONESIA
Journal homepage: https://jrgi.uho.ac.id/
Vol. 07 No. 01, January 2025 (001-026) doi: 10.56099/jrgi.v8i01.194

Analisis Kinerja Komputasi Full Vectorization dan Nested Loop pada Forward Modeling Gravitasi Last–Kubik

Computational Performance Analysis of Full Vectorization and Nested Loop in Last–Kubik Gravity Forward Modeling

La Hamimu[1*], Al Rubayin[2]

[1]Jurusan Teknik Geofisika, Universitas Halu Oleo, Kendari Sulawesi Tenggara
[2] Jurusan Teknik Geofisika, Universitas Halu Oleo, Kendari Sulawesi Tenggara

Article history:

Received: 02-Februari-2026
Accepted: 04-April-2026

Keywords:
Computational Perfomance;
Full Vectorization;
Nested Loop;
Gravity Forward Modeling;
Last–Kubik Kernel.

Corespondent author:
lahamimu.geoph@gmail.com

Abstrak. Forward modeling gravitasi berbasis kernel Last–Kubik membutuhkan komputasi intensif ketika jumlah blok model dan titik observasi meningkat. Kajian ini menganalisis kinerja komputasi implementasi *Full Vectorization* (FV) dibandingkan *Nested Loop* (NL) tanpa mengubah formulasi matematis kernel Last–Kubik. Pengujian dilakukan menggunakan model sintetik dengan 20 konfigurasi, dari 20.000 hingga 168.200 blok. Kinerja algoritma dievaluasi berdasarkan kesesuaian respon gravitasi, waktu komputasi, dan penggunaan memori. Waktu komputasi dianalisis menggunakan *Total Execution Time* (TET) dan *Mean Core Computational Time* (MCCT) dari 30 pengulangan, sedangkan penggunaan memori dievaluasi menggunakan *Kernel Workspace Memory Increase* (KWMI). Hasil menunjukkan bahwa respon gravitasi FV dan NL identik secara numerik, dengan selisih absolut maksimum sebesar 0 pada konfigurasi yang diuji. FV memberikan percepatan MCCT sekitar 6–7 kali pada model kecil hingga menengah, tetapi menurun menjadi 1,56 kali pada 168.200 blok. Sebaliknya, kebutuhan memori FV sekitar 18 kali lebih besar daripada NL. Analisis regresi menunjukkan hubungan linier yang sangat kuat antara KWMI dan jumlah pasangan observasi–blok. Hasil kajian menunjukkan bahwa FV efektif untuk mempercepat forward modeling, tetapi memiliki keterbatasan skalabilitas akibat kebutuhan memori yang tinggi.

Abstract. *Gravity forward modeling based on the Last–Kubik kernel requires intensive computation as the number of model blocks and observation points increases. This study analyzes the computational*

performance of Full Vectorization (FV) compared with Nested Loop (NL) without modifying the mathematical formulation of the Last–Kubik kernel. Experiments were conducted using synthetic models with 20 configurations ranging from 20,000 to 168,200 blocks. Algorithm performance was evaluated based on gravity-response consistency, computational time, and memory usage. Computational time was assessed using Total Execution Time (TET) and Mean Core Computational Time (MCCT) from 30 repetitions, while memory usage was evaluated using Kernel Workspace Memory Increase (KWMI). The results show that the FV and NL implementations produce numerically identical gravity responses, with a maximum absolute difference of 0 for the tested configurations. FV provides an MCCT speedup of approximately 6–7 times for small to medium-sized models, but the speedup decreases to 1.56 times at 168,200 blocks. In contrast, FV requires approximately 18 times more memory than NL. Regression analysis shows a very strong linear relationship between KWMI and the number of observation–block pairs. The results demonstrate that FV effectively accelerates gravity forward modeling but has scalability limitations due to its substantially higher memory requirements.

1. PENDAHULUAN

Metode gravitasi merupakan salah satu metode geofisika yang banyak digunakan untuk mengidentifikasi variasi distribusi densitas bawah permukaan berdasarkan pengukuran perubahan percepatan gravitasi bumi. Metode ini telah diterapkan secara luas dalam eksplorasi mineral, hidrokarbon, panas bumi, studi geologi regional, maupun investigasi geoteknik karena mampu memberikan informasi mengenai struktur bawah permukaan secara tidak langsung. Interpretasi data gravitasi umumnya diawali dengan proses forward modeling, yaitu perhitungan respon gravitasi teoritis dari suatu model distribusi densitas yang selanjutnya dibandingkan dengan data hasil pengukuran lapangan. Akurasi dan efisiensi proses forward modeling menjadi faktor penting karena mempengaruhi kualitas interpretasi serta waktu komputasi, terutama ketika digunakan sebagai bagian dari proses inversi.

Salah satu formulasi analitik yang banyak digunakan dalam forward gravity modeling adalah kernel yang dikembangkan oleh Last dan Kubik (1983). Formulasi tersebut menghitung kontribusi gravitasi setiap blok model terhadap setiap titik observasi sehingga

dapat dinyatakan dalam bentuk matriks kernel yang menghubungkan model densitas dengan respon gravitasi. Keunggulan formulasi Last–Kubik terletak pada ketelitian solusi analitiknya serta kemudahan penerapannya pada model dua dimensi yang didiskretisasi menjadi blok-blok persegi panjang. Oleh karena itu, formulasi ini masih banyak digunakan sebagai dasar dalam berbagai penelitian pemodelan dan inversi gravitasi.

Meskipun formulasi matematis Last–Kubik telah mapan, implementasi komputasinya masih sering dilakukan menggunakan algoritma nested-loop. Pada pendekatan ini, setiap elemen matriks kernel dihitung secara berurutan melalui perulangan terhadap seluruh titik observasi dan seluruh blok model. Implementasi tersebut relatif sederhana, namun waktu komputasinya meningkat secara signifikan seiring bertambahnya jumlah blok model dan titik observasi. Kondisi ini menjadi kendala ketika model memiliki resolusi yang lebih tinggi atau ketika proses forward modeling harus dilakukan secara berulang dalam algoritma inversi.

Salah satu pendekatan untuk meningkatkan efisiensi komputasi adalah full vectorization, yaitu teknik pemrograman yang menggantikan operasi perulangan (loop) dengan operasi matriks yang dieksekusi secara langsung oleh pustaka aljabar linear yang telah dioptimalkan pada MATLAB. Pendekatan ini memungkinkan seluruh elemen matriks kernel dihitung secara simultan sehingga dapat mengurangi interpreter overhead yang umumnya muncul pada implementasi berbasis nested-loop. Namun demikian, peningkatan kecepatan komputasi tersebut diikuti oleh peningkatan kebutuhan memori karena seluruh matriks antara (intermediate matrices) dibentuk secara bersamaan selama proses komputasi.

Beberapa penelitian sebelumnya lebih banyak memanfaatkan formulasi Last–Kubik sebagai dasar pemodelan gravitasi atau pengembangan algoritma inversi, sedangkan kajian yang secara khusus mengevaluasi transformasi implementasi komputasi dari nested-loop menjadi full vectorization masih relatif terbatas. Selain itu, perbandingan kuantitatif mengenai efisiensi kedua pendekatan tersebut pada formulasi kernel Last–Kubik belum banyak dilaporkan dalam literatur, khususnya untuk implementasi menggunakan MATLAB.

Berdasarkan latar belakang tersebut, penelitian ini bertujuan mengembangkan algoritma full vectorization untuk forward modeling gravitasi berbasis kernel Last–Kubik tanpa mengubah formulasi matematis aslinya. Kinerja algoritma yang diusulkan kemudian dibandingkan secara langsung dengan implementasi nested-loop menggunakan model densitas dan parameter komputasi yang identik. Perbandingan dilakukan terhadap kesesuaian respon gravitasi yang dihasilkan serta efisiensi waktu komputasi sehingga peningkatan kinerja yang diperoleh semata-mata berasal dari perubahan arsitektur algoritma, bukan dari perubahan model matematis. Selain itu, penelitian ini juga membahas keterbatasan implementasi full vectorization, terutama peningkatan kebutuhan memori pada model berukuran sangat besar yang berpotensi menyebabkan kondisi out-of-memory.

Dalam beberapa tahun terakhir, penelitian penulis lebih banyak diarahkan pada pengembangan algoritma komputasi berbasis MATLAB untuk berbagai metode geofisika, antara lain forward dan inverse modeling Vertical Electrical Sounding (Hamimu, L., Safani, J., Ngkoimani, L.O., 2015), pemodelan dispersi gelombang Love (Hamimu, L., 2023), analisis sensitivitas parameter geofisika (Hamimu, L., Cahyono, E., Budiman, H., Haraty, S. R., Ransi, N., 2024a), tomografi refraksi seismik (Hamimu, L., Haraty, S. R., Rubaiyn, A., Juarzan, L. O. I., Indrawati, 2024b) serta pengembangan algoritma inversi berbasis systematic search (Hamimu, L, dkk., 2025) Seluruh penelitian tersebut menunjukkan bahwa MATLAB merupakan lingkungan komputasi yang efektif untuk pengembangan perangkat lunak geofisika yang bersifat terbuka dan mudah dikembangkan.

2. TINJAUAN PUSTAKA

2.1. Forward Modeling Gravitasi

Metode gravitasi merupakan salah satu metode geofisika pasif yang memanfaatkan variasi percepatan gravitasi bumi akibat perbedaan distribusi densitas batuan di bawah permukaan. Dalam interpretasi data gravitasi, proses forward modeling merupakan tahapan fundamental yang digunakan untuk menghitung respon gravitasi teoritis berdasarkan model distribusi densitas yang diasumsikan. Hasil forward modeling kemudian dibandingkan dengan data observasi sebagai dasar dalam proses inversi maupun interpretasi geologi.

Secara umum, forward modeling dilakukan dengan mendiskretisasi bawah permukaan menjadi sejumlah blok persegi panjang dua dimensi atau prisma tiga dimensi. Respon gravitasi total diperoleh melalui prinsip superposisi, yaitu penjumlahan kontribusi gravitasi seluruh elemen model terhadap setiap titik observasi. Pendekatan ini memungkinkan berbagai bentuk geologi yang kompleks direpresentasikan dalam bentuk model numerik dengan ketelitian yang tinggi.

Dalam bentuk diskret, hubungan antara distribusi densitas dan respon gravitasi dapat dinyatakan sebagai

$$\mathbf{g} = \mathbf{A}\,\mathbf{m} \tag{1}$$

dengan **g** menyatakan vektor respon gravitasi, **A** merupakan matriks kernel yang menggambarkan pengaruh setiap blok model terhadap setiap titik observasi, sedangkan **m** merupakan vektor densitas model. Matriks kernel menjadi komponen utama dalam proses forward maupun inverse modeling karena seluruh informasi geometrik sistem tersimpan di dalamnya (Grandis, 2009).

Perkembangan komputasi numerik menyebabkan forward modeling tidak hanya digunakan sebagai alat simulasi, tetapi juga sebagai engine utama berbagai algoritma inversi geofisika. Oleh karena itu, efisiensi pembentukan matriks kernel menjadi salah satu aspek penting dalam meningkatkan performa komputasi, khususnya ketika jumlah blok model semakin besar.

2.2. Kernel Last–Kubik pada Pemodelan Gravitasi

Salah satu formulasi analitik yang paling banyak digunakan dalam forward modeling

gravitasi adalah kernel yang dikembangkan oleh Last dan Kubik (1983). Formulasi ini menghitung respon gravitasi akibat blok persegi panjang dua dimensi menggunakan solusi analitik sehingga tidak memerlukan pendekatan numerik terhadap integral gravitasi.

Keunggulan utama formulasi Last–Kubik adalah ketelitian perhitungan yang tinggi, stabilitas numerik, serta kemudahan implementasinya pada model blok dua dimensi. Setiap elemen matriks kernel dihitung berdasarkan hubungan geometrik antara titik observasi dan pusat blok model melalui kombinasi fungsi logaritma dan fungsi trigonometri invers. Dengan demikian, respon gravitasi seluruh model dapat diperoleh melalui operasi perkalian matriks antara kernel dan distribusi densitas.

Hingga saat ini formulasi Last–Kubik masih banyak digunakan sebagai dasar berbagai penelitian mengenai forward modeling maupun inversi gravitasi karena mampu menghasilkan respon gravitasi yang konsisten terhadap solusi analitik. Namun demikian, sebagian besar penelitian lebih menitikberatkan pada pengembangan metode inversi, sedangkan implementasi algoritma pembentukan kernel relatif masih menggunakan pendekatan konvensional berbasis nested-loop.

Pada implementasi tersebut, setiap elemen kernel dihitung secara berurutan melalui dua tingkat perulangan, yaitu terhadap seluruh titik observasi dan seluruh blok model. Seiring bertambahnya ukuran model, jumlah iterasi meningkat secara signifikan sehingga waktu komputasi menjadi lebih lama.

2.3. Full Vectorization dan Nested Loop pada Komputasi MATLAB

MATLAB merupakan salah satu lingkungan komputasi numerik yang banyak digunakan dalam geofisika karena menyediakan kemampuan manipulasi matriks, visualisasi data, dan pustaka numerik yang lengkap. Berbeda dengan bahasa pemrograman terkompilasi seperti C atau Fortran, performa MATLAB sangat dipengaruhi oleh cara algoritma diimplementasikan. Penggunaan perulangan (for-loop) atau nested loop (NL) dalam jumlah besar umumnya meningkatkan interpreter overhead sehingga memperlambat waktu komputasi (Hamimu, L., Safani, J., Ngkoimani, L.O., 2015).

Salah satu pendekatan yang banyak dikembangkan untuk meningkatkan efisiensi komputasi MATLAB adalah *full vectorization* (FL), yaitu menggantikan operasi yang dilakukan secara berulang menjadi operasi matriks yang dieksekusi secara simultan oleh pustaka aljabar linear internal MATLAB. Pendekatan ini memanfaatkan operasi array, matrix algebra, serta fungsi-fungsi bawaan MATLAB sehingga sebagian besar proses komputasi dapat dijalankan oleh rutinitas BLAS yang telah dioptimalkan (Moskovka, A., Rahman, T., Valdman, J., Vatne, J.E., 2026).

Beberapa penelitian terbaru menunjukkan bahwa *vectorization* mampu meningkatkan efisiensi waktu komputasi secara signifikan pada berbagai bidang numerik. Moskovka dkk. mengembangkan kerangka *vectorized basic linear algebra* untuk meningkatkan keterbacaan sekaligus performa kode MATLAB pada komputasi metode elemen hingga melalui pemisahan antara formulasi matematis dan

implementasi *vectorization* (Moskovka, A., Rahman, T., Valdman, J., Vatne, J.E., 2026). Selain itu, Tchuigwa dkk. memperkenalkan algoritma *vectorization* berbasis MATLAB untuk proses perakitan matriks global metode elemen hingga yang menghasilkan percepatan komputasi beberapa kali lipat dibandingkan implementasi standar berbasis for-loop, tanpa mengubah formulasi matematis yang digunakan (Tchuigwa, B. S. S., Krmela, J., Pokorny, J., Krmelov´, V., Jilek, P., 2024).

Pada bidang geofisika komputasi, optimasi algoritma juga mulai diarahkan pada peningkatan efisiensi memori dan waktu komputasi. Wang dkk. mengembangkan strategi komputasi pada simulasi gelombang seismik tiga dimensi dengan memanfaatkan operasi vektor dan optimasi arsitektur komputasi sehingga mampu meningkatkan efisiensi memori serta mempercepat waktu simulasi secara signifikan. Temuan tersebut menunjukkan bahwa peningkatan performa komputasi dapat dicapai melalui optimasi implementasi algoritma tanpa mengubah model fisika yang digunakan (Wang, W., dkk., 2026).

3. METODE PENELITIAN

Kajian ini merupakan studi komputasi numerik yang bertujuan mengembangkan algoritma Full Vectorization untuk mempercepat proses forward modeling gravitasi berbasis kernel Last–Kubik. Pengembangan dilakukan pada tahap implementasi algoritma tanpa mengubah formulasi matematis kernel gravitasi yang dikembangkan oleh Last dan Kubik. Untuk mengevaluasi peningkatan kinerja algoritma, implementasi yang diusulkan dibandingkan secara langsung dengan algoritma konvensional berbasis nested-loop menggunakan model sintetik, parameter komputasi, serta konfigurasi titik observasi yang identik. Dengan demikian, setiap perbedaan performa yang diperoleh sepenuhnya berasal dari perubahan arsitektur algoritma komputasi, bukan akibat perubahan formulasi matematis maupun model geologi yang digunakan.

3.1. Penyusunan Model Sintetik

Model bawah permukaan direpresentasikan dalam bentuk blok persegi panjang dua dimensi (*rectangular cells*) dengan ukuran blok tetap sebesar 50 m $\times$ 50 m**.** Pendiskretisasian model dilakukan menggunakan dua parameter utama, yaitu jumlah blok horizontal (nx) dan jumlah blok vertikal (nz). Koordinat setiap blok ditentukan berdasarkan posisi pusat blok (*cell-centered coordinates*) sehingga seluruh perhitungan respon gravitasi dilakukan terhadap titik pusat masing-masing blok.

Distribusi densitas awal diasumsikan homogen dengan nilai nol, kemudian disisipkan sebuah anomali densitas berbentuk tanda (+) di bagian tengah model. Bentuk anomali tersebut dibangun menggunakan dua lengan, yaitu lengan vertikal dan lengan horizontal, sehingga menghasilkan distribusi densitas yang simetris terhadap pusat model. Kontras densitas anomali ditetapkan sebesar 1000 kg/m^3. Bentuk anomali ini dipilih untuk mempermudah proses verifikasi visual terhadap kesesuaian respon gravitasi yang dihasilkan oleh kedua algoritma. Implementasi pembentukan model sintetik pada algoritma

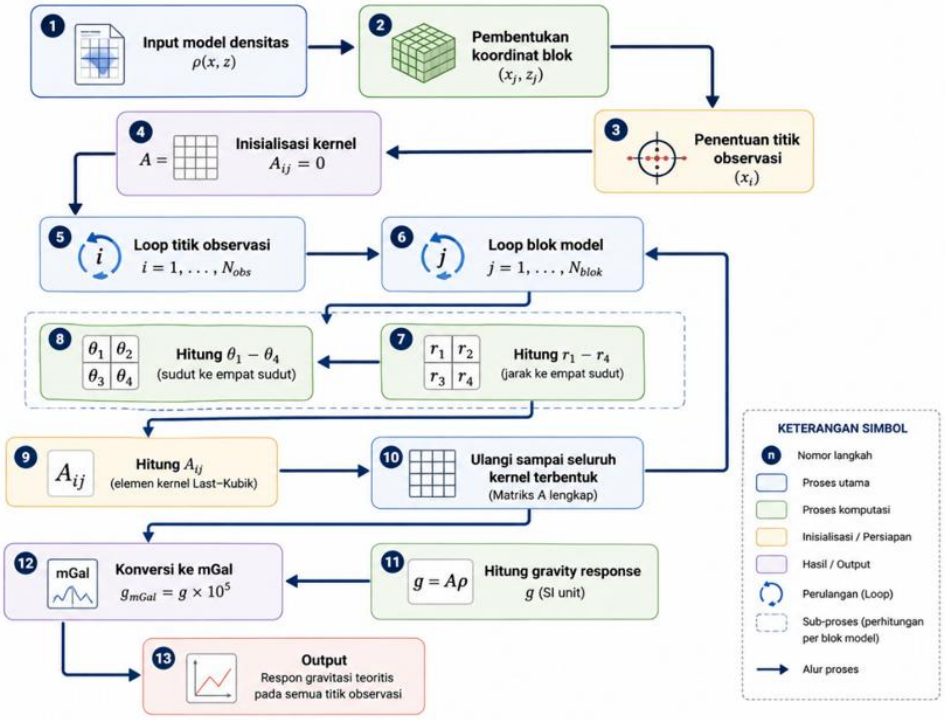
nested-loop maupun full vectorization dibuat identik sehingga tidak mempengaruhi proses perbandingan performa.

Titik observasi ditempatkan tepat di atas pusat setiap blok model sehingga jumlah titik observasi sama dengan jumlah blok pada arah horizontal. Posisi koordinat horizontal ditentukan berdasarkan pusat blok, sedangkan koordinat vertikal setiap blok dihitung menggunakan sistem koordinat *cell-centered*. Pada implementasi nested-loop, koordinat pusat blok dibangun menggunakan fungsi `repmat`, sedangkan pada implementasi full vectorization koordinat dibentuk menggunakan kombinasi fungsi `meshgrid`, transformasi vektor kolom, dan operasi matriks. Perbedaan ini menjadi dasar transformasi algoritma yang diusulkan karena memungkinkan seluruh operasi geometrik dihitung secara serentak melalui manipulasi matriks.

3.2. Implementasi Algoritma Nested-Loop (NL)

Implementasi pertama menggunakan algoritma konvensional berbasis *nested-loop*. Pada pendekatan ini setiap elemen matriks kernel dihitung melalui dua tingkat perulangan. Perulangan pertama dilakukan terhadap seluruh titik observasi, sedangkan perulangan kedua dilakukan terhadap seluruh blok model. Pada setiap pasangan titik observasi dan blok model dihitung empat komponen jarak (r_1–r_4), empat komponen sudut (θ_1–θ_4), kemudian digunakan untuk membentuk satu elemen kernel Last–Kubik. Setelah seluruh elemen selesai dihitung, matriks kernel dikalikan dengan vektor distribusi densitas untuk memperoleh respon gravitasi teoritis. .

Secara keseluruhan, algoritma nested-loop dimulai dari pembentukan model densitas dan koordinat pusat blok, kemudian menentukan

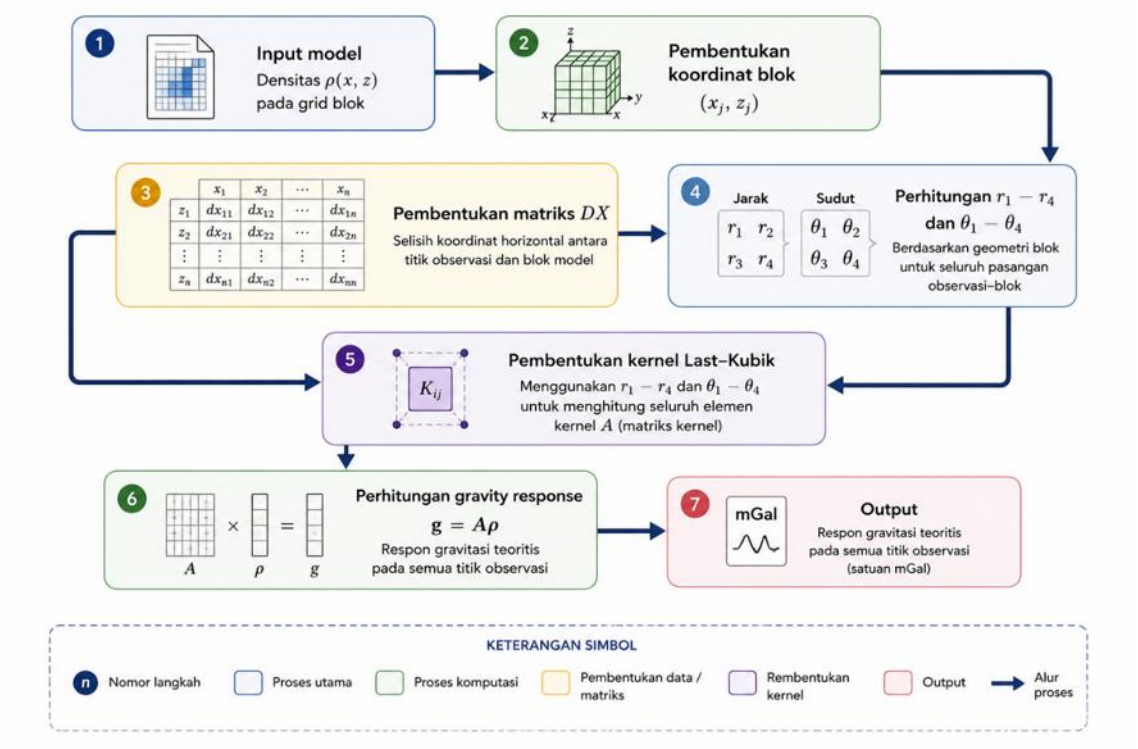


Gambar 1. Flowchart algoritma nested-loop

titik observasi dan menyiapkan matriks kernel. Untuk setiap titik observasi, algoritma melakukan perulangan terhadap seluruh blok model. Pada setiap pasangan observasi–blok, empat komponen jarak dan empat komponen sudut dihitung, kemudian digunakan untuk memperoleh satu elemen kernel Last–Kubik A_{ij}. Proses tersebut diulang hingga seluruh elemen matriks kernel terbentuk. Selanjutnya, matriks kernel dikalikan dengan vektor densitas untuk memperoleh respon gravitasi teoritis yang kemudian dikonversi ke satuan mGal. Secara visual algoritma nested-loop dapat ditampilkan dalam bentuk flowchart seperti pada **Gambar 1**

3.3. Implementasi Algoritma Full Vectorization (FV)

Algoritma yang diusulkan pada kajian ini menggantikan proses perulangan menjadi operasi matriks (FV). Langkah pertama dilakukan dengan membangun seluruh koordinat blok menggunakan fungsi meshgrid, kemudian koordinat tersebut diubah menjadi vektor kolom sehingga seluruh pasangan koordinat blok dan titik observasi dapat dibentuk secara simultan. Selanjutnya matriks selisih koordinat horizontal dihitung menggunakan fungsi bsxfun, sedangkan koordinat vertikal diperluas menggunakan repmat. Berdasarkan kedua matriks tersebut kemudian dihitung seluruh komponen jarak, komponen sudut, serta seluruh elemen kernel Last–Kubik secara serentak melalui operasi matriks. akibat proses interpretasi perulangan dapat dikurangi sehingga waktu komputasi menjadi lebih singkat. Secara keseluruhan, algoritma dimulai dengan memasukkan model distribusi densitas dan membentuk koordinat pusat seluruh blok menggunakan meshgrid. Koordinat tersebut kemudian diubah menjadi



Gambar 2. Flowchart algoritma full vectorization

bentuk vektor dan digunakan untuk membangun matriks selisih koordinat antara seluruh titik observasi dan seluruh blok model menggunakan operasi bsxfun. Berdasarkan matriks tersebut, seluruh komponen jarak $r_1 - r_4$ dan sudut $\theta_1 - \theta_4$ dihitung secara simultan, kemudian digunakan untuk membentuk seluruh elemen kernel Last–Kubik tanpa nested-loop. Setelah matriks kernel terbentuk, respon gravitasi dihitung melalui perkalian matriks antara kernel dan vektor densitas, kemudian dikonversi ke satuan mGal. Dengan demikian, karakteristik utama algoritma yang diusulkan terletak pada transformasi arsitektur komputasi dari evaluasi elemen-per-elemen menjadi operasi matriks secara simultan, sementara formulasi matematis kernel Last–Kubik tetap dipertahankan. Secara visual algoritma FV ditampilkan dalam bentuk flowchart seperti pada **Gambar 2**. Untuk mendukung reproduktibilitas hasil kajian ini, listing program MATLAB lengkap untuk implementasi *Nested Loop* (NL) dan *Full Vectorization* (FV) disediakan pada Lampiran A dan Lampiran B. Listing tersebut mencakup pembentukan model sintetik, pembentukan kernel Last–Kubik, perhitungan respon gravitasi, serta prosedur pengukuran waktu komputasi dan penggunaan memori yang digunakan dalam kajian ini.

3.4. Pengukuran Waktu Komputasi, Penggunaan Memori, dan Analisis Perbandingan Kinerja

Evaluasi performa algoritma dilakukan melalui pengukuran waktu komputasi menggunakan fungsi tic dan toc pada MATLAB. Pengukuran dibatasi hanya pada proses pembentukan kernel dan perhitungan respon gravitasi sehingga waktu yang diperoleh tidak dipengaruhi oleh proses visualisasi, penyimpanan data, maupun inisialisasi parameter. Untuk mengurangi pengaruh fluktuasi sistem operasi, setiap pengujian diulang sebanyak 30 kali (Nrepeat = 30), kemudian dihitung nilai median, rata-rata (mean), simpangan baku (standard deviation), dan waktu maksimum. Nilai rata-rata digunakan sebagai indikator utama waktu komputasi karena merepresentasikan waktu eksekusi rata-rata yang diperlukan algoritma untuk menyelesaikan proses komputasi pada suatu konfigurasi model.

Selain waktu komputasi, kajian ini juga mengevaluasi kebutuhan memori dari masing-masing algoritma. Pengukuran dilakukan menggunakan fungsi whos yang menghitung total memori workspace sebelum dan sesudah proses pembentukan kernel. Selisih antara kedua kondisi tersebut dinyatakan sebagai memori yang digunakan oleh proses komputasi kernel. Pada implementasi FV, pengukuran juga digunakan untuk memperoleh peak workspace memory yang menggambarkan kapasitas memori maksimum selama proses komputasi berlangsung. Penggunaan fungsi whos dipilih karena tersedia pada berbagai versi MATLAB sehingga hasil pengukuran dapat direproduksi secara konsisten. Kinerja kedua algoritma dibandingkan berdasarkan tiga parameter utama, yaitu:

1. Kesesuaian respon gravitasi, untuk memastikan bahwa transformasi algoritma tidak mengubah hasil numerik yang dihasilkan.

2. Efisiensi waktu komputasi, yang dievaluasi menggunakan nilai rata-rata dari 30 kali pengulangan.
3. Penggunaan memori, yang dievaluasi berdasarkan perubahan kapasitas workspace selama proses pembentukan kernel.

Seluruh analisis dilakukan menggunakan model sintetik dan parameter komputasi yang sama sehingga setiap perbedaan yang diperoleh merepresentasikan pengaruh implementasi algoritma secara langsung. Pendekatan ini memastikan bahwa peningkatan kinerja yang diamati berasal dari transformasi implementasi dari NL menjadi FV, bukan akibat perubahan model matematis ataupun konfigurasi simulasi.

Dua jenis waktu komputasi digunakan dalam kajian ini, yaitu waktu eksekusi total atau *total execution time* (TET) dan waktu komputasi inti atau *core computational time* (CCT). TET merupakan waktu eksekusi keseluruhan program yang diukur menggunakan stopwatch sejak program mulai dijalankan hingga proses selesai. Pengukuran ini merepresentasikan waktu yang diperlukan pengguna untuk menyelesaikan seluruh proses eksekusi program. Sementara itu, CCT merupakan waktu komputasi inti yang diukur menggunakan pasangan perintah *tic* dan *toc* pada bagian algoritma yang mencakup pembentukan kernel Last–Kubik dan perhitungan gravity response. Dengan demikian, CCT digunakan sebagai parameter utama untuk membandingkan efisiensi komputasi kedua algoritma, sedangkan TET digunakan sebagai indikator waktu eksekusi keseluruhan dari perspektif pengguna.

Pengukuran CCT dilakukan sebanyak 30 kali pengulangan pada setiap konfigurasi model. Nilai rata-rata dari 30 pengulangan digunakan sebagai *Mean Core Computational Time* (MCCT), sedangkan simpangan baku digunakan untuk mengevaluasi variasi waktu komputasi. Pendekatan ini memungkinkan perbandingan yang lebih terkontrol antara implementasi NL dan FV karena pengukuran difokuskan pada bagian komputasi inti yang sama.

Penggunaan memori algoritma dievaluasi menggunakan fungsi *whos* pada MATLAB. Karena *whos* melaporkan ukuran variabel yang tersimpan pada workspace MATLAB, parameter yang digunakan dalam kajian ini disebut *Workspace Memory*. Pengukuran dilakukan sebelum dan setelah proses pembentukan kernel untuk memperoleh memori *workspace* awal atau *baseline workspace memory* (BWM) dan memori *workspace* pascakomputasi atau *post-computational workspace memory* (PCWM). Selisih kedua nilai tersebut didefinisikan sebagai peningkatan memori *workspace* atau *workspace memory increase* (WMI). Untuk memfokuskan evaluasi pada beban memori yang ditimbulkan oleh pembentukan matriks kernel Last–Kubik, parameter utama yang digunakan dalam perbandingan NL dan FV adalah peningkatan memori workspace kernel atau *kernel workspace memory increase* (KWMI), yang dihitung sebagai selisih kapasitas workspace sebelum dan setelah pembentukan kernel. Nilai KWMI dinyatakan dalam megabyte (MB). Parameter ini digunakan untuk mengevaluasi konsekuensi kebutuhan memori akibat

perubahan arsitektur komputasi dari NL menjadi FV.

4. HASIL DAN PEMBAHASAN

4.1. Kesesuaian Respon Gravitasi antara NL dan FV

Tahap pertama dalam evaluasi kinerja dilakukan dengan memverifikasi bahwa transformasi algoritma dari NL menjadi FV tidak mengubah hasil numerik *forward modeling* gravitasi. Verifikasi ini merupakan tahap penting karena peningkatan efisiensi komputasi hanya dapat dianggap valid apabila algoritma baru menghasilkan respon gravitasi yang sama dengan implementasi acuan.

Pengujian dilakukan pada konfigurasi model dengan $nx = 200$ dan $nz = 100$. Distribusi densitas, ukuran blok, posisi titik observasi, konstanta gravitasi, serta formulasi kernel Last–Kubik dibuat identik pada kedua implementasi. Perbedaan hanya terdapat pada strategi komputasi kernel, yaitu penggunaan struktur nested-loop pada implementasi NL dan operasi matriks secara simultan pada implementasi FV.

Hasil respon gravitasi dari kedua implementasi kemudian dibandingkan pada seluruh titik observasi. Berdasarkan data hasil pengujian, diperoleh 200 nilai respon gravitasi untuk masing-masing algoritma. Hasil pemeriksaan menunjukkan bahwa setiap nilai respon gravitasi yang dihasilkan oleh implementasi NL memiliki nilai yang sama dengan implementasi FV pada titik observasi yang bersesuaian.

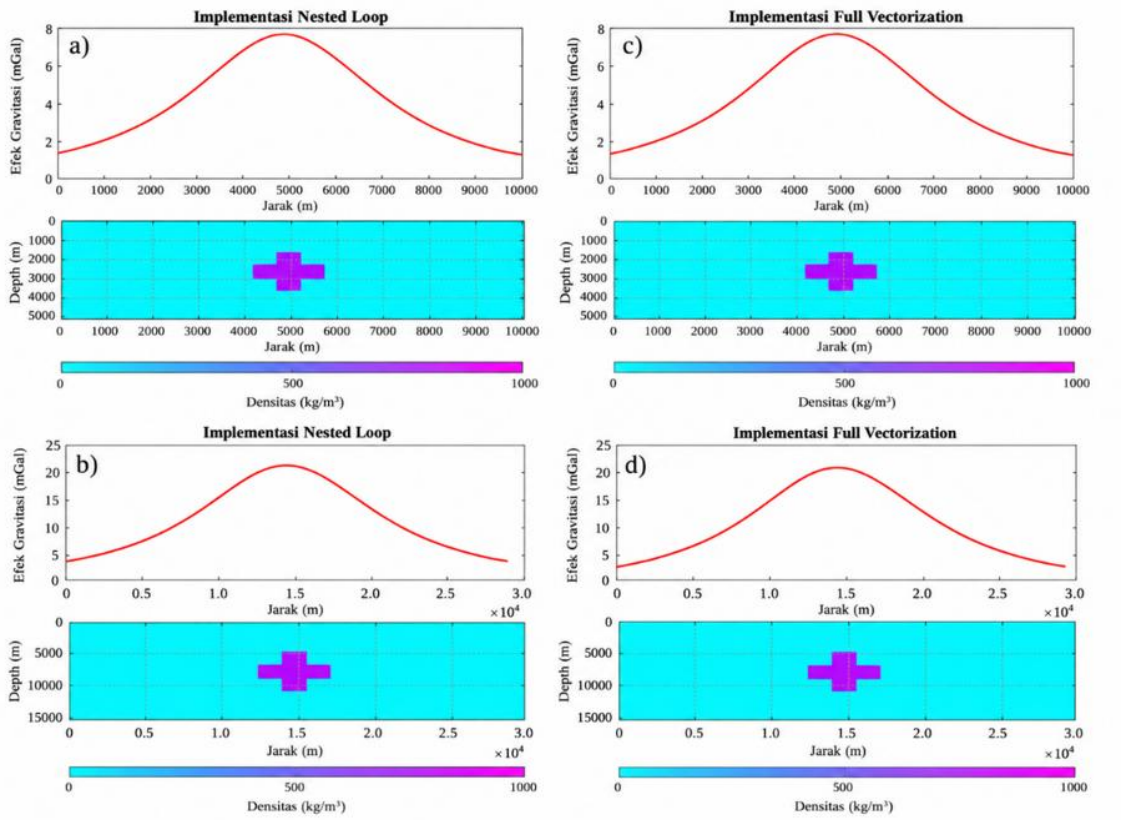
Salah satu aspek penting dalam kajian ini adalah memastikan bahwa perubahan arsitektur algoritma dari NL menjadi FV tidak mengubah formulasi matematis maupun hasil numerik forward modeling gravitasi. Oleh karena itu, sebelum membandingkan efisiensi waktu komputasi dan penggunaan memori, dilakukan verifikasi terhadap kesesuaian respon gravitasi yang dihasilkan oleh kedua implementasi. Verifikasi dilakukan dengan membandingkan respon gravitasi dan distribusi model densitas pada dua ukuran model, yaitu $nx = 200$, $nz = 100$ dan $nx = 580$, $nz = 290$.

Secara visual respon percepatan gravitasi untuk implementasi NL dan FV ditunjukkan oleh **Gambar 3**. Pada **Gambar 3**, panel atas (a) dan (c) masing-masing memperlihatkan hasil implementasi NL dan FV untuk konfigurasi $nx = 200$ dan $nz = 100$ sedangkan panel bawah (b) dan (d) memperlihatkan hasil masing-masing algoritma untuk konfigurasi yang lebih besar, yaitu $nx = 580$ dan $nz = 290$. Penggunaan dua konfigurasi tersebut dimaksudkan untuk memeriksa kesesuaian hasil tidak hanya pada model relatif kecil, tetapi juga pada model yang memiliki jumlah blok jauh lebih besar.

Pada Gambar 3(a), implementasi NL menghasilkan kurva respon gravitasi yang berbentuk halus dengan nilai maksimum sekitar 7–8 mGal pada bagian tengah lintasan. Respon meningkat secara bertahap menuju bagian tengah model, kemudian menurun kembali menuju kedua ujung lintasan. Pola tersebut menunjukkan pengaruh maksimum dari anomali densitas yang berada di sekitar bagian tengah model. Hasil yang ditampilkan pada Gambar 3(c), yang diperoleh menggunakan algoritma FV untuk konfigurasi

model yang sama, memperlihatkan pola respon gravitasi yang secara visual berimpit dan memiliki karakteristik yang sama dengan hasil NL.

Posisi maksimum respon berada pada lokasi yang sama, bentuk kurva dari kedua sisi juga menunjukkan kecenderungan yang sama, dan nilai respon pada bagian ujung lintasan menunjukkan karakteristik yang setara. Secara khusus, kesamaan antara Gambar 3(a) dan 3(c) menunjukkan bahwa operasi vectorization berhasil merepresentasikan kembali proses perhitungan yang sebelumnya dilakukan secara elemen-per-elemen melalui NL. Pada NL, setiap pasangan titik observasi dan blok model dihitung secara berurutan, sedangkan pada FV pasangan tersebut dihitung secara simultan melalui operasi matriks. Meskipun mekanisme evaluasinya berbeda, kedua implementasi menggunakan formulasi kernel Last–Kubik yang sama.



Gambar 3. Kesesuaiam respon percepatan gravitas untuk implementasi NL dan FV

Verifikasi berikutnya dilakukan pada model yang lebih besar dengan nx = 580 dan nz = 290, sebagaimana ditunjukkan pada **Gambar 3(b)** dan **3(d)**. Peningkatan ukuran model menghasilkan perubahan amplitudo respon gravitasi dibandingkan konfigurasi 200×100, dengan nilai maksimum respon berada pada kisaran sekitar $(20–22)\times10^4$ mGal. Meskipun ukuran model meningkat secara signifikan, karakteristik respon tetap memperlihatkan pola yang sama, yaitu peningkatan respon menuju bagian tengah dan penurunan secara bertahap menuju kedua sisi model.

Gambar 3(b) menunjukkan respon yang diperoleh menggunakan NL, sedangkan

Gambar 3(d) menunjukkan hasil FV. Kedua kurva memperlihatkan bentuk, posisi puncak, dan kecenderungan penurunan yang sangat serupa. Puncak respon pada kedua implementasi berada di sekitar bagian tengah bentangan, sesuai dengan posisi anomali densitas yang juga berada di bagian tengah model. Distribusi anomali densitas pada **Gambar 3(b)** dan **3(d)** juga menunjukkan geometri yang sama, sehingga perbandingan respon gravitasi dilakukan pada model yang identik.

Kesamaan tersebut semakin penting karena konfigurasi 580×290 memiliki jumlah blok yang jauh lebih besar daripada konfigurasi 200×100. Dengan demikian, kesesuaian hasil pada model yang lebih besar memberikan indikasi bahwa transformasi dari NL ke FV tetap mempertahankan hasil numerik ketika skala komputasi ditingkatkan. Berdasarkan perbandingan keempat panel tersebut, dapat diamati bahwa NL dan FV menghasilkan respon gravitasi dengan karakteristik yang sama pada kedua konfigurasi model. Kesamaan tidak hanya terlihat pada bentuk kurva respon, tetapi juga pada posisi puncak respon dan lokasi anomali densitas yang menjadi sumber respon.

Hasil verifikasi secara numerik menunjukkan bahwa transformasi implementasi dari NL menjadi FV tidak menghasilkan perubahan pada nilai respon gravitasi. Pada konfigurasi $nx = 200$, $nz = 100$ dan $nx = 580$, $nz = 290$, masing-masing sebanyak 200 dan 580 nilai respon gravitasi yang dihasilkan kedua algoritma menunjukkan kesesuaian numerik sempurna, dengan selisih absolut maksimum sebesar 0. Temuan ini mengonfirmasi bahwa FV mempertahankan formulasi dan hasil numerik kernel Last–Kubik, meskipun mekanisme evaluasi komputasinya diubah dari perhitungan elemen-per-elemen melalui nested-loop menjadi operasi matriks secara simultan. Oleh karena itu, perbandingan waktu komputasi dan penggunaan memori pada tahap berikutnya dapat dilakukan secara langsung karena kedua algoritma menyelesaikan permasalahan forward modeling yang sama dan menghasilkan respon gravitasi yang identik.

4.2. Perbandingan Waktu Komputasi dan Penggunaan Memori

Perbandingan kinerja algoritma NL dan FV dilakukan setelah kedua implementasi menunjukkan kesesuaian respon gravitasi. Dengan demikian, evaluasi pada subbagian ini difokuskan pada konsekuensi perubahan arsitektur komputasi, bukan pada perbedaan formulasi matematis. Kedua algoritma menggunakan model, parameter diskretisasi, distribusi densitas, titik observasi, dan kernel Last–Kubik yang sama. Perbedaan utama terletak pada mekanisme pembentukan kernel dimana NL menghitung setiap elemen kernel secara berurutan melalui dua tingkat perulangan, sedangkan FV menghitung komponen kernel secara simultan menggunakan operasi matriks. Struktur tersebut terlihat langsung pada algoritma NL yang menggunakan perulangan terhadap Nobs (nx) dan nb (nx*nz), sedangkan FV membentuk matriks DX, komponen jarak, sudut, dan kernel secara serentak .

JURNAL REKAYASA GEOFISIKA INDONESIA
Journal homepage: https://jrgi.uho.ac.id/
Vol. 07 No. 01, January 2025 (001-028) doi: 10.56099/jrgi.v4i03.28642

Tabel 1. Perbandingan waktu komputasi dan penggunaan memori pada implementasi NL dan FV

No	nx	nz	Jumlah Blok	TET (detik)		MCCT (detik)		KWMI (MB)	
				NL	FV	NL	FV	NL	FV
1	200	100	20.000	44,07	7,77	1,446	0,232	30,56	549,43
2	220	110	24.200	58,68	9,65	1,932	0,301	40,66	731,26
3	240	120	28.800	76,34	12,04	2,528	0,387	52,77	949,33
4	260	130	33.800	97,72	16,04	3,214	0,482	67,09	1.206,96
5	280	140	39.200	120,29	19,70	3,980	0,606	83,78	1.507,44
6	300	150	45.000	151,92	23,27	5,050	0,741	103,04	1.854,06
7	320	160	51.200	184,27	27,61	6,126	0,902	125,04	2.250,12
8	340	170	57.800	220,02	33,01	7,347	1,069	149,97	2.698,91
9	360	180	64.800	262,82	38,41	8,737	1,263	178,02	3203,73
10	380	190	72.200	311,99	46,40	10,378	1,494	209,36	3.767,88
11	400	200	80.000	365,77	54,75	12,139	1,791	244,18	4.394,65
12	420	210	88.200	425,37	63,57	14,114	2,092	282,66	5.087,34
13	440	220	96.800	489,03	73,12	16,276	2,430	324,99	5.849,24
14	460	230	105.800	559,60	85,23	18,608	2,792	371,35	6.683,65
15	480	240	115.200	638,27	100,67	21,234	3,299	421,92	7.593,87
16	500	250	125.000	721,11	111,49	24,014	3,655	476,88	8.583,19
17	520	260	135.200	790,03	125,82	26,314	4,097	536,42	9.654,90
18	540	270	145.800	888,25	148,99	29,555	4,850	600,72	10.812,31
19	560	280	156.800	960,85	275,12	32,305	9,021	669,96	12.058,71
20	580	290	168.200	1.107,59	753,55	36,899	23,590	744,33	13.397,40

Keterangan: TET = Total Execution Time, MCCT = Mean Core Computational Time (rata-rata dari 30 kali pengulangan), *KWMI = Kernel Workspace Memory Increase, NL = Nested Loop, FV = Full Vectorization*

Pengujian dilakukan pada 20 konfigurasi model, mulai dari nx = 200, nz = 100 hingga nx = 580,nz = 290, dengan jumlah blok meningkat dari 20.000 menjadi 168.200. Waktu komputasi inti diperoleh menggunakan *tic* dan *toc*, sedangkan penggunaan memori ditentukan berdasarkan perubahan ukuran *workspace* yang diperoleh menggunakan *whos*. Program MATLAB mencatat nilai rata-rata waktu komputasi dari pengulangan yang dilakukan serta penggunaan memori setelah pembentukan kernel. Hasil pengukuran waktu komputasi dan penggunaan memori untuk 20 konfigurasi model dengan implementasi NL dan FV disajikan pada **Tabel 1**.

Nilai TET, MCCT, dan KWMI pada **Tabel 1** merupakan hasil pengukuran pada lingkungan komputasi yang digunakan dalam kajian ini, sehingga nilai absolutnya dapat berbeda apabila program MATLAB dijalankan pada laptop atau komputer dengan spesifikasi perangkat keras dan lingkungan perangkat lunak yang berbeda. Namun demikian, karena implementasi NL dan FV diuji pada lingkungan komputasi dan konfigurasi model yang sama, hasil tersebut tetap relevan untuk membandingkan kinerja relatif kedua algoritma

4.2.1. Perbandingan Waktu Komputasi

Hasil pengujian menunjukkan bahwa FV secara umum memberikan waktu komputasi yang lebih rendah dibandingkan NL. Pada konfigurasi terkecil, yaitu nx = 200 dan nz = 100 dengan jumlah 20.000 blok, MCCT dengan implementasi NL sebesar 1,446 detik, sedangkan FV hanya 0,232 detik. Dengan demikian, FV memberikan percepatan sekitar 6,23 kali dibandingkan NL.

Keunggulan tersebut semakin terlihat pada konfigurasi menengah. Sebagai contoh, pada model nx = 400 dan nz = 200 dengan 80.000 blok, MCCT untuk implementasi NL mencapai 12,139 detik, sedangkan FV sebesar 1,791 detik. Rasio keduanya menunjukkan

percepatan sekitar 6,78 kali. Pada konfigurasi nx = 540 dan nz = 270 dengan 145.800 blok, MCCT untuk implementasi NL mencapai 29,555 detik, sementara FV hanya 4,850 detik, sehingga diperoleh percepatan sekitar 6,09 kali.

Secara keseluruhan, hingga konfigurasi 145.800 blok, FV mempertahankan keunggulan waktu komputasi yang relatif konsisten. Rasio percepatan MCCT pada rentang tersebut berada sekitar 6–7 kali, dengan nilai maksimum sekitar 6,95 kali pada konfigurasi nx = 380 dan nz = 190. Temuan ini menunjukkan bahwa penggantian operasi elemen-per-elemen dengan operasi matriks memberikan keuntungan komputasi yang nyata pada ukuran model kecil hingga menengah.

Pola tersebut sesuai dengan karakteristik kedua implementasi. Pada implementasi NL, program harus menjalankan perulangan untuk setiap titik observasi dan setiap blok model, kemudian menghitung empat komponen jarak, empat komponen sudut, dan satu elemen kernel untuk setiap pasangan observasi–blok. Sebaliknya, FV membentuk seluruh komponen jarak, sudut, dan kernel menggunakan operasi array secara simultan. Dengan demikian, pengurangan overhead perulangan dan pemanfaatan operasi matriks MATLAB menjadi faktor utama yang menjelaskan keunggulan waktu komputasi FV.

Namun, hasil pengujian menunjukkan bahwa keunggulan tersebut tidak bersifat konstan untuk seluruh ukuran model. Pada konfigurasi nx = 560 dan nz = 280 dengan 156.800 blok, MCCT dengan implementasi FV meningkat menjadi 9,021 detik, sedangkan Nested Loop sebesar 32,305 detik. Percepatan turun menjadi sekitar 3,58 kali. Penurunan yang lebih tajam terjadi pada konfigurasi terbesar nx = 580 dan nz = 290 dengan 168.200 blok. MCCT dengan implementasi FV meningkat menjadi 23,590 detik, sedangkan Nested Loop sebesar 36,899 detik, sehingga percepatannya hanya sekitar 1,56 kali.

Perubahan tersebut merupakan temuan penting karena menunjukkan bahwa keuntungan FV mulai berkurang ketika ukuran permasalahan semakin besar. Dengan kata lain, *vectorization* tidak menghasilkan peningkatan waktu yang tidak terbatas. Pada ukuran model yang semakin besar, kebutuhan pembentukan dan pengelolaan matriks berukuran besar mulai memberikan beban tambahan terhadap sistem komputasi.

Jika ditinjau berdasarkan TET, kecenderungan yang sama terlihat dengan jelas. Pada konfigurasi 20.000 blok, TET untuk implementasi NL sebesar 44,07 detik, sedangkan FV sebesar 7,77 detik. Pada konfigurasi terbesar 168.200 blok, TET meningkat menjadi 1.107,59 detik pada NL dan 753,55 detik pada FV. Dengan demikian, FV masih memberikan TET yang lebih rendah pada seluruh konfigurasi yang diuji, tetapi selisihnya semakin kecil pada ukuran model terbesar.

Perbedaan antara MCCT dan TET juga memberikan informasi penting. MCCT difokuskan pada komputasi inti yang dibatasi oleh *tic* dan *toc*, sedangkan TET merepresentasikan waktu eksekusi yang diamati secara keseluruhan. Oleh karena itu, MCCT lebih tepat digunakan sebagai parameter utama untuk membandingkan

efisiensi algoritma, sedangkan TET memberikan gambaran tambahan mengenai waktu eksekusi dari perspektif pengguna.

4.2.2. Perbandingan Penggunaan Memori

Keunggulan waktu FV disertai dengan konsekuensi peningkatan penggunaan memori yang sangat besar. Dalam kajian ini, penggunaan memori dinyatakan sebagai KWMI, yaitu peningkatan ukuran workspace yang terjadi selama proses pembentukan kernel. Pengukuran dilakukan menggunakan fungsi *whos* MATLAB dengan membandingkan ukuran *workspace* sebelum dan sesudah pembentukan kernel. Pada implementasi FV, program secara eksplisit menghitung memori terpakai (MB) berdasarkan selisih kedua kondisi tersebut.

Pada konfigurasi terkecil $nx = 200$, $nz = 100$, KWMI untuk implementasi NL hanya sebesar 30,56 MB, sedangkan FV mencapai 549,43 MB. Dengan demikian, FV membutuhkan sekitar 17,98 kali lebih banyak memori *workspace* dibandingkan NL. Perbedaan tersebut tetap konsisten ketika ukuran model diperbesar. Pada konfigurasi $nx = 400$, $nz = 200$ dengan 80.000 blok, KWMI pada implementasi NL sebesar 244,18 MB, sedangkan FV mencapai 4.394,65 MB. Pada konfigurasi terbesar $nx = 580$, $nz = 290$ kebutuhan memori NL meningkat menjadi 744,33 MB, sedangkan FV mencapai 13.397,40 MB. Dengan demikian, rasio penggunaan memori FV terhadap NL berada sangat konsisten pada kisaran 18 kali sepanjang seluruh konfigurasi pengujian. Nilai ini menunjukkan bahwa keuntungan waktu komputasi FV diperoleh dengan konsekuensi peningkatan kebutuhan memori yang signifikan.

Perbedaan tersebut dapat dijelaskan dari struktur penyimpanan kedua algoritma. Pada NL, komponen jarak, sudut, dan kernel dihitung secara bertahap untuk setiap pasangan titik observasi dan blok model. Walaupun matriks kernel tetap disimpan, sebagian besar variabel antara digunakan secara lokal pada setiap iterasi. Sebaliknya, FV membentuk matriks-matriks berukuran besar secara simultan, termasuk r_1–r_4, θ_1–θ_4, dan matriks kernel. Kondisi tersebut menyebabkan kebutuhan workspace meningkat secara signifikan.

4.2.3. Trade-off antara Efisiensi Waktu dan Penggunaan Memori

Jika MCCT dan KWMI dianalisis secara bersamaan, hasil kajian akan memperlihatkan adanya trade-off yang jelas antara efisiensi waktu dan efisiensi memori. Pada konfigurasi model 200×100, FV mampu menurunkan MCCT dari 1,446 detik menjadi 0,232 detik, tetapi KWMI meningkat dari 30,56 MB menjadi 549,43 MB. Pada model 580×290, FV masih memberikan waktu komputasi yang lebih rendah, yaitu 23,590 detik dibandingkan 36,899 detik, tetapi membutuhkan 13.397,40 MB dibandingkan hanya 744,33 MB pada NL.

Dengan demikian, hasil kajian ini menunjukkan bahwa FV lebih unggul dalam efisiensi waktu, tetapi NL lebih unggul dalam efisiensi penggunaan memori. Keunggulan FV menjadi paling signifikan pada ukuran model kecil hingga menengah, sedangkan pada ukuran model terbesar keuntungan waktunya mulai berkurang bersamaan dengan meningkatnya kebutuhan memori.

Temuan ini penting dalam konteks *forward modeling* gravitasi beresolusi tinggi. Peningkatan resolusi model menyebabkan jumlah blok bertambah dan pada saat yang sama memperbesar ukuran matriks yang harus diproses. Pada FV, seluruh pasangan observasi blok direpresentasikan secara simultan dalam struktur matriks. Oleh karena itu, peningkatan resolusi tidak hanya meningkatkan jumlah operasi, tetapi juga meningkatkan kebutuhan penyimpanan data antara (*temporary data*).

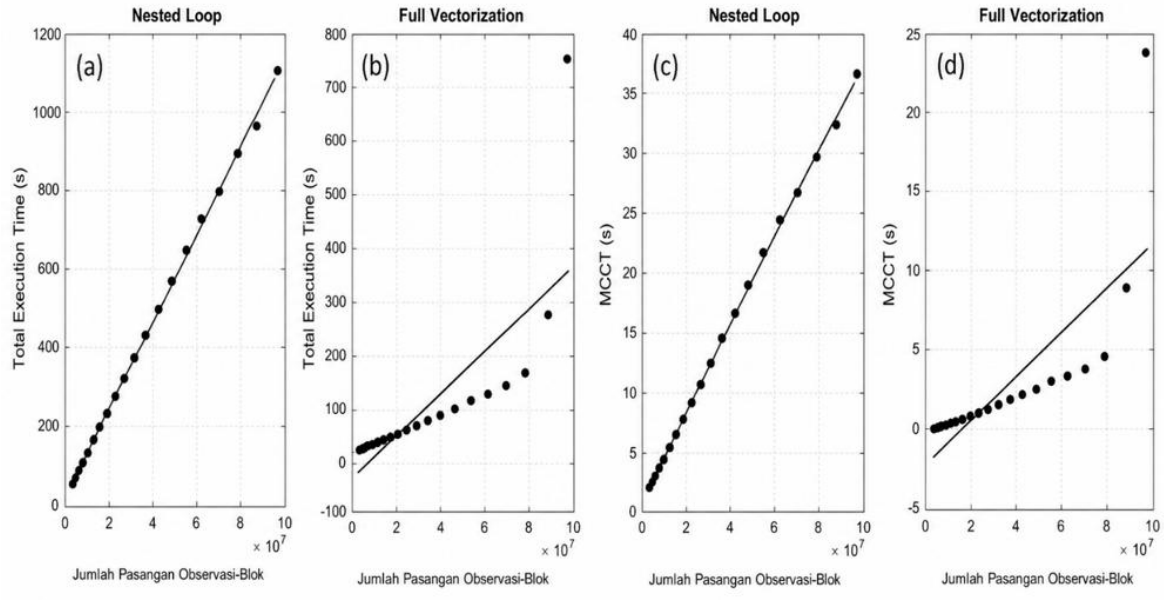
4.2.4. Perubahan Performa pada Ukuran Model Besar

Salah satu hasil yang perlu mendapat perhatian khusus adalah perubahan performa pada dua konfigurasi terakhir. Hingga 145.800 blok, FV memberikan percepatan MCCT sekitar enam kali atau lebih. Namun, pada 156.800 blok percepatan turun menjadi sekitar 3,58 kali, kemudian menjadi hanya 1,56 kali pada 168.200 blok. Penurunan tersebut berlangsung bersamaan dengan peningkatan KWMI pada implementasi FV dari 10.812,31 MB pada 145.800 blok menjadi 12.058,71 MB pada 156.800 blok dan 13.397,40 MB pada 168.200 blok. Sebaliknya, KWMI untuk implementasi NL hanya meningkat dari 600,72 MB menjadi 669,96 MB dan 744,33 MB pada ketiga konfigurasi tersebut.

Korelasi antara peningkatan kebutuhan memori dan berkurangnya keuntungan waktu memberikan indikasi bahwa tekanan memori (*memory pressure*) mulai menjadi faktor pembatas performa FV pada ukuran model besar. Temuan ini memberikan batas praktis yang penting yakni FV sangat efektif untuk mempercepat komputasi selama kapasitas memori masih mampu menampung matriks-matriks antara yang dibentuk secara simultan.

4.2.5. Implikasi terhadap Skalabilitas Algoritma

Hasil pengujian menunjukkan bahwa FV memberikan keuntungan nyata pada rentang model yang diuji, tetapi skalabilitasnya dibatasi oleh kebutuhan memori. Hal ini berbeda dengan NL yang memiliki waktu komputasi lebih panjang, tetapi pertumbuhan penggunaan memorinya jauh lebih terkendali.



Gambar 4. Kurva regresi linier TET dan MCCT untuk implementasi NL dan FV

JURNAL REKAYASA GEOFISIKA INDONESIA
Journal homepage: https://jrgi.uho.ac.id/
Vol. 07 No. 01, January 2025 (001-028) doi: 10.56099/jrgi.v4i03.28642

Oleh karena itu, pemilihan algoritma tidak dapat hanya didasarkan pada waktu komputasi. Untuk model berukuran kecil hingga menengah dengan kapasitas RAM yang memadai, FV merupakan pilihan yang lebih efisien karena dapat mengurangi waktu komputasi secara signifikan. Sebaliknya, ketika ukuran model semakin besar dan kapasitas memori menjadi kendala, NL dapat tetap dipertimbangkan meskipun membutuhkan waktu eksekusi yang lebih lama.

Hasil ini juga memberikan dasar untuk analisis lanjutan menggunakan regresi linier, sebagaimana dibahas pada subbagian berikutnya, untuk memprediksi pertumbuhan waktu komputasi dan kebutuhan memori di luar ukuran model yang dapat diuji secara langsung. Dengan pendekatan tersebut, batas praktis penerapan FV dan NL dapat diperkirakan sebelum proses komputasi dijalankan.

Secara keseluruhan, hasil kajian menunjukkan tiga temuan utama. Pertama, FV memberikan efisiensi waktu yang nyata, dengan percepatan maksimum MCCT sekitar 6,95 kali terhadap NL pada rentang model yang diuji. Kedua, keuntungan tersebut disertai peningkatan KWMI sekitar 18 kali dibandingkan NL. Ketiga, pada ukuran model terbesar, keunggulan waktu FV menurun secara tajam, bersamaan dengan meningkatnya kebutuhan memori. Temuan tersebut menunjukkan bahwa kontribusi utama algoritma yang dikembangkan bukan sekadar menghilangkan NL, tetapi menghasilkan strategi komputasi yang lebih cepat dengan konsekuensi trade-off waktu–memori yang perlu dipertimbangkan dalam penerapan *forward modeling* gravitasi beresolusi tinggi.

4.3. Analisis Regresi Linier pada Implementasi NL dan FV

Analisis regresi linier dilakukan untuk mengevaluasi hubungan antara ukuran permasalahan komputasi dan tiga parameter kinerja utama, yaitu TET, MCCT, dan KWMI. Pendekatan ini digunakan tidak hanya untuk mendeskripsikan kecenderungan peningkatan beban komputasi, tetapi juga untuk mengevaluasi kemampuan model regresi dalam memprediksi perilaku algoritma di luar konfigurasi yang diuji secara langsung.

Variabel prediktor yang digunakan adalah jumlah pasangan interaksi antara titik observasi dan blok model, yang didefinisikan sebagai P = Nobs*Nblok, dengan Nblok = nx*nz. Karena pada konfigurasi eksperimen jumlah titik observasi mengikuti jumlah sel pada arah horizontal, yaitu Nobs = nx, maka jumlah pasangan observasi–blok dapat dituliskan sebagai P = nx2*nz. Variabel P dipilih sebagai prediktor karena secara langsung merepresentasikan jumlah kombinasi observasi–blok yang harus dievaluasi dalam pembentukan kernel Last–Kubik. Dengan demikian, peningkatan P merepresentasikan adanya peningkatan skala permasalahan komputasi yang dihadapi oleh kedua algoritma. Secara visual grafik TET dan MCCT ditunjukkan oleh **Gambar 4**.

Gambar 4(a) memperlihatkan hubungan antara jumlah pasangan observasi–blok dan TET untuk implementasi NL. Titik-titik hasil eksperimen menunjukkan kecenderungan

yang sangat dekat dengan garis regresi linier. Hasil fitting menghasilkan persamaan

$$TET_{NL} = 1{,}12760032*10^{-5}P + 1{,}52594,$$

dengan koefisien determinasi $R^2 = 0{,}99920$. Nilai R^2 yang sangat mendekati satu menunjukkan bahwa model linier mampu merepresentasikan variasi TET NL dengan sangat baik pada rentang konfigurasi yang diuji. Dengan kata lain, peningkatan jumlah pasangan observasi-blok menghasilkan adanya peningkatan waktu eksekusi yang hampir linier.

Kondisi tersebut sesuai dengan karakteristik implementasi NL. Setiap pasangan titik observasi dan blok model diproses secara berurutan melalui struktur perulangan, sehingga peningkatan jumlah pasangan interaksi secara langsung meningkatkan jumlah operasi yang harus dilakukan. Oleh karena itu, TET pada implementasi NL menunjukkan pola pertumbuhan yang relatif stabil sebagaimana terlihat pada **Gambar 4(a)**.

Berbeda dengan NL, **Gambar 4(b)** menunjukkan bahwa hubungan antara P dan TET pada FV tidak dapat direpresentasikan dengan baik oleh satu model linier pada seluruh rentang pengujian. Regresi linier menghasilkan

$$TETFV = 4{,}31877576 \times 10^{-6}P - 60{,}38446,$$

dengan $R^2 = 0{,}57859$. Nilai R^2 tersebut jauh lebih rendah dibandingkan NL. Secara visual, titik-titik pada sebagian besar konfigurasi awal berada jauh di bawah garis regresi, sedangkan dua konfigurasi terbesar menunjukkan peningkatan waktu yang jauh lebih tajam. Fenomena tersebut terutama terlihat pada konfigurasi $nx = 560$, $nz = 280$ dan $nx = 580$, $nz = 290$, ketika TET FV meningkat masing-masing menjadi 275,12 detik dan 753,55 detik.

Hasil tersebut menunjukkan bahwa waktu eksekusi FV mengalami perubahan karakteristik pada ukuran model besar. Pada ukuran kecil hingga menengah, pembentukan dan operasi matriks memberikan keuntungan komputasi yang besar. Namun, ketika jumlah pasangan observasi–blok semakin besar, peningkatan ukuran matriks yang harus dibentuk dan dikelola mulai memberikan beban tambahan terhadap sistem komputasi. Akibatnya, pertumbuhan TET tidak lagi mengikuti kecenderungan linier sederhana.

Dengan demikian, regresi linier pada TET memberikan dua informasi penting. Pertama, NL menunjukkan hubungan linier yang sangat kuat terhadap ukuran permasalahan. Kedua, FV menunjukkan penyimpangan dari linearitas pada ukuran model besar. Penyimpangan tersebut bukan merupakan kegagalan algoritma, tetapi menjadi indikasi adanya batas skalabilitas implementasi FV.

Analisis berikutnya dilakukan terhadap MCCT, yaitu rata-rata waktu komputasi inti yang diperoleh dari pengukuran *tic-toc*. Parameter ini lebih spesifik dibandingkan TET karena pengukurannya difokuskan pada bagian inti proses komputasi. Untuk implementasi NL, hasil regresi menghasilkan persamaan

$$MCCTNL = 3{,}76679890 \times 10^{-7}P + 0{,}006905,$$

dengan $R^2 = 0{,}99945$. Nilai tersebut menunjukkan hubungan yang sangat kuat antara jumlah pasangan observasi–blok dan waktu komputasi inti NL. Hampir seluruh variasi MCCT dapat dijelaskan oleh perubahan

jumlah pasangan observasi–blok. Hasil ini memperkuat temuan pada analisis TET bahwa struktur NL menghasilkan pertumbuhan waktu yang relatif konsisten terhadap peningkatan ukuran permasalahan. Setiap peningkatan P menyebabkan peningkatan jumlah evaluasi kernel yang harus dilakukan sehingga MCCT meningkat secara sistematis.

Sebaliknya, implementasi FV menghasilkan persamaan regresi

$$\mathrm{MCCTFV} = 1{,}37643340 \times 10^{-7}\mathrm{P} - 1{,}898667,$$

dengan $R^2 = 0{,}59495$. Nilai R^2 yang relatif rendah kembali menunjukkan bahwa hubungan antara MCCT dan P pada FV tidak dapat dijelaskan secara memadai menggunakan satu persamaan linier untuk seluruh rentang data.

Pola tersebut konsisten dengan hasil TET. Pada konfigurasi kecil hingga menengah, MCCT untuk implementasi FV meningkat secara bertahap dan tetap jauh lebih rendah dibandingkan NL. Namun, pada konfigurasi $nx = 560$, $nz = 280$, MCCT meningkat menjadi 9,021 detik, dan pada $nx = 580$, $nz = 290$ meningkat menjadi 23,590 detik. Perubahan yang relatif tajam pada dua konfigurasi tersebut menyebabkan titik data menyimpang dari kecenderungan linier awal. Temuan ini menunjukkan bahwa keunggulan FV dalam waktu komputasi memiliki batas praktis. Konsep *Vectorization* mampu mengurangi overhead perulangan dan memanfaatkan operasi array MATLAB secara simultan, tetapi keuntungan tersebut mulai berkurang ketika ukuran matriks antara menjadi semakin besar. Dengan demikian, hasil MCCT memperlihatkan bahwa FV lebih efisien untuk ukuran model kecil hingga menengah, sedangkan pada ukuran model besar perlu diperhatikan peningkatan beban pengelolaan matriks dan penggunaan memori.

Berbeda dengan waktu komputasi FV, hubungan antara KWMI dan jumlah pasangan observasi–blok menunjukkan pola yang sangat berbeda. Pada implementasi NL, regresi menghasilkan

$$\mathrm{KWMI_{NL}} = 7{,}62939455 \times 10^{-6}\mathrm{P} + 0{,}040468,$$

dengan $R^2 \approx 1{,}00000$.

Sementara itu, implementasi FV menghasilkan

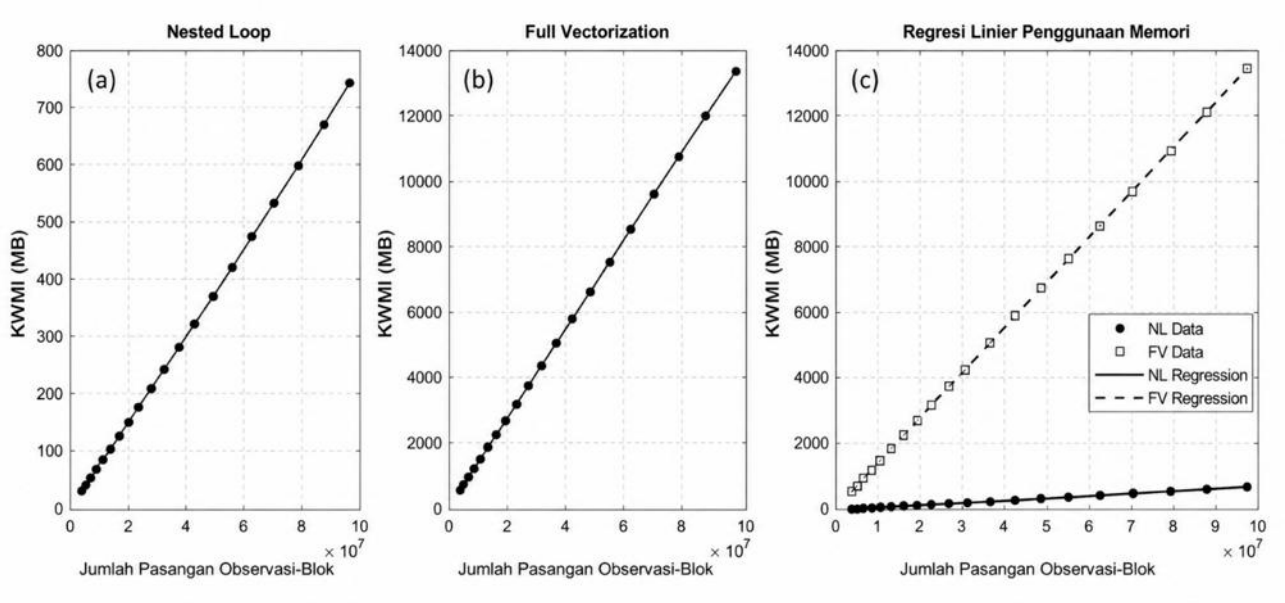
$$\mathrm{KWMI_{FV}} = 1{,}3733 \times 10^{-4}\mathrm{P} + 0{,}116158,$$

dengan $R^2 \approx 1{,}00000$.

Hasil tersebut ditunjukkan secara visual oleh **Gambar 5.** Baik data NL maupun FL mengikuti hubungan linier yang sangat kuat. Dengan demikian, berbeda dengan waktu komputasi FV, kebutuhan memori kedua algoritma dapat diprediksi dengan sangat baik menggunakan jumlah pasangan observasi–blok sebagai variabel prediktor. Perbedaan kemiringan kedua garis regresi menunjukkan perbedaan kebutuhan memori yang sangat signifikan. Koefisien regresi FV sekitar $(1{,}3733 \times 10^{-4})/(7{,}62939455 \times 10^{-6}) \approx 18$ kali lebih besar daripada NL.

Hal ini berarti bahwa untuk setiap peningkatan jumlah pasangan observasi–blok yang sama, pertambahan kebutuhan memori FV secara empiris sekitar 18 kali lebih besar dibandingkan NL. Hasil ini konsisten dengan struktur implementasi kedua algoritma. Implementasi FV menyimpan berbagai matriks antara secara simultan, termasuk matriks koordinat, jarak, sudut, komponen kernel, dan matriks kernel akhir. Sebaliknya, NL menghitung komponen tersebut secara bertahap dalam perulangan sehingga

JURNAL REKAYASA GEOFISIKA INDONESIA
Journal homepage: https://jrgi.uho.ac.id/
Vol. 07 No. 01, January 2025 (001-028) doi: 10.56099/jrgi.v4i03.28642



Gambar 5. Kurva regresi linier KWMI (MB) untuk implementasi NL dan FV

kebutuhan penyimpanan matriks antara relatif lebih rendah

Hasil regresi KWMI merupakan salah satu informasi penting dalam kajian ini karena memberikan dasar kuantitatif untuk menjelaskan trade-off antara efisiensi waktu dan penggunaan memori. FV memberikan keuntungan waktu yang signifikan pada ukuran model tertentu, tetapi keuntungan tersebut diperoleh dengan konsekuensi kebutuhan memori yang jauh lebih tinggi.

Selain itu, hasil regresi memberikan dasar untuk menjelaskan keterbatasan skalabilitas FV. Pada implementasi ini, peningkatan jumlah pasangan observasi–blok menyebabkan peningkatan jumlah elemen matriks yang harus disimpan secara simultan. Hubungan KWMI yang sangat linier menunjukkan bahwa kebutuhan memori dapat diproyeksikan dengan cukup baik dari P. Sebaliknya, hubungan waktu yang tidak lagi linier pada ukuran besar menunjukkan bahwa peningkatan kebutuhan memori mulai berpengaruh terhadap performa komputasi. Dengan kata lain, terdapat indikasi bahwa efisiensi operasi matriks yang menjadi keunggulan utama FV mulai berhadapan dengan biaya pengelolaan data berukuran besar.

Hal ini menjelaskan hasil eksperimen pada dua konfigurasi terbesar. Pada nx = 540, nz = 270, FV masih memiliki MCCT sebesar 4,850 detik dan KWMI sebesar 10.812,31 MB. Ketika ukuran model meningkat menjadi nx = 560, nz = 280, KWMI meningkat menjadi 12.058,71 MB, sementara MCCT meningkat menjadi 9,021 detik. Pada konfigurasi nx = 580, nz = 290, KWMI mencapai 13.397,40 MB dan MCCT meningkat menjadi 23,590 detik. Kenaikan tersebut menunjukkan bahwa titik-titik pada ukuran terbesar perlu diperlakukan sebagai indikator perubahan rezim performa, bukan sekadar kelanjutan dari kecenderungan linier pada ukuran kecil dan menengah.

Karena hubungan KWMI terhadap P memiliki R^2 yang mendekati satu untuk kedua algoritma, model regresi memori dapat digunakan sebagai pendekatan awal untuk memperkirakan batas penggunaan memori

pada ukuran model yang lebih besar. Untuk kapasitas RAM tertentu, batas teoritis jumlah pasangan observasi–blok dapat diperoleh dari hubungan: $M = aP + b$, sehingga

$$P_{max} = (M_{max} - b)/a$$

Pendekatan ini sangat relevan untuk FV karena koefisien pertumbuhan KWMI jauh lebih besar dibandingkan NL. Dengan demikian, peningkatan resolusi model yang tampaknya masih dapat ditangani oleh NL dapat menjadi tidak layak bagi FV karena kebutuhan memori yang lebih tinggi. Namun demikian, nilai P_{max} yang diperoleh melalui regresi harus dipahami sebagai batas estimasi berdasarkan kapasitas *workspace* yang tersedia, bukan batas Out of Memory (OOM) absolut komputer. Fungsi *whos* mengukur kapasitas variabel yang tersimpan pada *workspace* MATLAB, bukan seluruh penggunaan RAM fisik oleh sistem operasi. Oleh karena itu, prediksi tersebut sebaiknya digunakan sebagai indikator risiko OOM, sedangkan batas OOM aktual perlu diverifikasi melalui pengujian langsung atau pengukuran penggunaan RAM sistem.

Secara keseluruhan, analisis regresi memberikan tiga temuan utama. Pertama, NL menunjukkan hubungan linier yang sangat kuat antara jumlah pasangan observasi–blok dan waktu komputasi, dengan R^2 sebesar 0,99920 untuk TET dan 0,99945 untuk MCCT. Kedua, FV tidak menunjukkan hubungan linier yang kuat untuk waktu komputasi pada seluruh rentang eksperimen, dengan R^2 masing-masing sebesar 0,57859 untuk TET dan 0,59495 untuk MCCT. Penyimpangan tersebut terutama berkaitan dengan peningkatan waktu yang tajam pada konfigurasi model terbesar. Ketiga, kebutuhan memori kedua algoritma memiliki hubungan linier yang hampir sempurna terhadap jumlah pasangan observasi–blok, dengan R^2 mendekati 1.

Hasil tersebut memperlihatkan bahwa kebaruan FV tidak dapat dinilai hanya berdasarkan satu ukuran kinerja. Pada ukuran model kecil hingga menengah, transformasi dari NL menjadi FV memberikan keuntungan waktu komputasi yang signifikan. Akan tetapi, peningkatan tersebut disertai pertumbuhan penggunaan memori yang jauh lebih besar. Ketika ukuran model semakin besar, peningkatan kebutuhan memori mulai diikuti oleh penurunan keunggulan waktu komputasi Full Vectorization.

Dengan demikian, hasil regresi memperkuat kesimpulan bahwa Full Vectorization merupakan strategi yang efektif untuk mempercepat forward modeling gravitasi berbasis kernel Last–Kubik pada skala model yang masih sesuai dengan kapasitas memori komputer, tetapi memiliki keterbatasan skalabilitas akibat kebutuhan penyimpanan matriks yang besar. Sebaliknya, Nested Loop memiliki waktu komputasi yang lebih panjang, tetapi memberikan karakteristik penggunaan memori yang lebih terkendali dan perilaku waktu yang lebih konsisten terhadap peningkatan ukuran model.

Hasil analisis ini menjadi dasar penting bagi pembahasan selanjutnya mengenai batas penerapan algoritma, prediksi waktu komputasi, dan risiko OOM pada model resolusi tinggi. Secara khusus, regresi KWMI dapat digunakan untuk memperkirakan kebutuhan memori sebelum suatu konfigurasi model dijalankan, sehingga pengguna dapat menentukan strategi komputasi yang sesuai

dengan kapasitas perangkat keras yang tersedia.

5. KESIMPULAN

Artikel ini menunjukkan bahwa transformasi arsitektur komputasi dari Nested Loop (NL) menjadi Full Vectorization (FV) pada forward modeling gravitasi berbasis kernel Last–Kubik mampu mempertahankan hasil numerik sekaligus meningkatkan efisiensi komputasi. Kedua implementasi menghasilkan respon gravitasi yang identik dengan selisih absolut maksimum 0, sedangkan FV memberikan percepatan *mean core computational time* sekitar 6–7 kali pada model kecil hingga menengah.

Inovasi utama kajian ini terletak pada transformasi strategi evaluasi kernel dari perhitungan elemen-per-elemen menjadi operasi matriks secara simultan tanpa mengubah formulasi matematis Last–Kubik. Transformasi tersebut menghasilkan keuntungan waktu dengan konsekuensi peningkatan kebutuhan memori; KWMI untuk implementasi FV mencapai sekitar 18 kali KWMI untuk implementasi NL. Hasil ini menunjukkan bahwa efisiensi komputasi FV merupakan trade-off antara kecepatan atau waktu komputasi dan kapasitas memori.

Analisis regresi memperlihatkan bahwa kebutuhan memori memiliki hubungan linier yang sangat kuat terhadap jumlah pasangan observasi–blok, sedangkan keunggulan waktu FV mulai menurun pada ukuran model besar. Dengan demikian, studi ini menghasilkan strategi komputasi yang lebih efisien sekaligus memberikan dasar kuantitatif untuk menentukan batas penerapan FV berdasarkan ukuran model dan kapasitas memori, sehingga dapat menjadi landasan pengembangan komputasi geofisika beresolusi tinggi.

DAFTAR PUSTAKA

Grandis, H. (2009). Pengantar Pemodelan Inversi Geofisika. Jakarta: Himpunan Ahli Geofisika Indonesia (HAGI), pp. 45–104.

Hamimu, L. (2023). ForLoveDCs: A set of MATLAB function for calculating the forward modeling of multimode Love wave dispersion curves. Journal of Software Engineering and Simulation, 9, 129–140.

Hamimu, L., Safani, J., & Ngkoimani, L. O. (2015). Developed forward and inverse modelling of Vertical Electrical Sounding (VES) using MATLAB codes implementation. International Journal of Science and Research, 4(11), 1592–1597. https://doi.org/10.21275/v4i11.NOV151521

Hamimu, L., Cahyono, E., Budiman, H., Haraty, S. R., & Ransi, N. (2024a). Numerical analysis of the geophysical parameter sensitivity on the multimode Love wave dispersion curves. Journal of Physics Communications, 8, 015007. https://doi.org/10.1088/2399-6528/ad1f71

Hamimu, L., Haraty, S. R., Rubaiyn, A., Juarzan, L. O. I., & Indrawati. (2024b). Computation and modelling of seismic refraction tomography for anisotropic medium based on pseudo bending method. Journal of Physics: Conference Series, 2734(1), 012049. https://doi.org/10.1088/1742-6596/2734/1/012049

Hamimu, L., Aba, L., Muliddin, Cahyono, E., Juarzan, L. O. I., & Indrawati. (2025). Implementing a new inversion technique of geophysical data incorporating global optimization method based on systematic search, permutation, and sensitivity. IOP Conference Series: Earth and Environmental

Science, 1521(1), 012017. https://doi.org/10.1088/1755-1315/1521/1/012017

Last, B. J., & Kubik, K. (1983). Compact gravity inversion. Geophysics, 48(6), 713–721. https://doi.org/10.1190/1.1441501

Moskovka, A., Rahman, T., Valdman, J., & Vatne, J. E. (2026). Frameworking the vectorized basic linear algebra for prototyping codes in MATLAB. Applied Mathematics and Computation, 513, 129778. https://doi.org/10.1016/j.amc.2025.129778

Tchuigwa, B. S. S., Krmela, J., Pokorny, J., Krmelová, V., & Jilek, P. (2024). VectFEM: A generalized MATLAB-based vectorized algorithm for the computation of global matrix/force for finite elements of any type and approximation order in linear elasticity. Zeitschrift für Angewandte Mathematik und Physik, 75, 150. https://doi.org/10.1007/s00033-024-02293-w

Wang, W., Zheng, J., Ren, B., Yin, Z., Wan, W., Wang, Y., Gan, L., Zhang, Z., Zhang, W., Fu, H., & Chen, X. (2026). Accelerating three-dimensional seismic wave simulations on ARM using a hybrid half-precision and scalable vector extension approach. Geophysical Journal International, Volume 244, Page 1–12. https://doi.org/10.1093/gji/ggaf496

Lampiran A. Listing Program Nested Loop (NL)

```
%% Listing Program Nested Loop (NL)
clc; clear; close all
%% Konstanta G
G = 6.6732e-11; % universal gravitational
%% Kombinasi nx dan nz [nx nz]
kombinasi = [580 290];
%% Ukuran blok
dx = 50; % dimensi setiap blok arah horisontal
dh = 50; % dimensi setiap blok arah vertikal
d = dx; h = dh;
%% Jumlah benchmark
Nrepeat = 30;
%% Matriks hasil
hasil = zeros(size(kombinasi,1),6);
%% Loop kombinasi ukuran model

for k = 1:size(kombinasi,1)
   nx = kombinasi(k,1);
   nz = kombinasi(k,2);
   %% Inisialisasi model tubuh anomali geologi
   V = zeros(nz,nx);
   rho_anomaly = 1000;
   cx = round(nx/2); cz = round(nz/2);
   L = round(0.15*min(nx,nz));
   T = max(round(L/3),2);
   V(cz-L:cz+L,cx-T:cx+T)=rho_anomaly;
   V(cz-T:cz+T,cx-L:cx+L)=rho_anomaly;
   %% Koordinat pusat blok (cell-centered)
   x1 = 0:d:(nx-1)*d;
   x = x1+d/2;
   z1 = (0:h:(nz-1)*h)';
   z = z1+h/2;
   xx = repmat(x,nz,1);
   zz = repmat(z,1,nx);
   %% Titik observasi (di atas pusat blok)
   x_obs = x;
   Nobs = length(x_obs);
   nb = nx*nz;
   %% MEMORY AWAL
   S0 = whos;
   BWM = sum([S0.bytes]);
%% Starting Benchmark
t = zeros(Nrepeat,1);
 for r=1:Nrepeat
    tic
    disp('NREPEAT='); disp(r)
    AA = zeros(Nobs,nb);
    for i=1:Nobs
       for j=1:nb
          %% Pengulangan komputasi komponen jarak
          r1 = sqrt((zz(j)-h/2).^2 + (x_obs(i)-xx(j)+d/2).^2);
          r2 = sqrt((zz(j)+h/2).^2 + (x_obs(i)-xx(j)+d/2).^2);
          r3 = sqrt((zz(j)-h/2).^2 + (x_obs(i)-xx(j)-d/2).^2);
          r4 = sqrt((zz(j)+h/2).^2 + (x_obs(i)-xx(j)-d/2).^2);
          %% Pengulangan komputasi komponen sudut
          theta1 = atan((x_obs(i)-xx(j)+d/2)/(zz(j)-h/2));
          theta2 = atan((x_obs(i)-xx(j)+d/2)/(zz(j)+h/2));
          theta3 = atan((x_obs(i)-xx(j)-d/2)/(zz(j)-h/2));
          theta4 = atan((x_obs(i)-xx(j)-d/2)/(zz(j)+h/2));
          %% Pengulangan komputasi respon gravitasi
          AA(i,j)=2*G*((x_obs(i)-xx(j)+d/2)*log((r2*r3)/...
              (r1*r4)) + d*log(r4/r3) - (zz(j)+h/2)*...
              (theta4-theta2)+ (zz(j)-h/2)*(theta3-theta1));
       end
    end
    E=0; % noise-free
    GE = (AA*V(:)+E); % Gravity Effect
    % konversi ke mGal
    GE = GE*1e5;
    t(r)=toc;
end
%% MEMORY AKHIR
S1 = whos;
PCWM = sum([S1.bytes]);
WMI = PCWM - BWM;
WMI_MB = WMI/1024^2;
hasil(k,:)=[nx nz nb median(t) mean(t) WMI_MB];
fprintf(['%3d x %3d | blok=%6d | median=%8.5f s | ',...
      'mean=%8.5f s | WMI_MB=%8.2f MB\n'],...
      nx,nz,nb,median(t),mean(t),WMI_MB);
end
%% Plot Respon gravitasi
figure1 = figure('NumberTitle','off','Name',...
   'Komputasi Forward Modelling Gravity');
sp1=[0.1, 0.55, 0.85, 0.4];
axes1 = axes('Parent',figure1,'Position',sp1);
subplot('Position',sp1)
plot(x_obs,GE,'.r');
title('Implementasi Nested Loop', 'fontweight','bold',...
   'fontsize',10)
ylabel('Efek Gravitasi [mGal]');
%% Visualisasi model anomali
sp2=[0.1, 0.175, 0.85, 0.3];
subplot('Position',sp2)
imagesc(x_obs,z,V);hold on;
xlabel('Jarak (m)'); ylabel('Depth [m]');
colormap('cool');
colorbar('horiz','Position',[0.1 0.05 0.85 0.025],...
   'Ticks',[min(V(:)) 0.5*max(V(:)) max(V(:))]);
grid on
```

Listing Program Nested Loop (NL)

Lampiran B. Listing Program Full Vectorizatin (FV)

```
%% Listing Program Full Vectorization (FV)
clc; clear; close all
%% Konstanta G
G = 6.6732e-11; % universal gravitational
%% Kombinasi nx dan nz [nx nz]
kombinasi = [580 290];
%% Ukuran blok
dx = 50; % dimensi setiap blok arah horisontal
dh = 50; % dimensi setiap blok arah vertikal
d = dx; h = dh;
%% Jumlah benchmark
Nrepeat = 30;
%% Matriks hasil
hasil = zeros(size(kombinasi,1),6);
%% Loop kombinasi ukuran model
for k = 1:size(kombinasi,1)
   nx = kombinasi(k,1);
   nz = kombinasi(k,2);
   %% Model densitas
   V = zeros(nz,nx);
   rho = 1000;
   cx = round(nx/2); cz = round(nz/2);
   L = round(0.15*min(nx,nz));
   T = max(round(L/3),2);
   %% lengan vertikal anomali +
   V(cz-L:cz+L,cx-T:cx+T)=rho;
   %% lengan horizontal anomali +
   V(cz-T:cz+T,cx-L:cx+L)=rho;
   %% Grid observasi
   x = (0:nx-1)*d + d/2; z = ((0:nz-1)*h)+h/2;
   [X,Z] = meshgrid(x,z);
   X = X(:); Z = Z(:);
   nb = length(X);
   x_obs = x;
   Nobs = length(x_obs);
   XO = x_obs(:);
   %% Matriks koordinat
   DX = bsxfun(@minus,XO,X);
   ZZ = repmat(Z,1,Nobs);
   %% Memory awal
   S0 = whos;
   BWM = sum([S0.bytes]);
   %% Komputasi empat komponen jarak
   ZH1 = ZZ - h/2; ZH2 = ZZ + h/2;
   DX1 = DX + d/2; DX2 = DX - d/2;
   for i=1:Nrepeat
      tic
      disp('NREPEAT='); disp(i)
      r1 = sqrt(ZH1.^2 + DX1.^2);
      r2 = sqrt(ZH2.^2 + DX1.^2);
      r3 = sqrt(ZH1.^2 + DX2.^2);
      r4 = sqrt(ZH2.^2 + DX2.^2);
      %% Komputasi empat komponen sudut
      theta1 = atan(DX1./ZH1);
      theta2 = atan(DX1./ZH2);
      theta3 = atan(DX2./ZH1);
      theta4 = atan(DX2./ZH2);
      %% Komputasi Kernel Last-Kubik
      term1 = DX1 .* log((r2 .* r3) ./(r1 .* r4));
      term2 = d .* log(r4 ./ r3);
      term3 = -(ZH2) .* (theta4 - theta2);
      term4 = (ZH1) .* (theta3 - theta1);
      %% Kernel Last-Kubik
      A = 2 * G * (term1 + term2 + term3 + term4);
      %% Membentuk Kernel dan Gravity Response
      AA = A';
      GE = AA * V(:);
      %% Konversi ke mGal
      GE = GE * 1e5;
      t(r)=toc;
      %% Memory setelah kernel terbentuk
      S1 = whos;
      PCWM = sum([S1.bytes]);
      WMI = PCWM - BWM;
      WMI_MB = WMI/1024^2;
   end
   hasil(k,:)= [nx nz nb median(t) mean(t) WMI_MB];
   fprintf(['%3d x %3d | blok=%6d | median=%8.5f s | ',...
       'mean=%8.5f s | WMI_MB=%8.2f MB \n']',...
       nx,nz,nb,median(t),mean(t),WMI_MB);
end
%% Plot Respon gravitasi
figure1 = figure('NumberTitle','off','Name',...
    'Komputasi Forward Modelling Gravity');
sp1=[0.1, 0.55, 0.85, 0.4];
axes1 = axes('Parent',figure1,'Position',sp1);
subplot('Position',sp1)
plot(x_obs,GE,'.r');
title('Implementasi Full Vectorization', 'fontweight','bold',...
    'fontsize',10)
ylabel('Efek Gravitasi [mGal]');
%% Visualisasi model anomali
sp2=[0.1, 0.175, 0.85, 0.3];
subplot('Position',sp2)
imagesc(x_obs,z,V);hold on;
xlabel('Jarak (m)'); ylabel('Depth [m]');
colormap('cool');
colorbar('horiz','Position',[0.1 0.05 0.85 0.025],...
    'Ticks',[min(V(:)) 0.5*max(V(:)) max(V(:))]);
grid on
```

Listing Program Full Vectorization (FV)