ISSN : 2407 - 3911

EVALUASI PENGARUH FUNGSI PEMETAAN TERHADAP KINERJA DAN KONSUMSI DAYA *CACHE MEMORY*

Tati Erlina[1)], Rahmi Eka Putri[2)]
Jurusan Sistem Komputer Fakultas Teknologi Informasi
Universitas Andalas
Kampus Limau Manis Unand, PADANG
tatierlina@fti.unand.ac.id[1)], rahmi230784@gmail.com[2)]

Abstrak

Menciptakan komputer yang berkinerja tinggi dan mengkonsumsi daya minimal merupakan salah satu fokus yang menjadi tren penelitian dalam bidang arsitektur komputer belakangan ini. Diketahui bahwa sejumlah faktor berpengaruh dalam menentukan kinerja dan konsumsi daya dari setiap elemen pembangun komputer dan pada akhirnya mempengaruhi kinerja dan konsumsi daya sebuah komputer secara keseluruhan. Fokus penelitian ini adalah untuk mengetahui seberapa besar pengaruh dari fungsi pemetaan (*mapping techniques*) sebagai salah satu jenis elemen perancangan yang diterapkan pada sebuah *cache memory* terhadap kinerja dan konsumsi daya dari *cache memory* dengan mensimulasikannya pada SMP*Cache* dengan berbagai macam *benchmark* dan CACTI.

Kata kunci:

Cache memory, Konsumsi Daya, Kinerja, SMP*Cache*, CACTI, *Cache Miss, Cache Hit*

Abstract

Creating high performance computers which consume minimal power is one of the focus and has become the trend of research in the field of computer architecture lately. It has been known that a number of factors influence the determination of the performance and power consumption of each of the building elements of the computer and ultimately affect the performance and overall power consumption of a computer. The focus of this research is to find out how much mapping functions as one of the type of design elements applied to a cache memory influence the performance and power consumption of the cache memory by simulating various configuration elements of the design in SMPCache with various benchmarks and CACTI.

Key*words*:

Cache memory, Power Consumption, Performance, SMPCache, CACTI, Cache Miss, Cache Hit

I. PENDAHULUAN

Sejak beberapa puluh tahun yang lalu, kinerja micro*processor* berkembang dengan sangat pesat. Hal ini dicapai diantaranya dengan tingginya *clock frequency* operasi micro*processor* dan adanya pengintegrasian beberapa inti *processor* ke dalam satu *chip*. Akan tetapi, komponen utama pembangun komputer lainnya seperti *main memory* tidak mampu mengimbangi perkembangan kecepatan micro*processor* tersebut, sehingga terdapat perbedaan yang sangat signifikan antara kecepatan *main memory* dan *microprocessor*. Hal ini menyebabkan adanya "*bottleneck"* pada kinerja *processor*. Salah satu langkah yang sudah dilakukan untuk menyelesaikan masalah tersebut adalah dengan menggunakan *cache memory*.

Cache memory, disebut juga sebagai *Central Processing Unit (CPU) memory* adalah sebuah *memory* yang dapat diakses oleh micro*processor* dengan lebih cepat dibanding *main memory*. Salah satu penyebabnya adalah *cache memory* diposisikan sedemikian rupa pada *chip* yang sama dengan micro*processor*, sehingga waktu yang dibutuhkan micro*processor* untuk mengakses *cache memory* menurun secara signifikan. Setiap saat, *cache memory* menyimpan salinan sebagian data yang tersimpan di *main memory* sehingga ketika *processor* perlu mengakses data dari *main memory*, maka *processor* dapat mengakses salinan data yang dibutuhkannya

tersebut dari *cache memory*. Dengan demikian, keberadaan *cache memory* dapat mengurangi ketimpangan kecepatan antara *main memory* dan micro*processor* dan secara langsung berkontribusi penting dalam meningkatkan kecepatan komputer secara keseluruhan.

Bersamaan dengan keuntungan penggunaan *cache memory* dalam perancangan micro*processor* moderen, *cache memory* juga memiliki kekurangan yang perlu ditangani, dimana *cache memory* merupakan salah satu komponen yang paling banyak mengkonsumsi daya dalam arsitektur Komputer (Saito et al., 2016). Salah satu penyebabnya adalah karena *cache* biasanya merupakan struktur terbesar pembangun sebuah micro*processor*, sehingga *cache memory* menjadi sumber utama disipasi daya statis. Hal ini tentu saja tidak sejalan dengan fokus kalangan industri untuk menciptakan micro*processor* yang memiliki kinerja tinggi dan mengkonsumsi daya yang lebih sedikit (Kothari, 2011). Menekan konsumsi daya ini menjadi sangat penting karena beberapa alasan (Zhu, 2011), diantaranya yaitu konsumsi daya sangat berpengaruh pada pembatasan kerapatan elemen pembangun *processor*, kebutuhan pendinginan sistem, keandalan sistem serta masa hidup baterai terutama pada *portable system.*

Diketahui bahwa, terdapat banyak faktor yang mempengaruhi besarnya konsumsi daya dari sebuah *cache memory*. Salah satu hal yang berpengaruh besar yaitu pemilihan strategi yang diterapkan pada saat perancangan *cache memory* (Abella and Gonzales, 2003). Elemen-elemen strategi perancangan yang dimaksud adalah seperti jenis *mapping techniques* yang digunakan serta ukuran *memory cache*. Dimana faktor-faktor tersebut tidak hanya berpengaruh terhadap kinerja sebuah *cache memory*, akan tetapi juga berpengaruh terhadap konsumsi daya sebuah micro*processor* secara keseluruhan. Oleh karena itu, pada penelitian ini, penulis mengekplorasi beberapa aspek perancangan *cache memory* dan bagaimana pengaruhnya terhadap kinerja serta konsumsi daya dari sebuah *cache memory*.

Dalam penelitian ini sebuah simulasi dilakukan terhadap beragam konfigurasi menggunakan SMP*Cache* (*Symmetric Multiprocessor*s *Cache Simulator*) dan CACTI untuk mengetahui seberapa besar pengaruh teknik pemetaan sebagai salah satu jenis elemen perancangan yang *cache* yang diterapkan dapat mempengaruhi kinerja *cache memory* yang terutama diukur dari *cache hit* yang dihasilkan dan besarnya konsumsi daya yang digunakan.

II. KAJIAN LITERATUR

II.1 Prinsip-prinsip *Cache memory*

Perangkat keras pembangun sistem komputer modern terdiri atas sejumlah komponen pembangun yang harus bekerja sama lain. Setiap komponen yang terlibat berkontribusi dalam menentukan kinerja komputer secara keseluruhan. Agar kinerja yang diharapkan dapat tercapai, maka setiap komponen harus dapat mengimbangi kinerja komponen lainnya. Akan tetapi, pada prakteknya, terjadi ketimpangan kecepatan yang cukup signifikan antara *processor* dengan *main memory*. Oleh karena itu, sejumlah strategi sudah dilakukan, salah satunya yaitu dengan memanfaatkan *cache memory* yang berfungsi sebagai perantara antara *processor* dan *main memory*.

Cache memory merupakan sebuah *memory* semikonductor yang merupakan tempat untuk menyimpan serangkaian data atau instruksi yang berasal dari *memory* lain yang relatif lebih lambat (Kumar & Pradesh, 2016). *Cache memory* didesain untuk mengkombinasikan *memory access time* dari *memory* mahal yang berkecepatan tinggi dengan *memory* berukuran besar, berkecepatan lebih rendah dan berharga lebih murah. Dimana terdapat *memory* yang relatif lambat dan berukuran besar bersamaan dengan *cache memory* yang lebih cepat dan lebih kecil. *Cache* berisi salinan sebagian dari isi *main memory*. Ketika *processor* berusaha untuk membaca sebuah *word* dari *memory*, pengecekan dilakukan untuk memeriksa apakah *word* yang dibutuhkan berada dalam *cache*. Jika ya, maka *word* tersebut dikirim ke *processor*. Jika tidak, sebuah *block main memory* yang terdiri atas beberapa *word* yang berukuran tetap, dikirim ke *cache* dan kemudian *word* tersebut dikirim ke *processor*. Karena fenomena *locality of reference*, maka ada kemungkinan bahwa referensi selanjutya ke lokasi *memory* yang sama atau *word* lain pada *block* tersebut.

II.2 Elemen Perancangan *Cache memory*

Terdapat beberapa penelitian terdahulu yang meneliti tentang pengaruh penerapan berbagai elemen perancangan terhadap kinerja dari sebuah *cache memory*. Dimana yang dimaksud dengan kinerja *cache memory* dalam hal ini memiliki beberapa parameter, diantaranya seperti yang disebutkan pada (Kumar & Pradesh, 2016) bahwa basis utama perhitungan kinerja

cache memory adalah *miss rate*, *miss penalty* dan *average access time*. *Miss rate* adalah akses terhadap bagian *main memory* yang salinannya tidak dapat ditemukan pada *cache memory*, sebaliknya *hit rate* adalah akses terhadap bagian dari *main memory* yang salinannya dapat ditemukan di *cache memory*. Miss *penalty* adalah jumlah total *CPU cycles* yang terhenti akibat usaha untuk mengakses *main memory*.

Walaupun terdapat sejumlah implementasi *cache*, terdapat beberapa elemen perancangan dasar yang berfungsi untuk mengklasifikasikan dan membedakan arsitektur *cache* (Stallings, 2013). Elemen-elemen perancangan *cache* tersebut dijelaskan sebagai berikut:

Alamat *Cache (Cache Address)*

Hampir semua *nonembedded processor*, dan banyak *embedded processor*, mendukung *virtual memory*. Pada dasarnya, virtual *memory* adalah sebuah fasilitas yang mengizinkan program untuk mengalamati *memory* dari sudut pandang logika, tanpa tergantung pada jumlah *main memory* yang tersedia secara fisik. Ketika virtual *memory* digunakan, *field* alamat dari instruksi mesin berisi alamat *virtual*. Untuk membaca dan menulis dari *memory*, sebuah *hardware memory management unit (MMU)* menterjemahkan setiap alamat *virtual* menjadi alamat fisik dalam *main memory*.

Ukuran *Cache (Cache Size)*

Perancang *cache* menginginkan *cache* dengan ukuran yang cukup kecil sehingga dengan demikian harga rata-rata harga per bit mendekati harga *main memory* dan cukup besar sehingga waktu akses rata-rata secara keseluruhan mendekati *cache* itu sendiri. Terdapat beberapa motivasi untuk meminimalisasi ukuran *cache*. Dimana semakin besar *cache*, semakin besar jumlah *gate* yang terlibat dalam pengalamatan *cache*. Akibatnya, *cache* yang besar cendrung agak lebih lambat dibanding dengan *cache* yang lebih kecil –bahkan ketika dibangun dengan teknologi IC (Integrated) yang sama dan diletakkan pada tempat yang sama dalam *chip* dan papan sirkuit. Ketersediaan *chip* dan area papan sirkuit juga membatasi ukuran *cache*. Karena kinerja *cache* sangat sensitif terhadap sifat alami dari beban kerjanya, maka tidaklah mungkin untuk mendapatkan ukuran *cache* yang optimum.

Teknik Pemetaan (*Mapping Techniques*)

Mapping techniques digunakan untuk memetakan *main memory block*s yang jumlahnya sangat besar ke *cache memory lines* yang jumlahnya jauh lebih kecil serta untuk membuat tanda (*tag*) berupa bits pada setiap *memory cache line* untuk memastikan *main memory block*s yang mana yang salinannya sedang tersedia pada *cache memory line* tertentu. Terdapat beberapa *mapping techniques* yang berperan menentukan organisasi dari sebuah *cache*, yaitu *direct mapping*, *associative mapping* dan *set associative mapping*.

Direct mapping merupakan teknik yang paling sederhana, memetakan setiap *block main memory* ke hanya satu *cache line* yang mungkin.

Associative mapping menangani kelemahan dari *direct mapping* dengan mengizinkan setiap *block main memory* dimuat ke setiap *line* yang ada pada *cache*. Dalam hal ini, *cache control logic* menginterpretasikan sebuah *memory address* hanya sebagai sebuah *Tag* and sebuah *Word field*. *Tag field* secara unik mengidentifikasikan sebuah *block main memory*. Untuk mengetahui apakah sebuah *block* ada dalam *cache*, *cache control logic* harus secara simultan memeriksa setiap *tag line* untuk mengetahui yang sesuai.

Set associative mapping (disebut juga dengan *k-way set associative*) merupakan sebuah kompromi yang memperlihatkan kelebihan dari pendekatan *direct* dan *associative* dan sekaligus memperkecil kekurangannya. Dalam hal ini, *cache* terdiri atas serangkaian set, dimana masing-masingnya terdiri atas sejumlah *lines*.

Algoritma Penimpaan (*Replacement Algorithm*)

Replacement algorithm memiliki peranan penting dalam perancangan *cache memory* karena *replacement algorithm* berperan untuk memutuskan *cache memory line* mana yang akan ditimpa oleh *main memory block* yang dibutuhkan. Terdapat 3 (tiga) alternantif *replacement algorithm* utama (Patterson & Hennessy, 2009), yaitu *First In First Out (FIFO), Least Frequently Used (LFU), Least Recently Used (LRU)* dan *Random*.

Aturan Penulisan (*Writing Policy*)

Writing policies berperan untuk menentukan bagaimana cara menjaga konsistensi antara isi *cache memory line* tertentu dengan *main memory block* terkait. Alternatif metode yang tersedia diantaranya adalah *write back* dan *write through*. Pada *write back* operasi penulisan hanya dilakukan pada *cache memory line*, sedangkan isi *main memory block* diperbaharui hanya ketika *cache memory line* yang terkait dengannya akan dihapus dari *cache memory*. Sedangkan pada *write through*, operasi penulisan dilaksanakan pada *main memory block* bersamaan dengan penulisan pada *cache memory line*s.

II.3 Software Simulator *Cache memory*

Pemilihan nilai atau teknik yang berbeda pada parameter-parameter perancangan *cache memory* sebagaimana dijelaskan diatas, dapat mempengaruhi kinerja dan konsumsi daya pada *cache memory* (Upadhyay 2016). Oleh karena itu untuk memilih konfigurasi terbaik dari sebuah *cache* maka diperlukan eksplorasi elemen-elemen perancangan *cache*. Penelitian (D. M. Cambre, E. Boemo, and E. Todorovich, 2008) mempelajari tentang hubungan antara instruksi dan ukuran data *cache memory* dan kemudian menganalisa hubungannya dengan konsumsi daya. Diketahui bahwa menggunakan *cache* size yang optimal dapat menghasilkan konsumsi daya terendah walaupun tetap mempertimbangkan waktu eksekusi, daya dan pemanfaatan sumberdaya-sumberdaya yang terdapat dalam sebuah FPGA.

Sebuah simulator untuk *Memory* Systems pada Symmetric Multi*processor*s (SMP*Cache*), yaitu sebuah alat bantu dari Department of Computer and Communication Technologies at the University Extremadura Escuela Politecnica di Spanyol. SMP*Cache* (M. Á. Vega Rodríguez, J. M. Sánchez Pérez, and J. A. Gómez Pulido, 2001) adalah sebuah *trace driven simulator* yang digunakan untuk menganalisa dan mensimulasikan *cache memory systems* pada *symmetric multiprocessor*s yang menggunakan bus yang berbasis *shared memory*. Simulator ini memungkinkan user untuk mengkonfigurasi arsitektur berbagai macam *processor* dengan merubah beragam parameter seperti: jumlah *processor*s, *cache coherence protocol*, skema arbitrasi, panjang *word* (*word* length), jumlah *word* dalam sebuah *block*, ukuran memori (*memory* size), jumlah tingkatan *cache memory*, teknik pemetaan *cache memory* (*mapping techniques*), ukuran *block cache memory* (*line* size), aturan penimpaan dan penulisan pada *cache memory* (*replacement policies*). Setelah memilih arsitektur tertentu pada *cache memory* tersebut, maka arsitektur tersebut diuji menggunakan *benchmark*s yang disediakan oleh simulator tersebut. Dimana, secara umum terdapat dua kategori *benchmark*s yaitu: sembilan buah *benchmark* untuk arsitektur *processor* tunggal dan empat buah *benchmark* untuk arsitektur *multicore computer*. Setelah pengujian dilakukan, simulator tersebut menyajikan sejumlah karakteristik dari *cache memory* seperti jumlah arbitrasi bus, jumlah *block* yang diangkut melalui bus tersebut, jumlah *memory* akses (pengambilan, pembacaan dan penulisan data), jumlah *cache hit* dan *cache* miss dalam *cache memory*.

CACTI (S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi, 2008) adalah sebuah software simulasi yang digunakan untuk menganalisa daya, area dan *timing* untuk arsitektur yang berbasis memori. CACTI (HP Labs, 2008) merupakan sebuah model terintegrasi untuk mensimulasikan *cache memory* dan *main memory access time, cycle time, area, leakage dan dynamic power*. Simulator ini ditujukan untuk digunakan oleh *computer architects* untuk memahami kompromi kinerja dalam organisasi sistem memori.

III. PEMODELAN DAN ANALISA HASIL SIMULASI

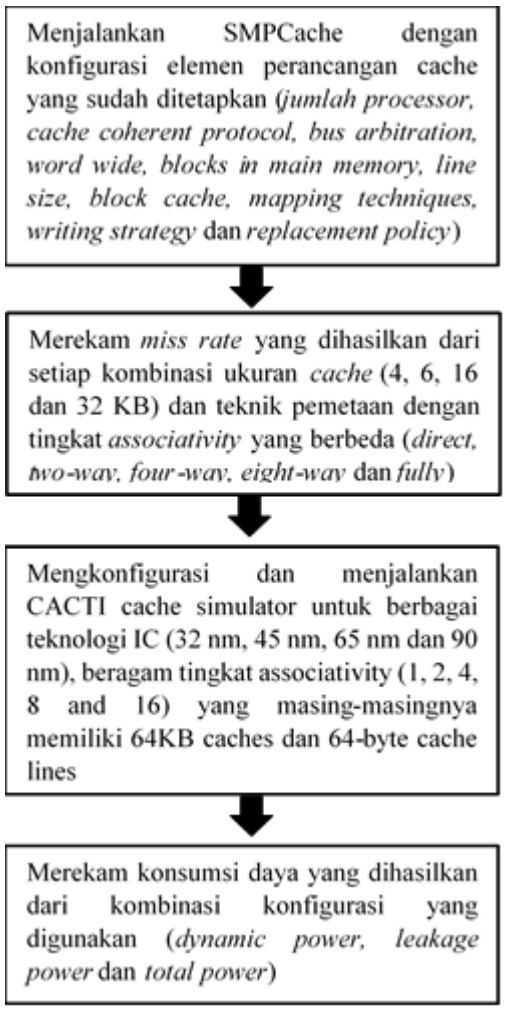
Secara umum, analisis dan perancangan dalam penelitian ini dilakukan melalui langkah-langkah yang terdapat pada gambar 1.

Dalam penelitian ini, terdapat 2 jenis peubah yang diamati, yaitu pertama, peubah yang akan diamati dalam penelitian ini adalah besarnya *cache hit* sebagai salah satu peubah yang dijadikan parameter kinerja dari sebuah *cache memory* yang dihasilkan dari berbagai macam kombinasi input yang diberikan dan berbagai macam teknik dari elemen perancangan yang diterapkan.

Kedua, peubah yang diamati adalah daya yang dikonsumsi sebuah *cache memory*. Hal ini juga dipengaruhi oleh elemen perancangan *cache memory*. Dimana penerapan sebuah teknik yang berbeda dapat memberikan pengaruh terhadap besarnya konsumsi daya dari suatu *cache memory*.

Gambar 1. Metodologi Penelitian

Langkah pengumpulan data dilakukan sebagai berikut: Pertama, mengkonfigurasi SMP*Cache* dengan menggunakan elemen perancangan seperti jumlah *processor*, *cache* coherence protocol, bus arbitration, *word* wide, *block*s in *main memory*, *line* size, *block* in *cache*, mapping, *writing strategy* dan *replacement policy* tertentu, menjalankan SMP*Cache* dengan konfigurasi tersebut dan merekam tingkat *miss rate* yang dihasilkan sebagai salah satu parameter kinerja *cache memory* (langkah ini diulangi dengan menggunakan setiap kemungkinan kombinasi konfigurasi yang ada). Kedua, mengkonfigurasi CACTI *cache* simulator dengan menggunakan konfigurasi yang sama dengan yang digunakan pada SMP*Cache* dan menjalankan setiap kombinasi aspek perancangan yang digunakan untuk mengetahui besarnya konsumsi daya dari setiap konfigurasi yang digunakan. Ketiga, menganalisa hubungan antara keluaran SPM*Cache* dan CACTI *cache* simulator yang menggunakan konfigurasi yang sama, kemudian membandingkan keluaran tersebut dengan keluaran kedua simulator tersebut saat menggunakan kombinasi konfigurasi lainnya, kemudian menganalisa data secara keseluruhan.

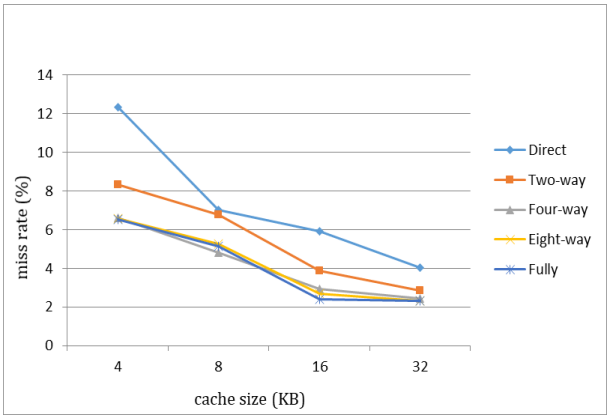


Gambar 2. Langkah Pengujian dan Pengumpulan Data

III.1 Hasil Simulasi Menggunakan SMP*Cache*

Pengujian dengan menggunakan software simulasi ini adalah untuk mengetahui pengaruh perubahan pada fungsi pemetaan yang digunakan terhadap *cache* hit (melalui penghitungan persentase *cache* miss) yang dihasilkan.

Pengujian dilakukan pada *processor* tunggal dan multicore, dengan menggunakan *Cache* coherence protocol = MESI, Scheme for bus arbitration = Random, *Word* wide (bits) = 32, *Word*s by *block* = 64 (*block* size = 256 bytes), *block*s in *main memory* = 4096 (*main memory* size = 1024 KB) dan *replacement* policy = LRU.



Gambar 3. Pengaruh Tingkat *Associativity* dan *Cache Size* terhadap *miss rate*

Dari grafik diatas dapat diketahui bahwa persentase *miss rate* menurun seiring dengan meningkatnya tingkat *associativity*. Dimana yang dimaksud dengan tingkat *associativity* di sini adalah jumlah line yang terdapat pada sebuah rangkaian (set) yang pada akhirnya menentukan jumlah set keseluruhan yang terdapat pada sebuah *cache memory*.

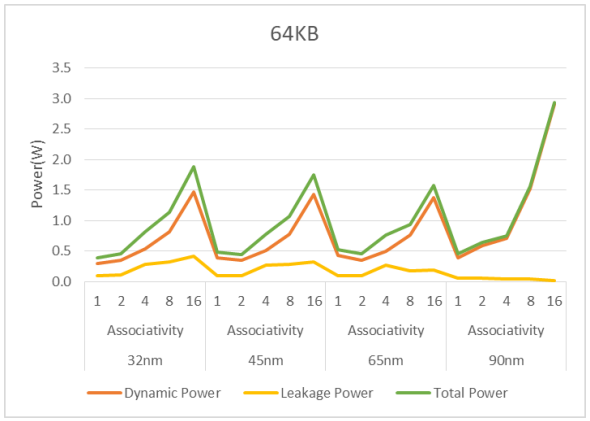
$$m = \mathrm{v} \times \mathrm{k}$$
$$i = j \; modulp \; v$$

dimana,
i = *cache set number*
j = *main memory block number*
m = *number of lines in the cache*
v = *numbers of sets*
k = number of lines in each set

Tingkat *associativity* juga memiliki pengaruh terhadap persentase *miss rate* ketika ukuran *cache* diperbesar. Dalam hal ini, semakin besar ukuran *cache*, maka jumlah *line* yang tersedia pada *cache* tersebut untuk menampung *block* yang disalin dari *main memory* semakin banyak, sehingga ketika *processor* meminta data yang tersimpan di alamat *memory* tertentu, kemungkinan besar *block* tersebut ada di dalam *cache* (*cache* hit). Akan tetapi, pengaruh *cache* size tersebut semakin kecil untuk menurunkan tingkat *miss rate* seiring dengan meningkatnya ukuran *cache*. Dimana ketika *cache* size diperbesar dari 16KB menjadi 32 KB, persentase *cache* miss-nya cendrung tidak berubah.

III.2 Hasil Simulasi Menggunakan CACTI

Pengujian dengan menggunakan simulator CACTI dilakukan untuk mengetahui pengaruh tingkat *associativity* yang diterapkan terhadap *dynamic power*, *leakage power* serta *total power* yang dikonsumsi. Pengujian dilakukan pada teknologi IC (Integrated Circuit) 32 nm, 45 nm, 65 nm dan 90 nm dengan 64KB *caches* dengan 64-byte *cache lines*.



Gambar 4. Pengaruh Assosiatifitas terhadap Konsumsi Daya

Dari pengujian tersebut diketahui bahwa tingkat *associativity* berpengaruh terhadap besarnya *leakage power* (static power) yaitu konsumsi daya yang terjadi saat tidak ada aktifitas dalam rangkaian, dan *dynamic power* (konsumsi daya saat input bernilai aktif) pada sebuah *cache*. Dimana Semakin tinggi tingkat *associativity* pada sebuah *cache* yang berukuran sama, semakin besar pula jumlah konsumsi daya *cache* tersebut.

Untuk ukuran *cache* yang berbeda, pola yang hampir sama juga terjadi. Secara umum jumlah konsumsi daya tersebut cendrung sedikit menurun seiring dengan meningkatnya technology *processor* yang digunakan. Akan tetapi, perubahan yang sangat signifikan terjadi pada technology IC 90nm dimana konsumsi daya totalnya meningkat secara tajam seiring dengan peningkatan jumlah *associativity* terutama dari 8-way ke 16 *way associativity*.

IV. KESIMPULAN DAN SARAN

Dari eksperimen-eksperimen yang dilakukan diketahui bahwa jenis pemetaan sangat berpengaruh terhap persentase *miss rate* yang terjadi pada sebuah *cache*. Dimana semakin tinggi tingkat *associativity*,

semakin rendah *miss rate* yang terjadi pada sebuah *cache*. Artinya adalah kinerja *cache* tersebut meningkat dari segi *cache* hit yang terjadi. Akan tetapi, peningkatan kinerja tersebut juga harus diiringi oleh peningkatan konsumsi daya. Penelitian tentang pengaruh aspek-aspek lain dari perancangan *cache memory* perlu ditelaah lebih lanjut sehingga didapatkan *trade off* yang optimal antara kinerja yang diperoleh dari peningkatan *associativity* dengan peningkatan jumlah daya yang diperlukan.

REFERENSI

K. Saito, R. Kobayashi, and H. Shimada, "Reduction of *Cache* Energy by Switching between Ll High Speed and Low Speed *Cache* under application of DVFS," in *International Conference On Advanced Informatics: Concepts, Theory And Application (ICAICTA)*, 2016.

D. P. Kothari, "A Study on Factors Influencing Power Consumption in Multithreaded and Multicore CPUs," *WSEAS Trans. Comput.*, vol. 10, no. 3, pp. 93–103, 2011.

Z. Zhu, "Power Efficient Designs." [On*line*]. Available: http://home.eng.iastate.edu/~zzhang/courses/cpre585-f04/slides/Lecture24.pdf. [Accessed: 22-Feb-2017].

J. Abella and A. Gonzalez, "Power efficient data *cache* designs," in *Proceedings 21st International Conference on Computer Design*, 2003, no. 3, pp. 8–13.

S. Kumar and U. Pradesh, "An Overview of Modern *Cache* and Performance Analysis of *Replacement* Policies," in IEEE International Conference on Engineering and TEchnology (ICETECH), 2016, no. March, pp. 210–214.

W. Stallings, *Computer Organization and Architecture : Designing for Performance*, Ninth Edit. Pearson, 2013.

D. A. Patterson and J. L. Hennessy, Computer Organization and Design: The Hardware/Software Interface, vol. 4th, no. 0. 2009.

B. R. Upadhyay and S. TSB, "Design Space Exploration of *Cache memory* – A Survey," in International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT), 2016, pp. 2294–2297.

D. M. Cambre, E. Boemo, and E. Todorovich, "Energy Evaluation in the Nios II *Processor* as a Function of *Cache* Sizes," in Southern Conference on Programmable Logic, 2008, pp. 55–61.

M. Á. Vega Rodríguez, J. M. Sánchez Pérez, and J. A. Gómez Pulido, "An educational tool for testing *cache*s on symmetric multi*processor*s," Micro*processor*s and Microsystems, 2001.

S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi, "A comprehensive *memory* modeling tool and its application to the design and analysis of future *memory* hierarchies," Proc. - Int. Symp. Comput. Archit., pp. 51–62, 2008.

[11] HP Labs, "CACTI - An integrated *cache* and *memory* access time, cycle time, area, leakage, and dynamic power model," 2008. [On*line*]. Available: http://www.hpl.hp.com/research/cacti/. [Accessed: 23-Feb-2017].