JIKO (JURNAL INFORMATIKA DAN KOMPUTER)
Februari 2025, Volume: **9**, No. **1** | Pages 140–148
doi: 10.26798/jiko.v9i1.1418
e-ISSN : 2477-3964 – p-ISSN : 2477-4413

ARTICLE

Optimasi Unjuk Kerja Memory dalam Bahasa Dart pada Aplikasi Monitoring Pegawai

Memory Performance Optimization in Dart Language, in the Employee Monitoring Application

Indra Listiawan,* Farida Nur Aini, Zaidir, dan Ahmad Sahal

Teknologi Informasi Fakultas Sains dan Teknologi Informasi, Universitas Respati, Yogyakarta, Indonesia
*Penulis Korespondensi: indra@respati.ac.id

(Disubmit 10-08-24; Diterima 22-10-24; Dipublikasikan online pada 05-02-25)

Abstrak
Penelitian ini mengeksplorasi dan menganalisis teknik optimisasi kinerja memori dalam bahasa pemrograman Dart, khususnya dalam konteks aplikasi pemantauan karyawan. Mengingat pentingnya pengelolaan memori yang efisien dalam pengembangan aplikasi real-time, penelitian ini berfokus pada penggunaan keyword const untuk meningkatkan performa aplikasi. Hasil eksperimen menunjukkan bahwa penggunaan const menghasilkan waktu proses yang lebih cepat tanpa mengurangi penggunaan memori. Analisis lebih lanjut mengungkapkan bahwa const memungkinkan optimisasi pada waktu kompilasi yang meningkatkan kecepatan eksekusi dengan memastikan objek tidak dapat diubah setelah dibuat. Namun, const tidak mempengaruhi jumlah memori yang dialokasikan untuk objek, menunjukkan bahwa ukuran memori tetap sama terlepas dari penggunaannya. Penelitian ini memberikan wawasan penting tentang bagaimana const dapat digunakan untuk mengoptimalkan performa aplikasi Dart dalam hal kecepatan, meskipun tidak berdampak signifikan pada penggunaan memori. Temuan ini diharapkan dapat memberikan kontribusi signifikan dalam pengembangan aplikasi Dart yang lebih efisien dan responsif, serta memperkaya literatur akademik terkait optimisasi kinerja memori dalam bahasa pemrograman Dart.

Kata kunci: memori; efisiensi; const ; Dart

Abstract
This study explores and analyzes memory performance optimization techniques in the Dart programming language, particularly in the context of employee monitoring applications. Given the importance of efficient memory management in real-time application development, this research focuses on the use of the const keyword to enhance application performance. Experimental results indicate that using const results in faster processing times without reducing memory usage. Further analysis reveals that const allows for compile-time optimizations that improve execution speed by ensuring that objects cannot be modified after creation. However, const does not affect the amount of memory allocated to objects, indicating that memory size remains the same regardless of its use. This study provides important insights into how const can be used to optimize Dart application performance in terms of speed, although it does not significantly impact memory usage. These findings are expected to contribute significantly to the development of more efficient and responsive Dart applications, as well as enrich the academic literature related to memory performance optimization in the Dart programming language.

KeyWords: memory; eficiency; const ; Dart

How to Cite: I. Listiawan *et al.*, "Optimasi Unjuk Kerja Memory dalam Bahasa Dart pada Aplikasi Monitoring Pegawai", *JIKO (JURNAL INFORMATIKA DAN KOMPUTER)*, Volume: **9**, No.**1**, Pages 140–148, Februari 2025, doi: 10.26798/jiko.v9i1.1418.

1. Pendahuluan

Dalam konteks aplikasi pemantauan karyawan, yang seringkali harus menangani data dalam jumlah besar secara real-time, optimisasi kinerja memori menjadi sangat penting[1]. Penggunaan memori yang efisien dapat meningkatkan kecepatan respons aplikasi, mengurangi konsumsi daya, dan meningkatkan pengalaman pengguna secara keseluruhan[2]. Namun, penelitian dan literatur yang membahas optimisasi kinerja memori dalam bahasa Dart masih relatif terbatas[3].

Beberapa studi terdahulu telah menyoroti pentingnya optimisasi memori dalam pengembangan aplikasi[4]. Sebuah penelitian yang menekankan bahwa optimisasi memori dapat secara signifikan mengurangi waktu respons aplikasi dan meningkatkan efisiensi secara keseluruhan[5]. Selain itu, sebuah survey menunjukkan bahwa efisiensi memori adalah salah satu faktor kunci dalam keberhasilan pengembangan aplikasi mobile, yang mencakup manajemen sumber daya yang lebih baik dan pengurangan lag aplikasi[6].

Pengalaman pengguna meningkat dengan optimisasi memori komputer melalui berbagai teknik yang diusulkan dalam makalah penelitian. Ice, sebuah kerangka kerja untuk perangkat mobile dengan sumber daya terbatas, mengidentifikasi dan membekukan proses yang menyebabkan kesalahan berulang, sehingga meningkatkan pengalaman pengguna dengan meningkatkan frame rate secara signifikan[7].

SEAL memperkenalkan mekanisme swapping dua tingkat yang memperhatikan pengalaman pengguna, yang memaksimalkan manfaat swapping memori sambil meminimalkan dampak negatif pada interaksi, menghasilkan peningkatan 2,43x dalam kemampuan caching aplikasi tanpa mengorbankan pengalaman pengguna[8]. Selain itu, penjadwal memori di ruang pengguna mengalokasikan node memori ideal berdasarkan karakteristik arsitektur memori non-uniform, menghasilkan peningkatan kinerja hingga 25% untuk aplikasi[9]. Lebih jauh lagi, teknik optimisasi seperti kompresi data dan menghindari pola akses ganda dalam akses memori mengurangi latensi, kebutuhan penyimpanan, dan permintaan akses memori, yang pada akhirnya meningkatkan pengalaman pengguna dalam analisis lalu lintas jaringan dan simulasi jaringan saraf skala besar[10].

Tujuan utama dari penelitian dalam mengoptimalkan kinerja memori dalam Dart meliputi pengenalan teknik untuk optimisasi memori dalam bahasa array fungsional untuk eksekusi GPU[11], mengusulkan kerangka kerja untuk menentukan tingkat perkiraan memori dalam hierarki untuk aplikasi yang tahan terhadap kesalahan, dengan tujuan mengoptimalkan konfigurasi memori untuk efisiensi energi[12], memperkenalkan metode untuk membagi frame tumpukan di beberapa unit memori untuk alokasi halus variabel memori otomatis dalam sistem embedded real-time[13], mempertimbangkan penentuan otomatis subsistem memori spesifik aplikasi melalui superoptimasi untuk mengurangi waktu akses memori dan meminimalkan penulisan[14], serta mengatasi masalah mendasar dalam bottleneck bandwidth memori melalui perspektif tingkat sistem untuk memberikan kinerja bandwidth tinggi tanpa mengubah semantik panggilan sistem operasi. Tujuan-tujuan ini secara kolektif bertujuan untuk meningkatkan pemanfaatan memori, kinerja, dan efisiensi energi dalam aplikasi Dart.

Studi ini mengeksplorasi berbagai teknik untuk optimasi kinerja memori dalam berbagai bahasa pemrograman. Salah satu teknik berfokus pada pengenalan dan optimasi penggunaan memori dalam bahasa array fungsional untuk eksekusi GPU, mendukung paralelisme yang benar dengan konstruksi dan memungkinkan transformasi tata letak tanpa manifestasi memori. Teknik lain melibatkan optimasi kinerja Apache Spark dengan menyesuaikan rasio kapasitas ruang shuffle dan storage serta menerapkan caching memori yang didukung SSD, yang menghasilkan peningkatan kinerja hingga 42%[15]. Selain itu, sebuah kerangka kerja diusulkan untuk menentukan tingkat perkiraan memori dalam hierarki, mengoptimalkan konfigurasi memori untuk aplikasi yang tahan terhadap kesalahan dan mencapai perbaikan konsumsi daya yang signifikan pada cache SRAM dan memori DRAM. Terakhir, sebuah metodologi dipresentasikan untuk menerapkan memoization dalam aplikasi, mengotomatisasi analisis dan transformasi kode untuk meningkatkan kinerja dengan upaya minimal[16].

Manajemen memori yang efisien sangat penting dalam pengembangan aplikasi Dart untuk mengoptimalkan kinerja dan pemanfaatan sumber daya. Berbagai pendekatan, seperti menetapkan token anonim pada objek[17], mengalokasikan segmen memori berdasarkan jenis pemuat kelas[18], dan menerapkan manajer memori dinamis terdistribusi untuk sistem multi-core[19], memainkan peran penting dalam meningkatkan

efisiensi memori. Dalam bahasa pemrograman konkuren seperti Dart, di mana komunikasi terjadi melalui pengiriman pesan, skema manajemen memori dengan heap lokal proses dan pengumpulan sampah bertahap sangat penting untuk meminimalkan overhead sinkronisasi dan memastikan pemulihan memori tepat waktu[20]. Dengan mengadopsi strategi-strategi ini, pengembang Dart dapat meningkatkan responsivitas dan skalabilitas aplikasi mereka sambil mengelola sumber daya memori secara efisien untuk memenuhi tuntutan lingkungan pengembangan perangkat lunak modern.

2. Metode

Dalam bahasa Dart, field statis diinisialisasi secara lambat. Artinya, saat pertama kali aplikasi Flutter dijalankan dan sistem akan membaca sebuah field statis, field tersebut akan diatur sesuai dengan nilai yang ditentukan oleh inisialisatornya. Variabel global dan field statis dianggap sebagai bagian dari status aplikasi, sehingga tidak diinisialisasi ulang selama hot reload. Jika diakukan perubahan pada inisialisator variabel global dan field statis, maka perlu dilakukan hot restart atau restart status di mana inisialisator tersebut berada untuk melihat perubahan tersebut. Sementara perubahan pada nilai field const selalu hot reloaded, inisialisator field statis tidak dijalankan ulang. Secara konseptual, field const diperlakukan seperti alias daripada status.

Mesin Virtual akan mendeteksi perubahan inisialisator dan memberi tanda ketika satu set perubahan memerlukan hot restart agar berlaku. Mekanisme pemberian tanda dipicu untuk sebagian besar pekerjaan inisialisasi. Untuk memperbarui dan melihat perubahan nilai sebuah field setelah hot reload, maka perlu mendefinisikan ulang field sebagai const atau menggunakan getter untuk mengembalikan nilai.

Liu dkk, mempublikasikan penelitiannya tentang model optimasi dan efisiensi memory system. Berdasarkan penelitiannya dapat dikembangkan model matematis penggunaan memory dan efisiensi dengan mengadopsi pendekatan yang digunakan dalam artikel tersebut[21]. Model ini menggunakan tiga metrik utama: *Concurrent Average Memory Access Time* (C-AMAT), *Accesses Per Cycle* (APC), dan *Layered Performance Matching* (LPM).

Definisi Variabel

1. *Concurrent Average Memory Access Time* (C-AMAT):

$$C - AMAT = \frac{\omega}{\alpha} \tag{1}$$

   dimana ω adalah jumlah siklus aktif memori, dan α adalah jumlah akses memori
2. *Accesses Per Cycle* (APC):

$$APC = \frac{\alpha}{\omega} = \frac{1}{C - AMAT} \tag{2}$$

   Mengukur jumlah akses memori per siklus memori aktif
3. *Layered Performance Matching* (LPM):

$$LPMR(l) = \frac{\lambda(l)}{v(l)} \tag{3}$$

   di mana $\lambda(l)$ adalah laju permintaan pada level cache l, dan $v(l)$ adalah laju pasokan pada level cache l.

Model Matematis Rumus C-AMAT diperoleh dari persamaan berikut:

$$C - AMAT = \frac{H(l)}{CH(l)} + \rho_m(l) \times \kappa l \times C - AMAT(l + 1) \tag{4}$$

dimana:
$H(l)$ adalah waktu hit pada level cache l.
$CH(l)$ adalah concurrency hit pada level cache l.
$\rho m(l)$ adalah rasio miss pada level cache l.
κl adalah rasio pure miss pada level cache l.

Dengan rumus efisiensi memori:

$$MSE = \frac{1}{1 + \Delta} \tag{5}$$

dimana:

$$\Delta = \frac{MST - f_{mem}}{CPI_{exe}} \tag{6}$$

Dan rumus *Layered Performance Matching* (LPM):

$$LMPR(l) = \frac{\lambda(l)}{\nu(l)} \tag{7}$$

dimana:

$$\lambda(l) = \frac{\alpha(l)}{IC \times CPI_{exe}} \tag{8}$$

dan

$$\nu(l) = \frac{\alpha(l)}{\omega(l)} \tag{9}$$

Tahapan Penelitian:

1. Membuat dataset yang terdiri dari objek-objek karyawan dengan atribut yang relevan. Untuk membuat dataset dilakukan dengan menggunakan metode List.generate untuk membuat dataset dengan ukuran besar (misalnya 10.000 karyawan).
2. Implementasi Program dengan const. Implementasikan program Dart dengan menggunakan kata kunci const pada objek yang sesuai.dalam hal ini kata kunci const ditambahkan pada deklarasi objek dan konstruktor.
3. Implementasi Program tanpa const. Implementasikan program Dart tanpa menggunakan kata kunci const. Imlementasinya dengan cara menghapus kata kunci const dari deklarasi objek dan konstruktor.
4. Pengukuran Penggunaan Memori. Profiling penggunaan memori sebelum dan sesudah optimisasi. Profiling dilakukan dengan cara menggunakan fungsi calculateMemoryUsage untuk menghitung penggunaan memori berdasarkan jumlah objek dan ukuran estimasi.
5. Pengukuran Waktu Eksekusi. Mengukur waktu eksekusi untuk proses optimisasi data. Pengukuran waktu menggunakan fungsi Stopwatch untuk mengukur durasi eksekusi metode optimisasi.
6. Pengujian dan Pengumpulan Data. Tahap ini dilakukan dengan menjalankan kedua program (dengan dan tanpa const) dan mencatat hasil penggunaan memori dan waktu eksekusi. Pencatatan menggunakan logging dan pencatatan hasil dalam file JSON untuk analisis lebih lanjut.
7. Analisis Hasil. Analisa hasil dilakukan dengan membandingkan hasil penggunaan memori dan waktu eksekusi dari kedua pendekatan.Hasil divisualisasikan dalam bentuk grafik untuk memudahkan perbandingan.
8. Kesimpulan berdasarkan hasil analisis data, diambil kesimpulan manfaat atau kekurangan penggunaan const dalam konteks yang diuji.

Berdasarkan tahapan peneltian, dirumuskan algoritmanya sebagai berikut:

1. Inisialisasi Data Karyawan.
   (a) Inisialisasi dataset karyawan besar dengan berbagai atribut.
   (b) Simpan data ini dalam list employees
2. Konstruktor Aplikasi Pemantauan Karyawan.
   (a) Terima list employees sebagai parameter.
   (b) Inisialisasi atribut initialMemoryUsage, optimizedMemoryUsage, dan optimizationExecutionTime dengan nilai awal 0.

3. Profiling Penggunaan Memori.

   (a) Hitung dan catat penggunaan memori saat ini dengan metode calculateMemoryUsage.
   (b) Kembalikan nilai penggunaan memori ini.

4. Optimisasi Penggunaan Memori.

   (a) Buat list baru uniqueEmployees dari dataset employees dengan menghilangkan duplikasi.
   (b) Untuk setiap karyawan dalam uniqueEmployees, panggil metode optimizeData untuk mengoptimalkan data mereka.

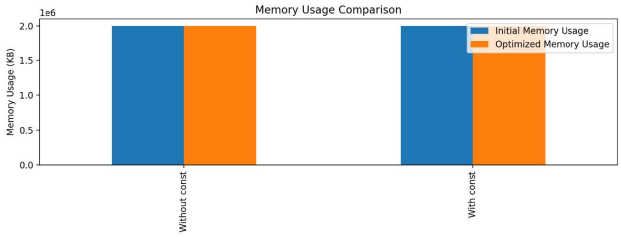
5. Pengujian Kinerja.

   (a) Profiling penggunaan memori awal dan simpan hasilnya dalam initialMemoryUsage. 2. Mulai stopwatch untuk mengukur waktu eksekusi optimisasi. 3. Panggil metode optimizeMemoryUsage. 4. Hentikan stopwatch dan catat waktu eksekusi dalam optimizationExecutionTime. 5. Profiling penggunaan memori setelah optimisasi dan simpan hasilnya dalam optimizedMemoryUsage

6. Cetak Hasil
   Cetak hasil penggunaan memori awal, penggunaan memori setelah optimisasi, dan waktu eksekusi optimisasi

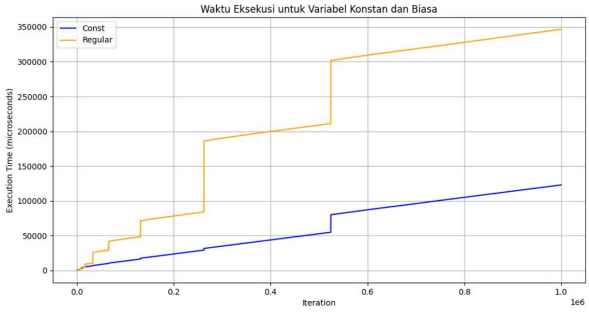
3. Hasil Penelitian

Program melakukan beberapa komputasi yaitu pengembangan dataset secara automatis, melakukan operasi optimasi memori, menghitung penggunaan memori awal dan memori setelah proses selesai serta menghitung kecepatan proses. Agar dapat membandingkan hasil dari komputasi maka komputasi dilakukan dua kali yang pertama tanpa melibatkan keyword const dan komputasi kedua yang melibatkan keyword const. Hasilnya direpresentasikan dengan grafik 2 jenis grafik yaitu grafik penggunaan memori dan grafik perhitungan waktu proses Berikut ini grafik perbandingan penggunaan memori.



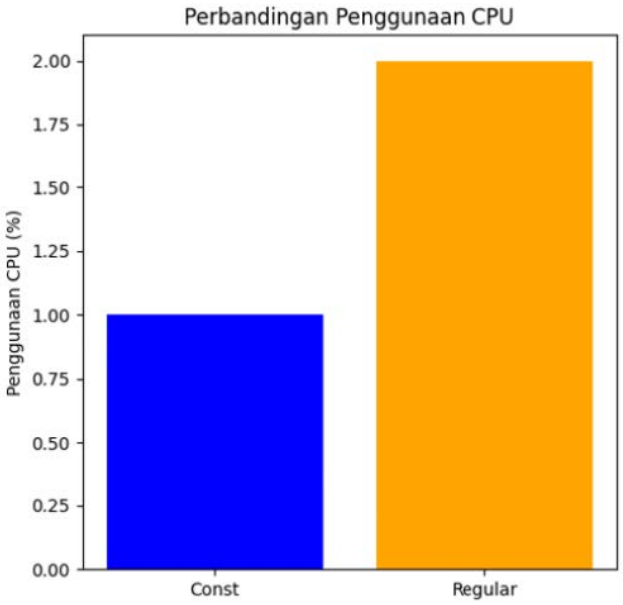
Gambar 1. Perbandingan Penggunaan Memori Proses Komputasi Dengan dan Tanpa Mengunakan Keyword const

Tampak bahwa grafik penggunaan memory pada proses komputasi yang menggunakan keyword const maupun yang tidak menggunakan memiliki nilai yang sama. Sedangkan Gambar 2 menunjukkan perbandingan grafik penggunaan waktu komputasi



Gambar 2. Perbandingan Waktu Komputasi Dengan dan Tanpa Keyword const

Pada Gambar 2 terlihat ada perbedaan waktu proses yang cukup signifikan antara proses yang menggunakan keyword cost (garis biru) dan yang tanpa menggunakan keyword const (garis kuning). Menurut penelitian Zhang dkk, bahwa semakin meningkat efisiensi memori, maka semakin berkurang beban kerja CPU atau penggunan CPU berkurang[22]. Penelitian ini melakukan komputasi dan menghitung penggunaan CPU (CPU Usage) saat komputasi program dilakukan dengan penggunaan keyword const dan tanpa menggunakan keyword const. Pada gambar 3 grafik batang terlihat bahwa penggunaan atau beban CPU berkurang saat dilakukan komputasi menggunakan keyword const (batang biru)



Gambar 3. Perbandingan Penggunaan CPU Dengan dan Tanpa Penggunaan Keyword const

4. Pembahasan

Penggunaan keyword const dalam eksperimen menunjukkan peningkatan kecepatan proses, namun tidak berdampak pada penggunaan memori. Hal ini memberikan wawasan penting mengenai optimisasi performa dalam konteks penggunaan const pada pemrograman Dart.

Kecepatan

1. Optimisasi Kompilasi:
   Penggunaan const memastikan bahwa objek tidak dapat diubah setelah dibuat. Ini memungkinkan Dart melakukan optimisasi pada waktu kompilasi, seperti penghapusan kode redundan atau penyederhanaan instruksi. Hal ini mengurangi overhead pada saat eksekusi karena Dart tidak perlu menangani kemungkinan perubahan state dari objek tersebut.
2. Eksekusi Lebih Cepat:
   Mengacu pada gambar 2 terdapat perbedaan kecepatan komputasi yang signifikan antara komputasi dengan penggunaan keyword const. Disertakannya keyword const, Dart dapat membuat asumsi yang lebih kuat tentang state program, yang memungkinkan operasi pada objek const dieksekusi lebih cepat. Contohnya, akses ke variabel const bisa lebih cepat dibandingkan akses ke variabel yang tidak const karena Dart tahu bahwa nilai tersebut tidak akan berubah dan bisa di-cache lebih efektif.
3. Efisiensi Penggunaan CPU:
   Mengacu pada gambar 3 adanya pengurangan beban kerja pada CPU terjadi karena tidak ada perubahan state yang harus dipantau atau di-update. Ini berarti pengalokasian dan dealokasi memori yang lebih efisien serta pengurangan instruksi-instruksi yang tidak perlu, yang secara kumulatif meningkatkan kecepatan eksekusi aplikasi.

Memori

1. Alokasi Memori
   Penggunaan const tidak mempengaruhi jumlah memori yang dialokasikan untuk objek. Setiap objek Employee dan data terkaitnya tetap memerlukan ruang memori yang sama, baik objek tersebut const atau tidak. Const hanya menjamin bahwa data tidak dapat diubah setelah diinisialisasi, bukan mengurangi ukuran data itu sendiri.
2. Fragmentasi Memori
   Tidak ada pengurangan fragmentasi memori dengan penggunaan const. Fragmentasi memori biasanya terjadi karena alokasi dan dealokasi objek dinamis, dan const tidak mempengaruhi proses ini. Karena itu, total penggunaan memori tetap sama.
3. Penggunaan Memori Konstan
   Dalam skenario aplikasi pemantauan karyawan, objek-objek Employee memiliki ukuran tetap yang tidak dipengaruhi oleh apakah mereka const atau tidak. Konstanta hanya menjamin integritas data, tetapi tidak mengubah jejak memori dari objek tersebut.
4. Pengumpulan Sampah (G*arbage Collections*)
   Const tidak mempengaruhi cara pengumpulan sampah bekerja dalam Dart. Memori yang dialokasikan untuk objek const masih tunduk pada mekanisme pengumpulan sampah yang sama seperti objek non-const. Karena itu, tidak ada pengurangan signifikan dalam penggunaan memori yang dapat diatributkan langsung kepada const.

5. Simpulan

Berdasarkan analisis hasil eksperimen, penggunaan keyword const dalam pemrograman Dart menghasilkan waktu proses yang lebih cepat namun tidak mempengaruhi penggunaan memori secara signifikan.

Kecepatan: Penggunaan const memastikan bahwa objek yang dibuat tidak dapat diubah setelah dibuat, yang memungkinkan Dart untuk melakukan optimisasi pada waktu kompilasi. Dengan demikian, operasi yang melibatkan objek const dapat dieksekusi lebih cepat karena Dart mengetahui bahwa objek tersebut tidak akan mengalami perubahan.

Memori: Meskipun const memastikan bahwa objek tidak dapat diubah, hal ini tidak mengurangi jumlah memori yang digunakan oleh aplikasi secara keseluruhan. Setiap objek Employee dan data terkaitnya tetap membutuhkan ruang memori yang sama terlepas dari apakah objek tersebut const atau tidak. Ini menunjukkan bahwa const lebih berpengaruh pada optimisasi performa dalam hal kecepatan eksekusi daripada pada pengurangan penggunaan memori. Dengan demikian, penelitian ini menyimpulkan bahwa penggunaan keyword const dalam pemrograman Dart dapat meningkatkan kecepatan eksekusi aplikasi tanpa memberikan dampak signifikan terhadap efisiensi penggunaan memori. Ini memberikan wawasan penting bagi pengembang aplikasi untuk mengoptimalkan performa aplikasi mereka dengan mempertimbangkan penggunaan const untuk operasi yang membutuhkan kecepatan tinggi.

Hal menarik untuk melanjutkan penelitian ini adalah sejauh mana akses modifier terhadap variabel yang menggunakan const berpengaruh terhadap kinerja memory maupun CPU. Dengan mengetahui pengaruhnya, maka saat status variabel sebagai public maupun sebagai private menjadi pertimbangan tersendiri saat mengimplementasikannya

Sumber dana

Sumber dana penelitian ini berasal dari Universitas Respati Yogyakarta.

Pustaka

[1] A. Fatkharrofiqi, F. W. Handono, and H. Nurdin, “Employee attendance application using location based service (lbs) method based on android,” *J. Phys.*, 2020.

[2] A. Maryensyah, I. Kanedi, and R. Zulfiandry, "Academic administration information system for junior high school (smp) pgri bengkulu city," *J. Komput. Inf. Dan Teknol.*, vol. 1, no. 1, pp. 84–93, Jun. 2021.

[3] G. W. Wiriasto, R. W. S. Aji, and D. F. Budiman, "Design and development of attendance system application using android-based flutter," in *2020 Third International Conference on Vocational Education and Electrical Engineering (ICVEE)*, Oct. 2020, pp. 1–6.

[4] A. M. Hassan, "Java and dart programming languages: conceptual comparison," *Indones. J. Electr. Eng. Comput. Sci.*, vol. 17, no. 2, p. 845, Feb. 2020.

[5] H. Shu, A. Wang, Z. Shi, H. Zhao, Y. Li, and L. Lu, "Roam: memory-efficient large dnn training via optimized operator ordering and memory layout," *arXiv*, Oct. 30 2023, arXiv:2310.19295. [Online]. Available: http://arxiv.org/abs/2310.19295

[6] A. Kumar, V. Seshadri, and R. Sharma, "Shiftry: Rnn inference in 2kb of ram," *Proc. ACM Program. Lang.*, vol. 4, no. OOPSLA, pp. 1–30, 2020.

[7] C. Li, Y. Liang, R. Ausavarungnirun, Z. Zhu, L. Shi, and C. J. Xue, "Ice: Collaborating memory and process management for user experience on resource-limited mobile devices," in *Proceedings of the Eighteenth European Conference on Computer Systems (EuroSys '23).* Association for Computing Machinery, 2023, pp. 79–93.

[8] C. Li, L. Shi, Y. Liang, and C. J. Xue, "Seal: User experience-aware two-level swap for mobile devices," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 39, no. 11, pp. 4102–4114, 2020.

[9] G. Lim and S.-B. Suh, "User-level memory scheduler for optimizing application performance in numa-based multicore systems," in *2014 IEEE 5th International Conference on Software Engineering and Service Science*, 2014, pp. 240–243.

[10] Y. Yang, D. Stathis, R. Jordão, A. Hemani, and A. Lansner, "Optimizing bcpnn learning rule for memory access," *Front. Neurosci.*, vol. 14, p. 878, 2020.

[11] P. Munksgaard, T. Henriksen, P. Sadayappan, and C. Oancea, "Memory optimizations in an array language," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2022, pp. 1–15.

[12] R. Yarmand, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Dart: A framework for determining approximation levels in an approximable memory hierarchy," *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 28, no. 1, pp. 273–286, 2020.

[13] S. Kang and A. G. Dean, "Darts: Techniques and tools for predictably fast memory using integrated data allocation and real-time task scheduling," in *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2010, pp. 333–342.

[14] J. G. Wingbermuehle, R. K. Cytron, and R. D. Chamberlain, "Superoptimizing memory subsystems for multiple objectives," in *Euro-Par 2015: Parallel Processing Workshops*, S. Hunold, A. Costan, D. Giménez, A. Iosup, L. Ricci, M. E. G. Requena, V. Scarano, A. L. Varbanescu, S. L. Scott, S. Lankes, J. Weidendorfer, and M. Alexander, Eds. Springer International Publishing, 2015, pp. 352–363.

[15] J. Choi, J. Lee, J.-S. Kim, and J. Lee, "Optimization techniques for a distributed in-memory computing platform by leveraging ssd," *Appl. Sci.*, vol. 11, no. 18, p. 8476, 2021.

[16] P. Pinto and J. M. P. Cardoso, "A methodology and framework for software memoization of functions," in *Proceedings of the 18th ACM International Conference on Computing Frontiers.* ACM, 2021, pp. 93–101.

[17] C. F. Russ, "System and method for using anonymous tokens for efficient memory management," 2001, patent.

[18] K. K. et al., "Efficient class memory management," 2005, patent.

[19] M. M. Jaafar and A. H. Obaid, "An efficient memory management in single and multi-core embedded system using global shared memory," in *AIP Conf. Proc.*, vol. 3051, no. 1, 2024, p. 040004.

[20] K. S. et al., "Efficient memory management for concurrent programs that use," 2006, patent.

[21] J. Liu, P. Espina, and X.-H. Sun, "A study on modeling and optimization of memory systems," *J. Comput. Sci. Technol.*, vol. 36, no. 1, pp. 71–89, 2021.

[22] K. Zhang, D. Ou, C. Jiang, Y. Qiu, and L. Yan, "Power and performance evaluation of memory-intensive applications," *Energies*, vol. 14, no. 14, p. 4089, 2021.