ANALISIS PERFORMANSI CACHE COHERENCY PROTOCOL MOESI CMPT TOKEN TERHADAP PENAMBAHAN PROSESSOR PADA NUMA

Firdaus Rachmawan[1], Tri Brotoharsono [2], Maman Abdurrohman[3]

[1]Teknik Informatika, Fakultas Teknik Informatika, Universitas Telkom

Abstrak

Saat ini, NUMA sudah menerapkan cache coherence protocol, salah satunya MOESI CMP Token.Hanya saja, muncul masalah lain berupa cache miss rate yang makin besar dengan jumlah prosessor yang makin meningkat. Oleh karena itu, dalam tugas akhir ini,dilakukan pengukuran performansi ketika jumlah prosessor semakin bertambah menggunakan simulator gem5 yang dipadukan dengan TOPAZ dan dua macam benchmark. Adapun parameter lain yang diukur berupa execution time, thorughput, latency, dan average memory access time. Hasil yang didapatkan dari simulasi ini adalah penambahan prosessor mampu mengurangi waktu eksekusi seluruh proses sekaligus mendorong maksimal throughput yang berlaku bagi kedua benchmark. Adapun memori lokal akan lebih maksimal waktu pengaksesannya dibandingkan memori di node lain. Persentase penerimaan paket atau datamencapai 39-40% pada 4-CPU dan 31-32% pada 8-CPU.Sebagai pengecualian pada penambahan 16-CPU ditemukan persentase maksimal penerimaan paket sebesar 26%.

Kata Kunci : NUMA, gem5, TOPAZ, latency, cache coherence, cache missrate

Abstract

Currently, NUMA cache coherence protocols already implemented, one of which MOESI CMP Token. However, another problem arises in the form of a cache miss rate is greater with increasing number of processors. Therefore, in this thesis, measured as the number of processor performance using a simulator gem5 increasingly combined with TOPAZ and two kinds of benchmarks. The other parameters are measured in the form of execution time, thorughput, latency, and average memory access time. Results obtained from these simulations is the addition of processor capable of reducing the execution time of the whole process while encouraging maximum throughput benchmarks apply to both. The local memory will be more than the maximum time accessing memory on another node. Percentage acceptance or data packets reach 39-40% in the 4-CPU and 31-32% on an 8-CPU. As an exception to the addition of 16-CPU found the maximum percentage of 26% packet reception.

Keywords : NUMA, gem5, TOPAZ, latency, cache coherence, cache missrate

menjadi kunci penting dari perancangan sistem yang matang. Dalam menentukan perancangan ini digambarkan dalam bentuk diagram blok.

5. **Implementasi Sistem**

   Tahap ini dilakukan simulasi menggunakan *Gem5* dengan mode *full-system* sehingga dapat diketahui simulasi sistem tersebut dengan mempertimbangkan *OS* yang digunakan beserta aplikasi yang dijalankan di *OS* itu.

6. **Pengujian Sistem**

   Tahap ini memastikan sistem dapat dijalankan sesuai dengan hipotesa dan tujuan yang ingin dicapai. Pengujian ini dilakukan dengan skenario yang telah ditentukan sehingga memilki alur yang jelas

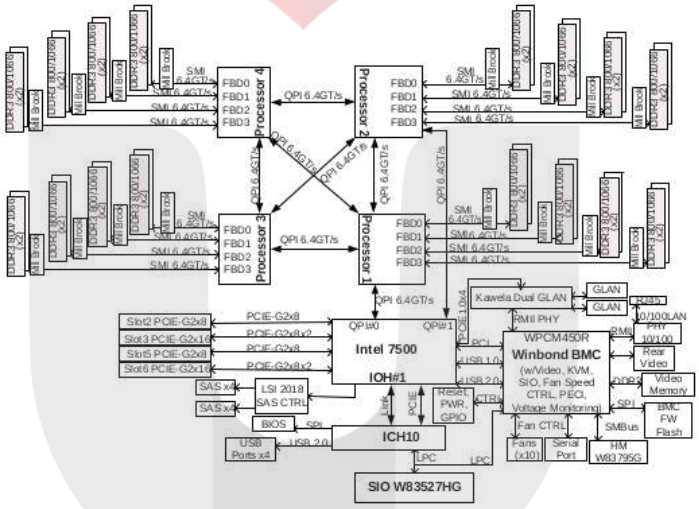
7. **Penyusunan Buku TA**

   Penyusunan buku ini sekaligus memaparkan hasil pengujian serta menarik kesimpulan dari sistem yang telah dirancang

2. Tinjauan Pustaka

2.1. NUMA

NUMA merupakan arsitektur komputer yang menerapkan memori local pada setiap *prosessor* sehingga *cache* mampu mengakses proses yang ada di dalam memori dengan waktu yang relatif lebih cepat. Adapun *prosessor* dapat mengakses memori di prosessor lain dengan bantuan protokol tertentu yaitu *cache coherence protocol*. NUMA yang memanfaatkan *cache coherence protocol* dinamakan ccNUMA[11].



Gambar 2-1 Arsitektur NUMA[15]

2.2. Cache Coherence

Cache coherence merupakan teknik yang digunakan oleh *multiprosessor* agar dapat menjaga keutuhan dan kebenaran data ketika *cache* saling berbagi. Adapun penerapannya pada model SMP *(Shared Memori Prosessor)* hingga NUMA *(Non-Uniform Memori Access).Cache coherence* diatur oleh sebuah aturan atau protokol yang sering disebut dengan *cache coherence protocol*[11].

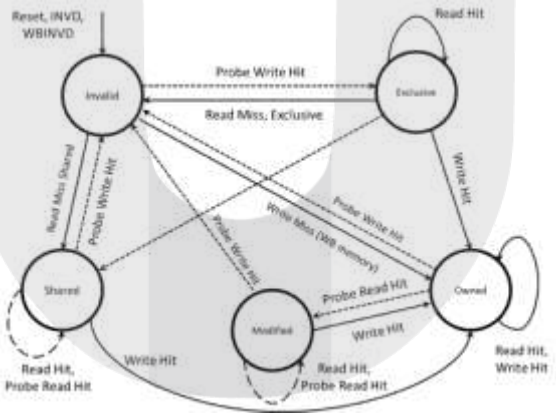
2.2.1. Cache Coherence Protocol

Cache coherence protocol berguna untuk membantu *cache coherence* tetap menjalankan fungsinya dengan baik. Protokol ini diimplementasikan sebagai tatat aturan bagi hardware komputer agar pengaksesan data yang dilakukan oleh *cache* menuju pada lokasi memori yang seharusnya. *Cache coherence protocol* dibagi menjadi dua macam *snooping* dan *directory-based*. Saat ini, *directory-based* yang sering digunakan, salah satu contohnya *MOESI*[11].

2.2.1.1. MOESI

Protokol ini berbasis directory-based yang merupakan pengembangan dari protokol MESI. Penambahan state O yang berarti Owned menjadi bukti dari protokol sebelumnya. Adapun detail dari masing-masing statenya [5]:

a. *Invalid* : *cache block* pada *state* ini tidak meyimpan salinan data. Salinan data terdapat pada memori atau *cache* yang lain[5].
b. *Exclusive*: *cache block* pada *state* ini menyimpan salinan data baru sekaligus benar. Salinan data terdapat pada memori utama dan tidak ada prosesor lain yang memegang salinan datanya[5].
c. *Shared*: *cache block* pada *state* ini menyimpan salinan data yang baru dan benar, tapi prosesor lain mungkin juga mempunyai salinan data .
d. *Modified* : *cache block* pada *state* ini menyimpan salinan data yang baru dan benar. Salinan data di memori utama yang kurang tepat dan hanya satu prosessor yang mempunyai salinan datanya[5].
e. *Owned* : *cache block* pada *state* ini menyimpan salinan data yang baru dan benar. Adapun pada state "*shared*"semua prosesor wajib memegang data[5]



Gambar 2-2 MOESI *Coherence Protocol State* [8]

2.2.1.2. MOESI CMP Token

Salah satu jenis dari *MOESI* yang menggunakan token sebagai cara untuk suatu cache blok menulis data. Adapun token ini telah ditetapkan pada masing-masing cache yang ada pada setiap *prosessor* sehingga tidak akan pernah berubah. Susunan *2 level cache* menjadi model utama dari protokol ini[5][8]

2.3. Gem5 *Simulator*

gem5 merupakan simulator yang mampu memodelkan arsitektur komputer layakya mikroarsitektur komputer[6]. Secara garis besar, simulator ini perpaduan dari M5 dan GEMS. Keduanya memiliki fitur masing-masing yang diharapkan dapat menjadi simulator yang bagus. M5 memiliki kelebihan di antaranya mampu dilakukan konfigurasi, menyediakan beberapa jenis *ISA (X86, ALPHA, MIPS,*

dsb), serta pemodelan yang berbeda dari *CPU (atomic, timing,* dan *detailed).* Sedangkan GEMS fitur yang mampu menggambarkan alur kerja cache coherence protocol dan interkoneksi.

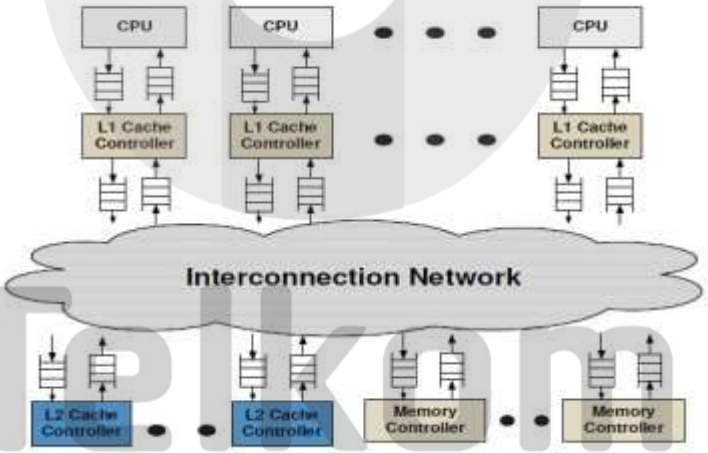
Gem5 menyediakan dua mode simulasi yaitu *System-call Emulation* dan *Full System*. Perbedaannya terletak pada penggunaan OS dan keseluruhan perangkat ketika simulasi dijalankan. Terkait dengan model memori yang ditawarkan terdapat dua macam yaitu *Classic dan Ruby.Classic* menawarkan cepat , kemudahan konfigurasi, tetapi akurasi belum bisa dijamin. Lain halnya, dengan *Ruby* yang lebih fleksibel karena mampu dikoneksikan dengan *cache coherence protocol* serta fitur interkoneksi[4]

2.3.1.1. SLICC

SLICC adalah bahasa khusus untuk menerjemahkan *cache coherence protocol*. SLICC dibangun dengan menggunakan *compiler* berbasis *C++* yang mampu dikombinasikan dengan *Ruby* dan*HTML*[6].

2.3.1.2. Interconnection Network

Fitur dalam simulator gem5 yang mampu menggambarkan simulasi dari alurpacket data terkirim pada topologi suatu jaringan. Dalam hal ini, gem5 menyediakan beberapa topologi yang memiliki karakteristik masing-masing, diantaranya *Pt2pt, MeshDir, Mesh, Torus,* dan *Crossbar*. Adapun penyusunan topologi interkoneksi dapat menggunakan simulator eksternal bernama *TOPAZ*[9].



Gambar 2-3 *Interconnection Network* pada Ruby[9]

2.4. TOPAZ

Simulator interkoneksi yang diadaptasi dari *Syscosis*. Simulator ini dapat memberikan mekanisme kerja paket ketika di-*inject* ke dalam jaringan dengan kecepatan dan presisi yang berbeda. TOPAZ dapat berjalan pada OS berbasis UNIX dengan menggunakan compiler C++. Penggabungan dengan gem5 akan membantu dalam melakukan simulasi jaringan dengan penataan yang lebih rapi berdasarkan parameter yang tersedia pada Ruby.py di Gem5[1].

Saat menjalankan TOPAZ akan terbagi dalam tiga langkah yaitu *building, running*, dan *printing*. Di lain sisi, struktur dari TOPAZ terdiri atas*component* yang menjabarkan perangkat yang tersusun pada topologi yang dibangun, misalkan berapa router atau node yang akan dipakai. Kemudian, komponen tersebut akan diatur alurnya oleh tahap *flow* sehingga dapat menghasilkan latency, throughput, dan hop yang dilewati oleh *message*. *Flow* pada TOPAZ diatur berdasarkan file SGML sebagai referensi router dan bentuk topologi yang dibangun[1].

2.4.1. *SGML*

SGML terdiri atas tiga buah file yaitu Router.sgm , Network.sgm, dan Simula.sgm. Masing-masing merupakan penggambaran komponen dan flow dari topologi yang dibangun menggunakan TOPAZ. Router.sgm mengatur arsitektur mikrorouter serta port yang disediakan oleh router tersebut. Network.sgm yang menggambarkan susunan dari router yang disediakan pada Router.sgm. Terakhir, Simula.sgm memuat mekanisme *flow* yang akan bekerja ketika pesan atau data dikirimkan pada jaringan tersebut. Ketiga file tadi, dapat dijadikan referensi dengan mencantumkan file TPZSimul.ini menggunakan paramater *topaz-init-file*[14]

2.5. PARSEC *Benchmark*

Pengukuran cara kerja *prosessor* beserta arsitekturnya harus memiliki tolak ukur tertentu, salah satunya *benchmark*. Ada beberapa *benchmark*, salah satunya PARSEC.PARSEC dirancang untuk menggambarkan performansi dari *prosessor* ketika diberi beban aplikasi tertentu.Di dalam PARSEC sudah menyediakan aplikasi yang berbasis *recognition, mining*, dan *sinthesis* yang semirip mungkin dengan aplikasi komersial atau berbayar. Adapun aplikasi yang termasuk dalam PARSEC *benchmark* :[3]

a. Blackscholes
b. Bodytrack
c. Canneal
d. Dedup
e. Facesim
f. Ferret
g. Fluidanimate
h. Freqmine
i. Raytrace
j. Streamcluster
k. Swaptions
l. Vips
m. x264

PARSEC dapat disatukan dengan gem5 menggunakan parameter *script.* Cara mendapatkan *thread* dari PARSEC dapat membuatnya sendiri atau menciptakan dari generator yang disediakannya[3].

2.6. Quality Of Service (QOS)

Quality of service adalah jaminan suatu layanan terhadapperformansi untukmengukur kualitas. Parameter uji dalam simulasi ini diantaranya *execution time, latency, throughput, cache miss rate, dan average memori access latency.*

2.6.1. *Execution Time*

Waktu yang diperlukan untuk menjalankan proses secara bersamaan pada setiap CPU.

2.6.2. *Latency*

Latency adalah waktu yang ditempuh oleh suatu data untuk mencapai node lainnya. Dalam simulasi ini, latency adalah waktu tempuh ketika mengirimkan data melalui jalur hypertransport yang menghubungkan antar node prosessor pada NUMA.

2.6.3. *Throughput*

Throughput adalah ukuran suatu data yang mampu diterima oleh node tujuan melalui bus atau jalur lainnya. Adapun rumusnya sebagai berikut:

$$Throughput = \frac{\sum paket\ yang\ diterima\ dalam\ bit}{\sum waktu\ pengiriman\ paket} \quad bit/s$$

2.6.4. *Cache Miss Rate*

Cache miss dapat diartikan bahwa cache tidak dapat menemukan instruksi atau data yang dicari pada cache. Terkadang, kasus ini akibat pembacaan dari memori utama yang kurang sempurna. Berikut rumus pencariannya :

$$Cache\ Miss\ Rate = \frac{\sum cache\ demand\ misses}{\sum cache\ demand\ accesses}$$

2.6.5. *Average Memori Access Time*

Average Memori Access Time merupakan waktu yang dibutuhkan untuk mengakses memori di mana dipertimbangkan waktu hit pada cache L1 dan Miss Rate pada L1 dengan satuan persen. Adapun *miss penalty* dihitung dari L2 cache dengan satuan international second.[2]

$$AMAT = Hit\ Time + Miss\ Rate\ x\ Miss\ Penalty$$

3. Perancangan Simulasi

3.1. Gambaran Umum Simulasi

Sistem yang dibangun menggunakan model *ISA X86* yang didukung dengan penambahan prosessor secara bertingkat hingga *16 CPU*. Protokol *cache coherence* yang dipakai adalah *MOESI CMP Token* berbasis *directory-based.*

3.2. Analisis Kebutuhan Pembangunan Simulasi

Berikut spesifikasi simulasi sistem NUMA pada tugas akhir ini sebagai berikut.

Perangkat	Keterangan
Sistem Operasi	Ubuntu 12.04 64bit
Simulator	Gem5 + TOPAZ
Motherboard	Keterbatasan sistem gem5 dan TOPAZ untuk NUMA sehingga belum ada motherboard di dunia nyata dalam simulasi ini
Prosesor	Intel® 7500 Series CPU 2,9 GHz
RAM	2GB dengan dua *memori-channel*

Tabel 3-1 Spesifikasi perangkat keras dan lunak

3.3. Desain Simulasi

Tahap ini merupakan penggambaran sistem secara keseluruhan sebagai interpretasi yang akan dibangun. Beberapa hal yang akan dibahas antara lain Model Prosesor, Model Sistem Simulasi, *Benchmarks*, Model *Interconnection Network* .

3.3.1. Model Prosesor

Prosessor yang dipakai menggunakan produk dari Intel dengan seri 7500 Series. Adapun detail dari perangkat ini sebagai berikut :

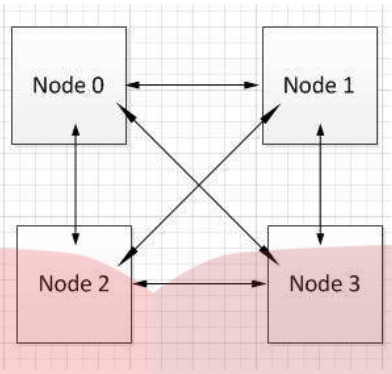
Nama Perangkat	Nilai
Prosesor	Intel® 7500 Series
Frekuensi (prosessor clock)	2 GHz
L1 *Instruction* + L1 Data	64 kB + 64kB
L2	2 MB
Cache Line Size	64
***QPI* clock**	2 GHz

Tabel 3-2 Model prosesor dan *cache*

3.3.2. Model *Interconnection Netwok*

Sistem interkoneksi yang dibangun menggunakan Mesh dengan ukuran 4x4. Adapun masing-masing node memiliki satu memori *controller* yang berperan

sebagai memori lokal. Kemudian jumlah CPU mewakili jumlah core dalam satu node.



Gambar 3-1 Model *Interconnection Network*

Jumlah Router	16
Jenis Topologi	Mesh-4x4 (M44_NOC_CT_UC)
Clock Ratio (QPI clock)	2 GHz
flit size	16

Tabel 3-3 Detail konfigurasi Topaz yang digunakan

Jenis topologi Mesh 4x4 yang digunakan merupakan konfigurasi tambahan pada file Network.SGML dari directory TOPAZ oleh penulis.

3.3.3. Benchmarks

Sistem benchmark yang digunakan adalah bodytrack dan blackscholes dengan mode test sebagai input dari simulasi menggunakan gem5 dan TOPAZ[3].

1. **Bodytrack**
   Benchmark yang digunakan untuk mengidentifikasi tubuh manusia dengan bantuan kamera berupa inputan serangkaian gambar yang saling terkait. Adapun sistem kerja thread pada benchmark ini yaitu dengan meletakkan pada pool terpusat atau disebut thread pool. Thread pool itu merupakan thread utama yang akan mencapai pada kernel secara paralel. Dengan adanya thread utama membantu resume eksekusi program tersebut menerima hasil dari thread worker. Dalam percobaan ini, digunakan thread sejumlah prosessor yaitu 4,8, dan 16 thread

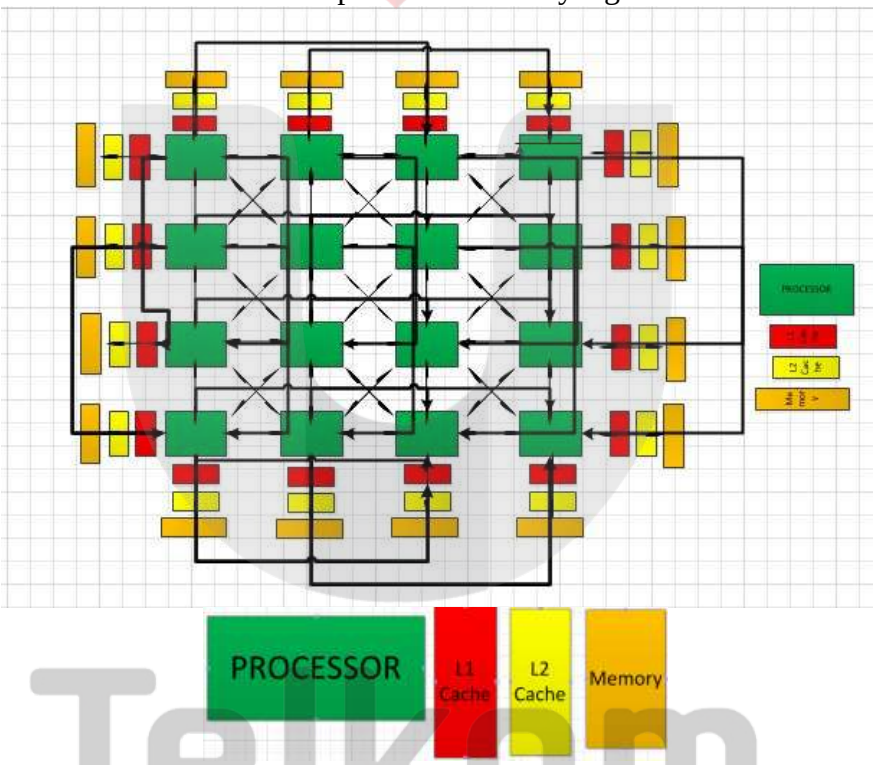
2. **Blackscholes**
   Benchmark ini banyak digunakan untuk membantu perhitungan harga yang diadaptasi dari algoritma blackscholes PDE sehingga memiliki komputasi yang rumit untuk memecahkan suatu masalah. Adapun sistem pembagian thread dan proses dari benchmark ini berdasarkan jumlah core secara bersamaan. Input sets pada benchmark ini berupa jumlah option komputasi. Dalam percobaan ini akan dilakukan jumlah therad sesuai dengan jumlah corenya yaitu 4, 8, dan 16 thread.

3. **Dedup**

Benchmark ini menerapkan kombinasi data stream baik secara global maupun lokal sehingga menciptakan deduplikasi pada jaringan untuk memudahkan dalam storage. Adapun alur kerja paralisasi benchmark ini berupa pipeline. Cara kerja sistem threadnya hampir sama dengan blackscholes sesuai dengan jumlah core prosessor. Namun, ada semacam penjadwalan tertentu supaya menghindari *contention* berupa queue pada input maupun outputnya. Jumlah threadnya sesuai dengan jumlah prosessor berupa 4,8, dan 16 thread.

3.3.1. Model Sistem Simulasi

Model NUMA yang diimplementasikan pada Gem5 Simulator diadaptasi motherboard yang dijabarkan sebelumnya. Dalam hal ini parameter yang sama adalah ukuran L1 baik Instuction dan Data, L2 cache, ukuran memori, dan memori channel. Berikut adalah spesifikasi sistem yang akan disimulasikan .



Gambar 3-2 Model Sistem Simulasi Menggunakan Mesh 4x4

Berikut detail sistem yang akan diimplementasikan pada Gem5 simulator.

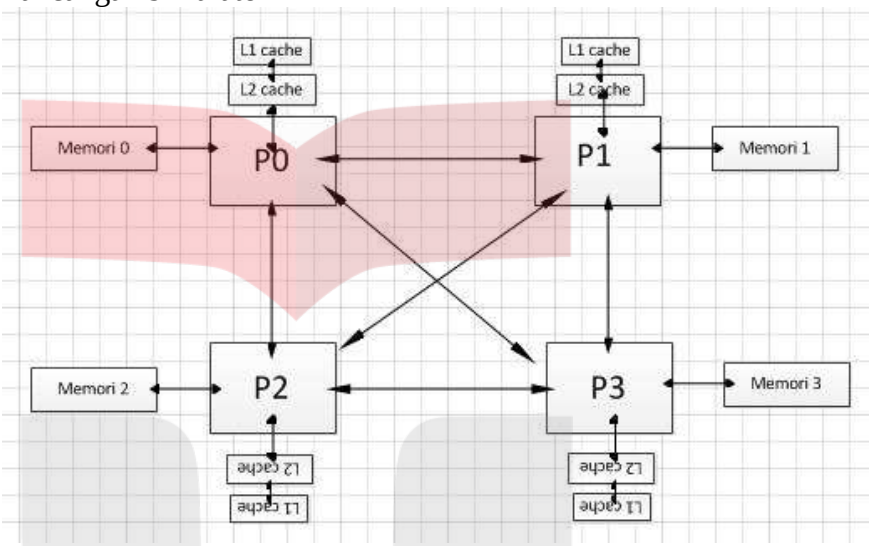
Nama Perangkat	Nilai
Jumlah Prosesor	4
Frequency (prosessor clock)	2 GHz
L1 *Instruction* + L1 Data	64 kB + 64kB
L2	2 MB
RAM	2 GB DDR3
***Clock Ratio*(*QPI clock*)**	2 GHz

Tabel 3-4 Detail sistem simulasi

Konfigurasi tersebut telah disediakan oleh *gem5* yang dikombinasikan dengan *Ruby.py* sebagai dasar *library* yang diimport pada program yang mendukung. Adapun mode simulasi yang digunakan adalah *Full System*. Skenario yang akan

disimulasikan hanya satu buah yaitu menambah jumlah prosessor secara bertingkat dengan dua macam *benchmark*

a. Rancangan simulator 1



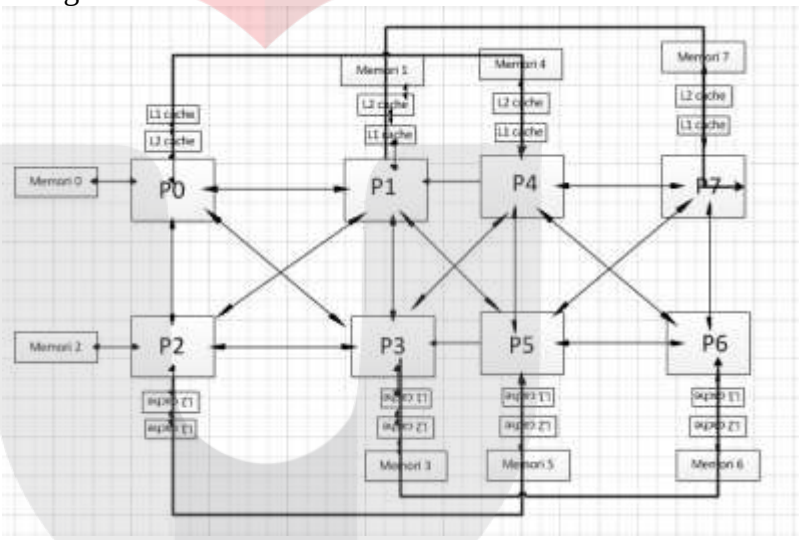
Gambar 3-3 Model Sistem Simulasi Menggunakan Mesh 4x4 untuk 4-CPU

Cache Coherence Protocol	MOESI CMP Token
Mode Simulasi	Full System
Jumlah CPU	4
Arsitektur	X86
CPU-Clock	2GHz
Ruby-Clock	1GHz
Jumlah Directory	*4*
Ukuran L1I Cache	64 kB
Ukuran L1D Cache	64 kB
Ukuran L2 Cache	2MB
Jumlah L2 Cache	4
Ukuran Cache Line	64
Ukuran Memori	2GB

Memori Channel	2
Memori Type	lpddr2 _s4 _1066_x32
Benchmark	Bodytrack, Blackscholes, dan Dedup
Topologi	Mesh
Mesh-row	2
Topaz Clock Ratio	1
Topaz Flit Size	16 byte

Tabel 3-5 Rancangan Simulator 1

b. Rancangan Simulator 2



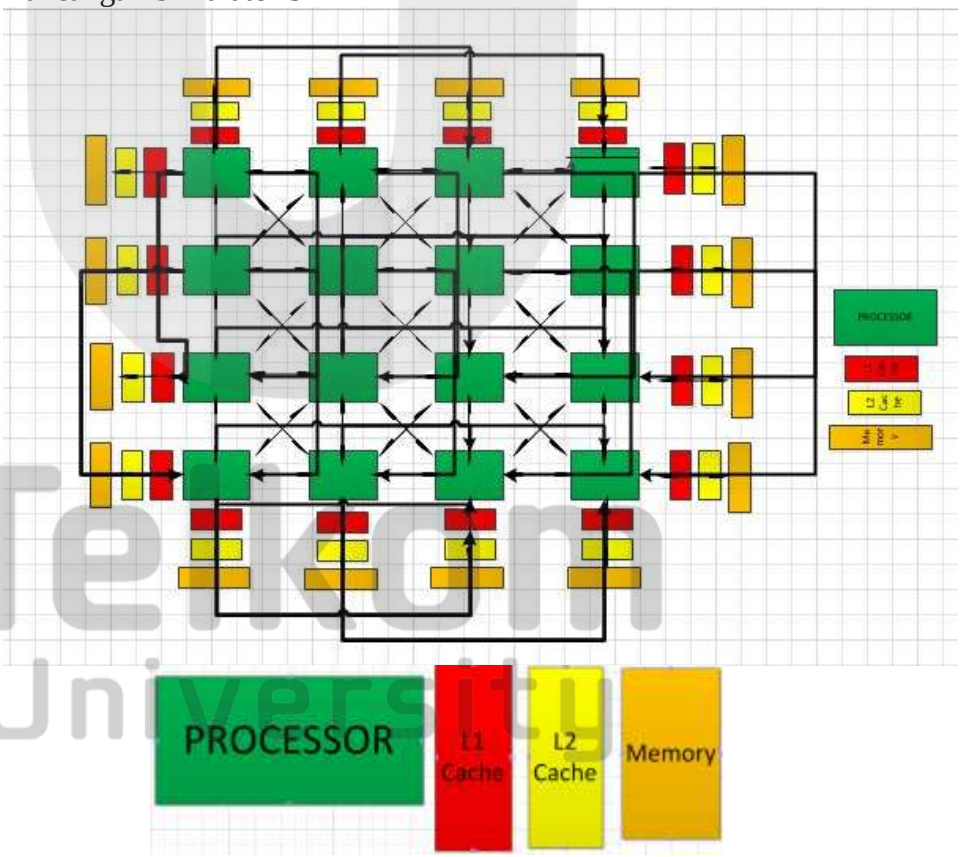
Gambar 3-4 Model Sistem Simulasi Menggunakan Mesh 4x4 untuk 8-CPU

Cache Coherence Protocol	MOESI CMP Token
Mode Simulasi	Full System
Jumlah CPU	8
Arsitektur	X86
CPU-Clock	2GHz
Ruby-Clock	1GHz
Jumlah Directory	8
Ukuran L1I Cache	64 kB
Ukuran L1D Cache	64 kB
Ukuran L2 Cache	2MB

Jumlah L2 Cache	8
Ukuran Cache Line	64
Ukuran Memori	2GB
Memori Channel	2
Memori Type	lpddr2 _s4 _1066_x32
Benchmark	Bodytrack,Blackscholes, dan Dedup
Topologi	Mesh
Mesh-row	2
Topaz Clock Ratio	1
Topaz Flit Size	16 byte

Tabel 3-6 Rancangan Simulator 2

c. Rancangan Simulator 3



Gambar 3-5 Model Sistem Simulasi Menggunakan Mesh 4x4 untuk 16-CPU

Cache Coherence Protocol	MOESI CMP Token

Mode Simulasi	Full System
Jumlah CPU	16
Arsitektur	X86
CPU-Clock	2GHz
Ruby-Clock	1GHz
Jumlah Directory	*16*
Ukuran L1I Cache	64 kB
Ukuran L1D Cache	64 kB
Ukuran L2 Cache	2MB
Jumlah L2 Cache	16
Ukuran Cache Line	64
Ukuran Memori	2GB
Memori Channel	2
Memori Type	lpddr2 _s4 _1066_x32
Benchmark	Bodytrack, Blackscholes, dan Dedup
Topologi	Mesh
Mesh-row	4
Topaz Clock Ratio	1
Topaz Flit Size	16 byte

Tabel 3-7 Rancangan Simulator 3

3.4. Desain Pengujian

Skenario pengujian terdiri atas tiga skenario sesuai yang dijabarkan di atas dengan aplikasi uji satu benchmark dari PARSEC yang jumlah thread-nya bervariasi sesuai dengan jumlah CPU dalam satu socket. Berikut detail langkah-langkah pengujian :

3.4.1 Pembangunan Simulasi

Simulasi ini dibangun menggunakan Gem5 dengan mode full system.Sebelum simulasi ini bisa dijalankan maka membutuhkan debug optimasi supaya menghasilkan file .opt sebagai file biner untuk menjalankan simulator ini. Kemudian beberapa file berformat python menjadi library dasar agar gem5 ini mampu menjalankan fungsinya. Dalam mode ini menggunakan model memori

Ruby di mana mampu mendapatkan hasil yang lebih akurat meskipun prosesnya lebih lama.Library python yang disertakan sebagai pengatur utama yaitu ruby_fs.py yang telah diintegrasikan dengan program python lainnya, diantaranya Ruby.py, CpuConfig.py, Cache.py, dan program pendukung.

Parameter pada gem5 yaitu scripts digunakan untuk melakukan input benchmark yang sudah direncanakan. Adapun formatnya berupa .rcS. Pembuatan file .rcS menggunakan generator yang disediakan oleh PARSEC menggunakan bahasa Perl.

Langkah terakhir dari pembangunan simulasi ini adalah mengikutsertakan konfigurasi TOPAZ sebagai model interkoneksi dan topologi.Parameter konfigurasi TOPAZ sendiri sudah terintegrasi dengan library Ruby.py yang terdapat pada Gem5

3.4.2 *Running* Simulasi

Hasil skenario yang telah diuji dapat diamati pada tiga buah file yang tercantum pada directory m5out pada gem5. File tersebut diantaranya ruby.stats merupakan file yang akan menampilkan hasil akurasi running dari Ruby. Kemudian, untuk melihat hasil statistic secara menyeluruh dapat dibuka pada file stats.txt.Adapun hasil waktu yang dihitung oleh TOPAZ mengenai interkoneksi dapat disalin dari console ataupun disimpan pada file berformat .txt.

3.4.3 Analisis Hasil

Analisis hasil dari skenario yang telah diuji berdasarkan file yang berperan ketika proses running simulator. Dari *output* tersebut akan didapatkan detail dari *execution time, latency, throughput, cache miss rate,*dan *average memori access time*.

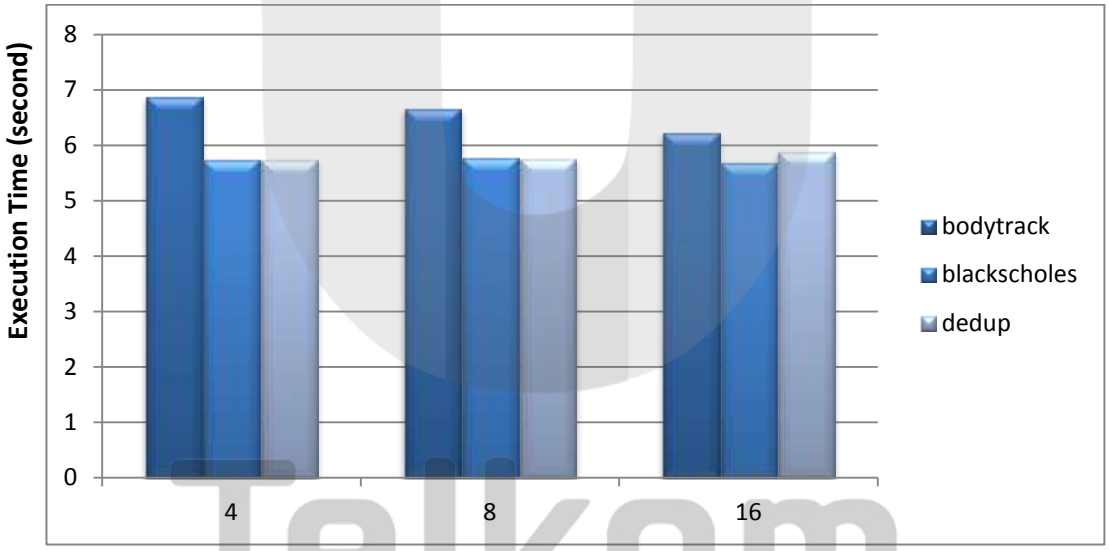
4. Analisis Hasil Pengujian

4.1. Execution Time

Hasil yang didapatkan berdasarkan simulasi dengan parameter execution time menghasilkan hal yang berbeda antara kedua benchmark. Berikut tabelnya

Jumlah CPU	Execution Time in (second)	Execution Time in (second)	Execution Time in (second)
4	6,880047318	5,750080182	5,73812541
8	6,660105529	5,778214282	5,76782761
16	6,230963605	5,695934235	5,88267384
Benchmark	bodytrack	blackscholes	dedup

Tabel 4-1 *Execution Time*



Gambar 4-1 Grafik *Execution Time* dengan jumlah CPU

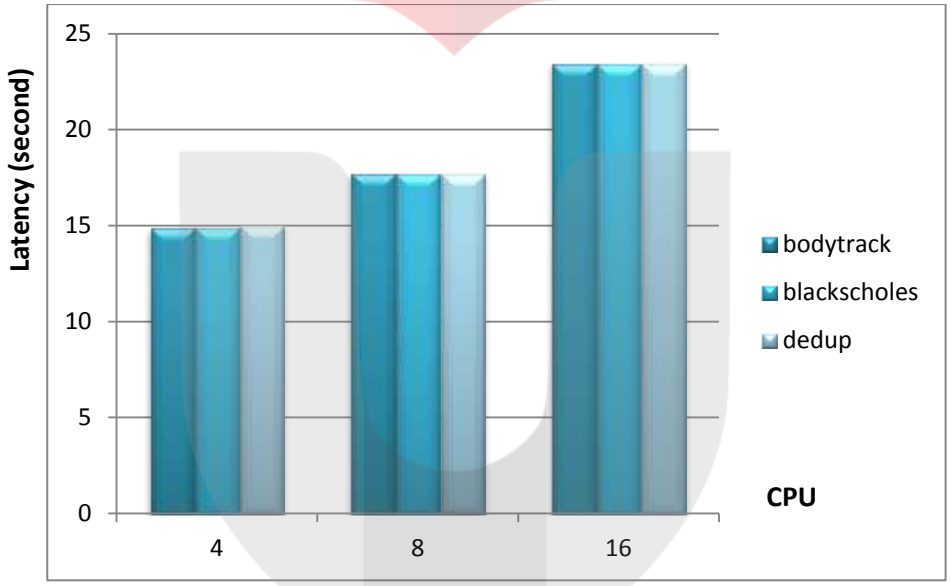
Berdasarkan hasil eksekusi di atas, masing-masing benchmark menunjukkan hasil tersendiri diseusaikan dengan cara kerja thread pada benchmark tersebut. Eksekusi paling singkat ditunjukkan oleh benchmark blachscholes dengan rata-rata perbandingan 32% dibandingkan benchmark yang lain. Hal ini disebabkan oleh sistem kerja yang membagi jumlah thread total sesuai dengan jumlah prosessor yang ada sehingga terbukti paralelisasi pada sistem NUMA berjalan dengan baik untuk benchmark yang memanfaatkan paralel program sebenarnya. Adapun jika ditinjau dari jumlah CPU membuktikan bahwa semakin ditambah prosessor performansi komputasi suatu program dapat berjalan lebih singkat. Adapun eksekusi yang dihadirkan dalam tabel adalah eksekusi secara total pada masing-masing benchmark ketika menjalankan thread

4.2. *Latency*

Berdasarkan hasil simulasi diperoleh hasil latency yang merupakan network latency dari TOPAZ. Berikut tabel dan grafik untuk kedua benchmark

Jumlah CPU	Network Latency	Network Latency	Network Latency
4	14,9128	14,9128	15,0277
8	17,751	17,7415	17,7429
16	23,4741	23,4711	23,4731
	Bodytrack	Blackscholes	Dedup

Tabel 4-2 *Latency* Antar Node



Gambar 4-2 Grafik *Latency* dengan jumlah CPU

.

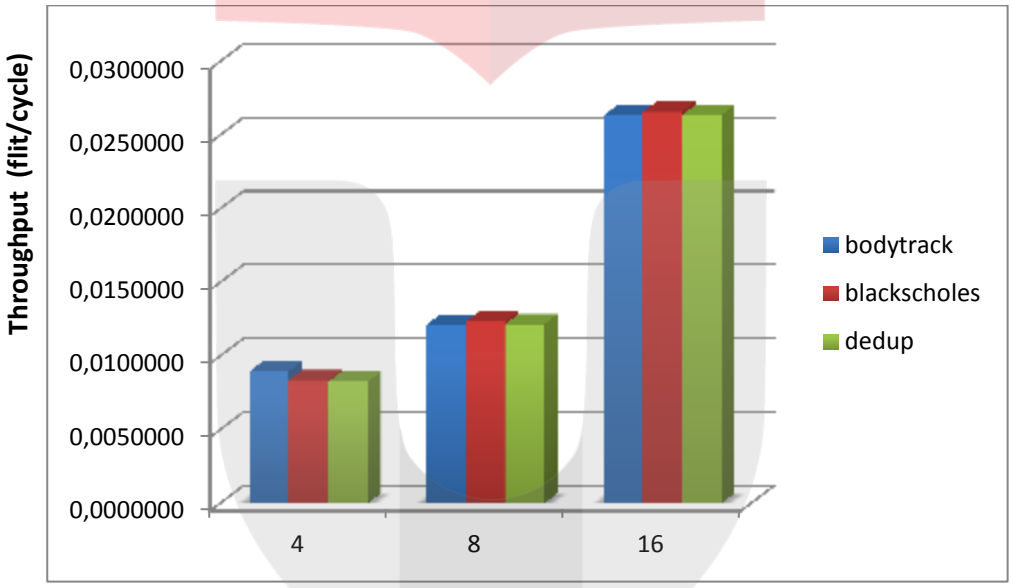
Ditinjau dari parameter latency, semakin banyak jumlah CPU membuktikan bahwa kenaikan latency semakin besar pula. Adapun masing-masing benchmarak menunjukkan pada nominal yang sama di mana 16 CPU memiliki resiko latency dalam suatu jaringan lebih besar sebesar 42% dibandingkan jumlah CPU lain ketika menjalankan ketiga benchmark tersebut. Hal ini jelas dipengaruhi oleh jumlah hop atau node yang harus dilewati oleh suatu thread untuk dijalankan oleh prosessor yang meminta suatu proses tertentu.

4.3. *Throughput*

Berdasarkan hasil simulasi, berikut hasil throughput yang ditampilkan dalam tabel dan grafik

Jumlah CPU	Throughput dalam satuan flit/cycle (bodytrack)	Throughput dalam satuan flit/cycle (blackscholes)	Throughput dalam satuan flit/cycle (Dedup)
4	0,0089692	0,0083714	0,0082548
8	0,0121269	0,0123251	0,0121655
16	0,0263294	0,0265750	0,0263735

Tabel 4-3 *Throughput* dan Jumlah CPU



Gambar 4-3 Grafik *Throughput* dengan jumlah CPU

Ditinjau dari parameter ini, hasilnya sebanding dengan parameter *execution time* di mana pada CPU dengan jumlah 16 buah menghasilkan *throughput* yang maksimal dibandingkan yang lainnya dengan persentase sebesar 56%. Adapun jika dihubungkan cara kerja thread pada masing-masing benchmark hasilnya ada yang berubah pada 4-CPU ketika dijalankan bodytrack dan blackscholes meski tipis pada jumlah core lebih sedikit bodytrack mampu menghasikan penjaminan data sampai ke node yang dituju lebih baik dibandingkan kedua benchmark lainnya. Namun, hasilnya berubah pada benchmark prosessor sebesar 8 dan 16 buah di mana blackscholes memiliki hasil yang maksimal dibandingkan kedua benchmark lain. Hal ini, disebabkan oleh cara kerja pada pembagian thread antara masing-masing benchmark. Bodytrack diserahkan pada pool thread yang melakukan eksekusi threadnya sebelum di eksekusi oleh worker thread sehingga hal ini perlu adanya kerja yang ekstra pada pool thread yang diterapkan pada node (dalam hal ini prosessor) tertentu. Lain halnya, dengan blackscholes yang membagi threadnya secara merata kepada masing-masing prosessor, artinya thread dipastikan sampai ke node hanya konsumsi dari masing-masing thread akan memiliki kadar yang berbeda.