PERANCANGAN MODEL NETWORK PROCESSOR UNTUK ROUTER IPV6 SEDERHANA

Onny Sidiq Gunawan[1)], Agus Virgono[2)]
[1,2)]Departemen Teknik Elektro ITTelkom Bandung
Jl. Telekomunikasi 1 Dayeuh Kolot Bandung 40254 Tlp (022)-7565933
email : agv@ittelkom.ac.id

Abstrak

Kebaruan teknologi tidak bisa diartikan bahwa teknologi tersebut semakin rumit dan "besar", sebagai contoh teknologi IPv6 yang menambahkan format-format baru, memperbarui dan menyederhanakan format-format pada teknologi IPv4. Router merupakan perangkat yang berhubungan langsung dengan terjadinya perubahan-perubahan tersebut. Jika teknologi menjadi lebih sederhana, apakah router harus disederhanakan atau sebaliknya?

Mendesain suatu Network Processor yang berhubungan dengan router adalah suatu hal yang saat ini banyak dilakukan dan hal tersebut berhubungan dengan pendesainan suatu arsitektur komputer.

Kunci teknologi dari desain Network Processor System-On-aChip dapat diperoleh :Frekuensi clock prosesor memiliki pengaruh yang signifikan pada konfigurasi dengan cache SRAM on-chip, Bandwitdh saluran memori yang lebih tinggi meningkatkan konfigurasi untuk kedua SRAM dan DRAM, Konfigurasi cache DRAM yang optimal diperoleh pada rata-rata hanya setengah dari energi pemrosesan konfigurasi cache SRAM, namun, dengan teknologi sekarang, batasan implementasi lain sepertinya membuat mereka sebagai alternatif yang lebih sedikit diinginkan dibanding dengan SRAM.

Keyword: *Network Processor, IPv6, SRAM, DRAM.*

1. PENDAHULUAN

Pada sebuah router, penggunaan sistem *single* prosessor secara umum tidak dapat memenuhi permintaan dari suatu pemrosesan sistem. Hal ini disebabkan karena menimbulkan "celah" yang semakin lebar antara *bandwidth* saluran dengan kecepatan pemrosesan.

Banyak keuntungan yang berkaitan dengan pengintegrasian sistem multi prosesor, memori, dan komponen-komponen I/O dalam suatu sistem *single-chip*, dan mengembangkan apa yang disebut dengan *System-On-a-Chip* (SOC). Keuntungan utamanya adalah kemampuan untuk mencapai performansi yang lebih tinggi dengan menggunakan jumlah *chip* yang sedikit, sehingga mengurangi biaya produksi maupun biaya pasar.

Pada penggunaan multi prosesor dalam suatu sistem memiliki beberapa pertanyaan penting seperti berapa jumlah prosesor yang akan digunakan dan berapa *cache* yang harus disediakan pada sistem tersebut. Hal tersebut penting karena kenyataan bahwa semakin banyak prosesor yang digunakan, jumlah *cache* yang diperlukan semakin kecil, namun semakin kecil *cache* dapat mempertinggi kemungkinan terjadinya kesalahan-kesalahan, bagaimana baiknya ?.

Pengukuran-pengukuran akan dihubungkan dengan performansi model ada dua macam. Yang pertama adalah energi pemrosesan per unit *chip*, dan yang kedua adalah energi pemrosesan total *chip*.

Evaluasi model dilakukan dengan pemberian beban kerja Network Processor pada parameter-parameter desain. Desain yang kemudian diperoleh, dapat dijadikan panduan untuk mendesain arsitektur Network Prosesor selanjutnya.

Pada penelitian ini akan dianalisa spesifikasi dan karakteristik dari Netwok Processor (NP) yang diinginkan, yaitu NP sederhana yang dapat menjalankan algoritma-algoritma yang terdapat dalam protokol routing RIPv2, serta melaksanakan operasi-operasi dasar router seperti : klasifikasi, modifikasi, antrian ,*buffering*, *forwarding* dan lain-lain

Tujun dari penelitian ini adalah untuk menentukan karakteristik Network Processor yang dibutuhkan untuk membangun router IPv6 sederhana dan mencari algoritma-algoritma efisien pada protokol routing RIPv2 yang dapat dioperasikan oleh processor yang dibutuhkan oleh router IPv6 tersebut.

2. TINJAUAN PUSTAKA

Pada sebuah router, penggunaan sistem *single* prosessor secara umum tidak dapat memenuhi permintaan dari suatu pemrosesan sistem. Hal ini disebabkan karena timbulnya celah yang semakin meningkat antara *bandwidth* saluran dengan kecepatan pemrosesan. Namun aliran paket hanya bergantung pada paket-paket yang memiliki aliran yang sama, tidak terjadi ketergantungan dengan aliran paket yang lain sehingga pemrosesannya dapat didistribusikan ke beberapa prosessor.

Terdapat banyak keuntungan yang berkaitan dengan pengintegrasian *multiple* prosesor, memori, dan komponen-komponen I/O dalam suatu *single-chip* dan mengembangkan apa yang disebut dengan SOC (*System-On-a-Chip*).

Keuntungan utamanya adalah kemampuan untuk mencapai performansi yang lebih tinggi dan dengan menggunakan jumlah *chip* yang lebih sedikit dapat mengurangi biaya produksi maupun biaya pasar. Pada penggunaan *multiple* prosesor dalam sistem memiliki suatu keputusan yang penting seperti berapa jumlah prosesor yang akan digunakan dan berapa besar memori/cache yang harus disediakan dalam sistem tersebut. desain arsitektur SOC terdiri dari beberapa mesin prosesor yang independen. Kumpulan dari prosesor tersebut bersama-sama menggunakan saluran memori yang sama untuk akses memori. Saluran I/O digunakan oleh sistem *controller/scheduler* untuk mengirimkan paket yang membutuhkan pemrosesan ke masing-masing prosesor.
Biasanya, sebuah paket pertama kali diterima dan disusun oleh *Transmission Interface* pada port masukan dari sebuah router. Kemudian paket masuk ke sebuah *Packet Demultiplexer* dimana *header* paket akan digunakan untuk menentukan arah tujuan dari paket tersebut. Berdasarkan informasi tujuan paket, *Packet Demultiplexer* kemudian memutuskan pemrosesan apa yang dibutuhkan oleh paket. Ketika sebuah pemroses (prosesor) telah tersedia, paket dan informasinya dikirimkan melalui saluran I/O ke salah satu prosesor yang terdapat pada *chip* Network Processor. Setelah pemrosesan selesai, paket dikembalikan ke *Packet Demultiplexer* dan dimasukan ke antrian untuk kemudian dikirimkan melalui *switching fabric* router untuk dilanjutkan ke masing-masing port keluaran.
Dari sini pertimbangan bahwa optimasi desain sistem *single chip* bergantung pada pemilihan dari :

- Jumlah Prosesor untuk tiap-tiap *cluster* (kelompok).
- Instruksi dan ukuran cache per-prosesor.
- Teknologi memori cache yang digunakan (SRAM, DRAM).
- *Bandwith* dan beban kerja dari saluran memori.
- Ukuran ASIC.
- Beban aplikasi.

Pada Table 1 terdapat daftar parameter yang perlu dipertimbangkan. Prosesor yang digunakan, diasumsikan sebuah prosesor dengan saluran sederhana, *general purpose* RISC.

Table 1. Parameter Sistem

KOMPONEN	SIMBOL	Deskripsi
prosesor	clk_p	processor clock frequency
program	f_{load}	frequency of load instructions
	f_{store}	frequency of store instructions
	mi_c	i-cache miss probability for cache size ci
	md_c	d-cache miss probalility for cache size cd
	$dirty_c$	prob. Of dirty bit set in d-cache of size cd
	compl	complexity (instruction per byte of packet)
cache	c_i	instruction cache size
	c_d	data cache size
	linesize	cache line size of i- and d-cache
	$t_{cache.dram}$	time for cache access (only DRAM)
off-chip memory	t_{mem}	time to access off-chip memory
memory channel	$width_{mchl}$	width of memory channel
	clk_{mchl}	memory channel clock frequency
	ρ	load on memory channel
I/O channel	$width_{io}$	width of I/O channel
	clk_{io}	clock rate of I/O channel
cluster	n	number of processors per cluster
ASIC	m	number of clusters and memory channels
	s(x)	actual size of component x, with x ϵ {ASIC, p, c_i, c_d,io,mchl}

Ukuran dari *chip* Network Processor membatasi jumlah prosesor yang digunakan, jumlah instruksi dan *cache* data per prosesor, dan jumlah saluran memori yang diperlukan. Misal *s(ASIC)* merupakan ukuran dari *chip* Network Processor, $s(p_k)$, $s(c_{ik})$, dan $s(c_{dk})$ berturut-turut adalah ukuran dari sebuah prosesor *k*, *cache* instruksi c_{ik}, dan cache data c_{dk}, dan *s(mchl)* dan *s(io)* adalah ukuran saluran memori dan saluran I/O. Dengan *n* prosesor per *cluster* dan terdapat *m* cluster, semua solusi hari sesuai dengan perhitungan berikut :

$$s(io) + \sum_{k=1}^{n-m} (s(p_k) + s(c_{ik}) + s(c_{dk})) + \sum_{k=1}^{m} s(mhcl) \le s(ASIC) \quad (1)$$

Dengan prosesor yang identik, konfigurasi *cache*, dan saluran I/O rumus diatas menjadi :

$$s(io) + m \cdot [s(mhcl) + n \cdot (s(p) + s(c_i) + s(c_d))] \le s(ASIC) \quad (2)$$

Pertama kali diperhatikan performansi model untuk *single* prosesor dalam hubungan jumlah dari instruksi per detik (IPS) yang dapat dieksekusi oleh prosesor. *Metric* ini sangat tinggi ketergantungannya pada arsitektur prosesor, namun hal ini menangkap efek dari performansi instruksi-instruksi aplikasi dan hierarki memori.
Jumlah dari intruksi-instruksi yang dieksekusi per detik untuk sebuah *single* prosesor, IPS tergantung pada kecepatan *clock* prosesor dan CPI :

$$IPS = \frac{clk_p}{CPI} \quad (3)$$

Sebuah prosesor RISC ideal, dimana tidak terdapat kesalahan *cache*, kesalahan prediksi atau resiko-resiko lainnya, semua instruksi dapat disalurkan tanpa teralihkan dan CPI adalah 1. Pada sistem sebenarnya kenaikan CPI muncul bersamaan dengan timbulnya kesalahan-kesalahan, untuk analisa ini hanya dipertimbangkan kesalahan pada memori pada saat kesalahan-kesalahan lain, seperti kesalahan prediksi cabang-cabang, dan timbulnya relatif jarang dan disebabkan adanya laporan stall (1-2 cycle) pada saluran pendek prosesor RISC dipertimbangkan disini.
Batasan model ini secara mudah dapat dihilangkan jika diperlukan akurasi yang lebih besar. Jika SRAM digunakan sebagai memori *cache*, maka akses ke sebuah *cache* dapat dilakukan oleh satu *clock cycle* prosesor dan tidak ada *stall cycle* yang dikeluarkan oleh keberhasilan *cache*. Jika yang digunakan adalah DRAM untuk *cache* instruksi dan data, maka *clock cycle* saluran dasar naik dari 1 ke $t_{cache.dram} \cdot clk_p$, sehingga :

$$CPI = \begin{cases} 1 + p_{miss} \cdot penalty, & .....SRAM \\ t_{cache.dram} \cdot clk_p + p_{miss} \cdot penalty, & .....DRAM \end{cases} \quad (4)$$

Dimana p_{miss} adalah kemungkinan dari kesalahan sebuah *cache* instruksi atau data. Kemungkinan adalah ketika muncul kesalahan *cache*, tergantung pada aplikasi yang dieksekusi dan parameter yang berhubungan dengan *cache*. Menggunakan frekuensi *load* dan *store* dan probabilitas kesalahan *cache*, dihasilkan oleh :

$$p_{miss} = mi_c + (f_{load} + f_{store}) \cdot md_c \quad (5)$$

Catatan untuk persamaan 5 hanya memperhitungkan kesalahan *cache* hasil dari oleh pembacaan memori. Penulisan ke memori, yang disebabkan oleh menggantikan saluran *cache* yang kotor, tidak menyebabkan prosesor stall. Dengan asumsi tidak ada anggapan untuk saluran memori, kesalahan hukuman dari kesalahan *cache* tergantung pada waktu akses memori dan waktu yang dibutuhkan untuk mentransfer sebuah saluran *cache* memalui bus memori (pada *clock* proseor) :

$$penalty = clk_p \cdot \left(t_{mem} + \frac{linesize}{width_{mchl} \cdot clk_{mchl}} \right) \quad (6)$$

Dengan kesalahan sebuah *cache*, satu ukuran saluran *cache* ditransfer memalui saluran memori. Sebagai tambahan, jika saluran *cache* yang digantikan koro, satu saluran *cache* ditulis kembali untuk memori. *Bandwidth* Memori *off-chip* ditimbulkan oleh sebuah *single* prosesor, $BW_{mchl,1}$, oleh karena itu tergantung oleh pada jumlah instruksi yang dieksekusi dan jumlah dari akses *off-chip* yang ditimbulkan, sehingga :

$$BW_{mhcl,1} = IPS_1 \cdot linesize \cdot \left(mi_c + \left(f_{load} + f_{store} \right) \cdot md_c \cdot \left(1 + dirty_c \right) \right) \quad (7)$$

Bandwidth I/O untuk prosesor tergantung pada kompleksitas dari aplikasi yang sedang berjalan. Kompleksitas dalam artian Network Processor adalah ditentukan sebagai jumlah instruksi yang dieksekusi per byte dari paket data (*header* dan *payload*). Aplikasi dengan kompleksitas yang tinggi memerlukan sedikit *bandwidth* I/O, pada saat waktu yang lebih banyak digunakan oleh pemrosesan. Sehingga *bandwidth* I/O dari sebuah *single* prosesor, $BW_{io,1}$ adalah :

$$BW_{io,1} = 2 \cdot \frac{IPS_1}{compl} \quad (8)$$

Perkalian dengan 2 muncul ketika setiap paket harus dikirimkan terlebih dahulu dari *scheduler* ke *chip* prosesor, dan kemudian nantinya akan kembali keluar ke jaringan.

3. METODA PENELITIAN

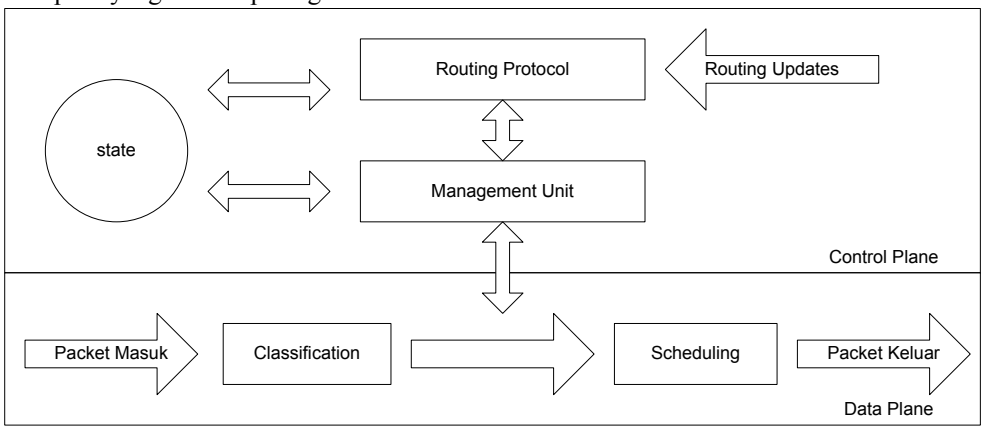
Untuk menganalisa performansi dari Network Processor pada sebuah router maka perlu dilakukan suatu proses desain yang kemudian disimulasikan dengan memberikan parameter-parameter desain. Langkah-langkah yang dilakukan untuk mensimulasikan network processor pada sebuah sistem router, yaitu :

1. Mendesain lingkungan suatu sistem router, dalam penelitian ini lingkungan router yang diinginkan adalah router IPv6 yang sederhana.
2. Mendesain model dari komponen-komponen network processor
3. Menentukan parameter-parameter desain suatu network processor
4. Mensimulasikan beban kerja router ke dalam model network processor yang telah didefinisikan.
5. Pengujian desain network processor dalam suatu router

3.1 DESAIN SIMULATOR NETWORK PROCESSOR (NPSim)

Network processor adalah perangkat yang mampu menyediakan dukungan yang fleksibel pada router untuk beban kerja komunikasi dengan performansi yang tinggi. Mendesain sebuah Network Processor melibatkan pendesainan dan optimasi dari beberapa komponen-komponen dan subsistem, antara lain : (multi)processor, sistem memori, hardware pendukung, interkoneksi dan sistem I/O.

NPSim Adalah simulator yang memodelkan elemen-elemen dari sebuah network processor yang ditanamkan dalam sebuah model lingkungan router berbasis IPv6. Simulator ini dibuat dengan menggunakan Borland Delphi 7 untuk memodelkan sistem tersebut. Network processor dalam simulator ini menangani pemrosesan *data plane* dari fungsi router seperti yang terlihat pada gambar 1.



Gambar 1. Arsitektur router dalam Control dan Data Plane

Elemen-elemen yang dimodelkan dalam Npsim memiliki parameter-parameter yang nilai-nilainya dapat diubah-ubah untuk mendapatkan desain yang diinginkan oleh user.

Kerangka utama dari Npsim disusun dari beberapa model yang masing-masing modul mendefinisikan proses dari router tersebut. Model-model tersebut terdiri dari :

1. **Model Aplikasi** : spesifikasi yang dapat dieksekusi dari fungsi router yang dijelaskan dengan elemen program dan aliran paket. Contoh model aplikasi untuk IP *Fragmentation*, IP *address lookup*, dan *packet classification.*
2. **Model Sistem** : deskripsi dari perangkat pemroses, memori dan interkoneksi dalam sistem yang dimodelkan. Contoh : satu atau lebih prosesor yang dihubungkan pada SDRAM melalui saluran memori berkecepatan tinggi. Kemudian semua elemen dari model aplikasi akan dipetakan ke dalam komponen-komponen sistem.
3. **Model Trafik dan Beban Kerja** : deskripsi dari tipe trafik yang ditawarkan ke sistem. Dan definisi beban kerja yang diberikan kepada model untuk menentukan performansi dari model sistem.

Kerangka pemodelan yang diperkenalkan disini didesain untuk membantu menjawab pertanyaan-pertanyaan yang muncul, seperti :

1. Apakah sistem **S** yang ditentukan mencukupi untuk mendukung aplikasi **W** pada target kecepatan saluran dan jumlah interface ?

2. Jika Tidak, dimanakah *bottleneck*-nya?
3. Jika Ya, dapatkah **S** mendukung aplikasi **W** (yaitu, aplikasi W plus beberapa tugas-tugas baru) dengan batasan sistem yang sama ?
4. Apakah dengan diberikan hardware pendukung dapat meningkatkan performansi sistem relatif terhadap implementasi software pada penerapan tugas-tugas yang sama ?

Kerangkanya disusun dari aplikasi , sistem dan model trafik yang independen yang menjelaskan tentang fungsi router, sistem sumber daya dan trafik paket. Dimana kerangka tersebut diprofilkan dan disimulasikan untuk menentukan aliran paket maksimum yang dapat melewati sistem serta penggunaan sumber daya pada model trafik yang diberikan.
Pendekatan umum dengan memperkirakan karakteristik dari aplikasi secara memuaskan dengan berdasar pada *instruction profiling*. Kemudian menggunakan perkiraan-perkiraan tersebut untuk menentukan penggunaan sumber daya sistem dan perkiraan adanya perebutan *shared resource*.

3.2 MODEL APLIKASI

Aplikasi dimodelkan dengan memprofilkan konfigurasi pada Npsim dan mensimulasikan eksekusi dari aplikasi tersebut pada prosesor target dengan masukan input paket yang akan dijelaskan dalam model trafik.
Profil-profil yang diperoleh dari konfigurasi tersebut menghasilkan beberapa informasi, antara lain :

a. Jumlah instruksi
b. Kelakuan dari *cache*
c. Kontribusi pada CPI

Pada umumnya aplikasi akan disimulasikan pada trafik masukan dan prosesor target. Hal-hal tersebut membantu untuk menentukan performansi dari pemroses paket. Namun *execution time* hanya merupakan bagian dari keseluruhan dari *forwarding time* paket, namun demikian waktu tersebut akan disesuaikan untuk menentukan adanya perebutan *shared resource*. Jika sebuah elemen tidak memiliki *software* (misal : penggunaan *Hardware* pendukung) dan elemen tersebut tidak dapat disimulasikan, maka profil dan ketergantungan terhadap sumber daya dari elemen tersebut harus disediakan secara manual oleh user.
Penelitian yang dilakukan yang akan dijelaskan pada pembahasan selanjutnya hanya menggunakan jumlah instruksi, CPI yang dicapai dan jumlah transaksi *bus* untuk memperkirakan performansi sistem.
Untuk mendapatkan informasi-informasi untuk memodelkan aplikasi-aplikasi yang dibutuhkan untuk menjalankan fungsi router, pada penelitian penulis menggunakan data yang diperoleh dari hasil disimulasi program *benchmark* yang digunakan untuk mengukur performansi dari perangkat network processor.
Parameter-parameter dari sebuah aplikasi pada sebuah software *benchmark* digunakan untuk dapat mewakili dari kebutuhan parameter-parameter yang diinginkan. Dipilih aplikasi benchmark *Commbench* [1] untuk menyediakan informasi dari aplikasi-aplikasi yang dibutuhkan. Disebabkan keterbatasan penulis untuk memperoleh dan menjalankan aplikasi *benchmark* tersebut, maka untuk kepentingan penelitian ini, data-data yang diperoleh diambil dari beberapa jurnal yang menggunakan aplikasi *benchmark* tersebut [2] [3].
Aplikasi-aplikasi dari *Commbench* telah dipilih untuk mewakili beban kerja untuk router-router tradisional (memfokuskan pada pemrosesan header) dan '*active'* router (menjalankan kedua pemrosesan *header* dan *stream*). Aplikasi-aplikasi tersebut dapat dibagi dalam dua group : *Header-Processing Applications* (HPA) dan *Payload-Processing Applications* (PPA).
HPA hanya memproses header-header pada paket yang secara umum aplikasi tersebut lebih sedikit membutuhkan proses komputasi dibandingkan dengan PPA yang memproses keseluruhan data dari paket yang datang. Aplikasi yang digunakan dari *Commbench* diperlihatkan pada table 2.

Table 2: Deskripsi dari aplikasi-aplikasi *Benchmark*

Appikasi	Deskripsi	Tipe
RTR	Radix-tree Routing	HPA
FRAG	Packet Fragmentation	HPA
DRR	Deficit Round Robin fair schedulling	HPA
TCP	TCP Traffic Monitor	HPA
CAST	CAST data encryption	PPA
ZIP	ZIP data compression	PPA
REED	Reed-Solomon forward error corection	PPA
JPEG	JPEG image transcoding	PPA

Lingkup karakteristik dari aplikasi-aplikasi *benchmark* yang dipilih untuk penelitian ini dibatasi pada :

a. Kompleksitas Komputasi
b. Frekuensi Instruksi

c. Performansi dari Cache yang digunakan

Ukuran dari aplikasi dari beberapa dimensi berkisar dari ukuran *source code* sampai jumlah byte yang paling sering digunakan selama proses eksekusi [1]. Ukuran dari *source code* dan hasil kompilasi untuk masing-program ditunjukkan oleh tabel 3.

Tabel 3: Ukuran kode aplikasi-aplikasi *Commbench*

Appikasi	Tipe	Ukuran C code (line)	Ukuran Object Code (byte)
TCP	HPA	19.100	352.000
JPEG	PPA	18.300	260.000
ZIP	PPA	6.500	117.000
RTR	HPA	1.130	16.000
REED	PPA	410	6.900
CAST	PPA	350	19.500
DRR	HPA	100	2.500
FRAG	HPA	100	2.400
Rata-rata		*5.750*	*97.000*

Perhitungan kompleksitas komputasi digunakan untuk membantu menentukan aspek-aspek performansi dari fungsi-fungsi beban kerja yang diberikan. Kompleksitas komputasi didefinisikan dengan $Compl_{a,l}$, sebagai jumlah instruksi per byte yang diperlukan oleh aplikasi *a* beroperasi pada panjang paket *l*.

$$compl = \frac{Instruksi\ yang\ dieksekusi}{ukuran\ packet}$$

Untuk pemrosesan *header*, nilai *l* adalah 64, 576, 1.536 berturut-turut adalah minimum ukuran paket-IP, minimum MTU (Metric Transfer Unit) pada IP, dan ukuran maksimum paket ethernet. Untuk aplikasi *payload processing*, nilai *l* adalah tidak terhingga berdasarkan pada besar paket yang masuk ke sistem

4 HASIL DAN PEMBAHASAN

Untuk mengevaluasi secara tepat dan untuk mendesain sebuah Network Processor perlu dispesifikasi sebuah beban kerja yang sesuai dengan lingkungannya. Hal tersebut telah dilakukan oleh aplikasi *benchmark* CommBench. Aplikasi untuk CommBench dipilih untuk memasukkan keseimbangan antara *header-processing application* (HPA) dan *payload-processing application* (PPA). HPA hanya memroses header-header paket yang secara umum membuat mereka secara komputasi membutuhkan lebih sedikit dibandingkan dengan PPA yang memroses keseluruhan data pada paket.

Sifat-sifat *cache* dari aplikasi *benchmark* juga diukur untuk memperoleh $mi_{c,i}$, $md_{c,i}$, dan $dirty_{c,i}$. Hal tersebut telah dilakukan dengan ukuran *cache* antara 1*k*B sampai dengan 1024*k*B. Untuk tujuan ini, simulator prosesor dan *cache* digunakan. Ada 2 cara yang dihubungkan dengan *write-back cache* dengan *linesize* 32 byte telah disimulasikan. Kecepatan *cache miss* telah dilunasi melalui jalannya program yang lama. Kemudian, dapat diasumsikan untuk mewakili kecapatan *steady-state miss* untuk aplikasi-aplikasi ini.

Mempertimbangkan dua beban kerja untuk mengevaluasi analisa, dipertimbangkan :

- Workload A – HPA : *Header-processing application*
- Workload B – PPA : *Payload-processing application*

Beban kerja diatas adalah sebuah distribusi yang sama dari kebutuhan pemrosesan melalui semua aplikasi yang terdapat pada masing-masing workload. Table 2 menunjukkan kumpulan kompleksitas dan frekuensi *load* dan *store* dari beban kerja. Catatan bahwa kompleksitas dari pemrosesan *payload* adalah naik secara signifikan dibandingkan untuk pemrosesan *header*. Hal tersebut sesuai dengan kenyataan bahwa pemrosesan *payload* memproses keseluruhan *byte* dari *payload* paket dan secara khusus mengeksekusi algoritma *transcoding* yang kompleks. Pemrosesan *header* dilain hal, secara khusus hanya membaca beberapa bidang pada *header* dan melakukan operasi sederhana untuk *lookup* dan pembandingan.

Kecapatan kelompok *cache miss* untuk instruksi dan *cache* data diperlihatkan pada gambar 2. kedua beban kerja mencapai kecapatan *miss* instruksi kurang dari 0.5% untuk ukuran *cache* 8*k*B atau lebih. Kecapatan data *cache miss* untuk beban kerja A juga turun dibawah 0.5% untuk 8*k*B. Untuk beban kerja B, meskipun, kecepatan data *cache miss* turun dibawah 1% untuk 32*k*B atau ukuran *cache* lebih besar.

Table 4. computational complexity and load and store frequencies of workloads

Workload	complw	fload,W	fstore,W
A - HPA	9.1	0.2319	0.065
B - PPA	249	0.1691	0.0595

4.1 EVALUASI

Untuk optimasi Network Processor harus ditentukan suatu lingkup desain yang mencerminkan teknologi ASIC sesungguhnya. Table 3 menunjukan nilai atau jangkauan nilai dari masing-masing parameter sistem yang dipertimbangkan. Untuk ukuran fetur dari komponen diasumsikan menggunakan teknologi .25µ.
Dengan diberikan analisa pada bagian 3 dan beban kerja dan sifat-sifat sistem pada bagian 4, konfigurasi optimal dari Network Processor sekarang dapat ditentukan.

Table 5, system parameters for optimization

Parameter	Value
clk_p	50 MHz … 400MHz
c_i	1kB … 1024kB
c_d	1kB … 1024kB
linesize	32byte
t_{mem}	40ns … 80ns
$t_{cache.dram}$	15ns
$width_{mchl}$	4bit … 64bit
clk_{mchl}	200MHz

Parameter	Value
P	0 … 1
$width_{io}$	up to 64bit
clk_{io}	200MHz
s(ASIC)	100mm^2 … 400 mm^2
s(proc)	2mm^2
s(ci), s(cd)	SRAM : 0.152mm^2 per kB
	DRAM : 0.015mm^2 per kB
s(mchl), s(io)	10mm^2 + width · 0.15mm^2

Optimasi Cluster

Optimasi ini terlihat hanya pada kondigurasi dari *cluster* tanpa mempertimbangkan batasan ukuran *chip* ASIC atau saluran I/O. Di bawah kondisi tersebut, tidak ada lingkup fragmentasi yang muncul dan desain dapat secara mudah diteliti. Untuk kedua beban kerja, dan konfigurasi SRAM dan DRAM, dievaluasi pengaruh dari *bandwidth* saluran memori dan *load*, kecepatan prosesor dan waktu akses memori *off-chip*.
Sebagai sifat-sifat dasar, digunakan *bandwidth* saluran memori, BW_{mchl} = 800MB/s, waktu akses *off-chip* memori, t_{mem} = 60ns, dan kecepatan *clock* prosesor clk_p = 400MHz. dimulai dengan konfigurasi tersebut merubah sifat-sifat untuk melihat efek pasa performansi sistem keseluruhan. Table 4 menunjukan konfigurasi optimal untuk sifat-sifat dasar tersebut.
Untuk beban kerja A, *cache* instruksi sebesar 8kB adalah cukup untuk memperoleh instruksi *cache miss* yang rendah. Untuk beban kerja B, membutuhkan 16kB instruksi *cache*. Sejah tidak ada "lutut" pada kurva *cache miss*, hasil dari optimasi adalah 16kB dan 32kB untuk *cache* data, dimana memperoleh kecapatan *miss* kurang dari 0.3% untuk beban kerja A dan kurang dari 1% untuk beban kerja B. *cache* yang besar tidak memperbaiki *cache miss* secara signifikan, namun membutuhkan area *chip* yang lebih besar.
Beban saluran memori untuk konfigurasi ini antara 69% s/d 79%. Jumlah prosesor per cluster adalah 6 dan 16 pada waktu SRAM muncul, dan kurang lebih 6 kali lebih besar, 40 dan 91 ketika digunakan DRAM. Hasil DRAM membutuhkan kira-kira sepuluh kali dari area SRAM. Walaupun menggunakan kecapatan prosesor yang lebih rendah untuk konfigurasi DRAM, sistem masih dapat memperoleh 50% s/d 75% dari rata-rata konfigurasi ISP A.
Salah satu penelitian penting adalah menaikkan kecepatan prosesor hanya berpengaruh pada konfigurasi *cache* SRAM. Hal ini dapat ditunjukkan pada Gambar 3, dimana ISP $A_{cluster}$ untuk beban kerja dan konfigurasi memori *on-chip* yang berbeda diplotkan melalui beberapa kecepatan prosesor. Pada kasus SRAM, pemrosesan total per area naik bersamaan dengan *clock* prosesor yang meningkat, dimana IPS_1 pada persamaan meningkat. Untuk *cache* DRAM, meskipun, kecepatan efektif prosesor meloncat sesuai dengan waktu, membutuhkan akses DRAM *on-chip*.

Tabel 6. Konfigurasi optimal dari cluster dengan BW_{mchl} = 800MB/s, t_{mem} = 60ns, dan clk_p = 400MHz.

Workload	on-chip memory	n	c_i (kB)	c_d (kB)	ρ	IPS A (MIPS/mm^2)
A – HPA	SRAM	16	8	16	0.72	50.31
B – PPA	DRAM	6	16	16	0.69	25.55
A – HPA	DRAM	91	8	16	0.78	25.2
B – PPA	SRAM	40	16	32	0.79	19.38

Pengaruh dari beban saluran memori, ρ, pada $IPSA_{cluster}$,juga ditunjukkan panjang antrian N_Q, yang berkaitan dengan model M/D/1 dari saluran memori. Jumlah instruksi terbesar yang dieksekusi dapa diperoleh untuk beban saluran memori $\rho = 0.6 \ldots 0.85$, yang dihubungkan dengan wilayah dimana panjang antrian adalah kecil (<3). Ketika semakin kecil beban yang menyebabkan delay antrian rendah, juga membutuhkan area *chip* per prosesor yang lebih sejak prosesor yang lebih sedikit berbagi ukuran saluran yang tetap. Beban yang semakin tinggi meningkatkan delay antrian yang signifikan, dimana pada gilirannya menyebabkan prosesor *stall* sangat lama pada *cache miss*.

$IPSA_{cluster}$ untuk *bandwidth* saluran memori yang berbeda. Untuk konfigurasi SRAM, semakin cepan saluran memori menambah $IPSA_{cluster}$ sekitar 20MIPS/mm^2, dari 100MB/s menjadi 1600 MB/s. Hal ini berkaitan dengan pengurangan waktu transfer untuk saluran *cache*. Peningkatan ini lebih sedikit signifikan untuk konfigurasi DRAM, sejak prosesor beroperasi pada kecepatan yang lebih rendah (meloncat dengan waktu akses *on-chip* DRAM) dan pengurangan pada waktu transfer memori telah berkurang akibat pengaruh pada total CPI_{DRAM}.

Tipe yang berbeda dari memori *off-chip* dengan waktu akses yang berbeda juga dapat digunakan. Pengaruh dari waktu akses memori t_{mem} pada energi pemrosesan per area adalah sangat terbatas. $IPSA_{cluster}$ total menurun secara datar (rendah) untuk memori yang lebih lambat (2-5% untuk t_{mem} = 80ns di atas t_{mem} = 40ns), namun waktu akses memori hanya komponen kecil pada *penalty cache miss* (persamaan 6). Yang lebih penting adalah *bandwitdh* memori sebenarnya dan beban pada saluran memori ditunjukkan oleh Gambar 4 dan 5.

Optimasi ASIC

Untuk optimasi ASIC, batasan ukuran ASIC harus dipertimbangkan sama seperti saluran I/O. untuk mengilustrasikan lingkup optimalisasi, IPS_{ASIC} yang optimal dari sebuah ASIC 400mm^2 dengan beban kerja A dan *cache* SRAM. *Bandwitdh* saluran memori adalah 800MB/s, kecepatan *clock* prosesor adalah 400MHz, dan waktu akses memori *off-chip* adalah 60ns. Dapat dilihat dengan jelas optimasi untuk 8kB instruksi dan 16kB *cache* data. Konfigurasi cache yang berubah-ubah secara signifikan dari optimal tersebut menunjukkan langkah kemunduran pada keseluruhan performansi. Hal tersebut menekankan pentingnya dari sebuah sistem yang dikonfigurasi secara optimal.

Table 7 Konfigurasi ASIC dengan maximum processing power (s(ASIC) = 400mm^2).

Workload	Cache Type	m	n	c_i (kB)	c_d (kB)	P	t_{mem} (ns)	BW_{mchl} (GB/s)	clk_p (MHz)	I/O pins	IPS (MIPS)
A - HPA	SRAM	3	20	8	8	0.8	40	1.6	400	365	19700
B - PPA	SRAM	6	10	4	8	0.86	40	1.6	400	389	12600
A - HPA	DRAM	1	145	8	16	0.63	40	1.6	≥673	148	9450
B – PPA	DRAM	1	64	16	16	0.92	40	1.6	≥673	131	8050

Maksimum IPS_{ASIC} ditemukan pada sembarang konfigurasi sistem yang ditunjukkan pada Table 5 untuk kedua beban kerja dan teknologi *cache*. Hal tersebut tidak mengherankan bahwa optimal diperoleh untuk sifat-sifat pada teknologi tercepat pada semua kategori (BW_{mchl} = 1.6 GB/s, t_{mem} = 40ns, dan clk_p =400MHz untuk *cache* SRAM). Kapasitas pemrosesan maksimum hampir 20000 MIPS untuk sebuah konfigurasi *cache* SRAM dengan 8kB untuk data dan 8kB untuk instruksi. Konfigurasi *cache* DRAM, lagi-lagi, mencapai sekitar setengah dari performansi *cache* SRAM. Meskipun, bahwa konfigurasi optimal DRAM yang diperoleh tidak mendapatkan catatan faktor lain yang akan membuat desain tidak dapat dikerjakan dengan mudah. Sebagai contoh, pembebanan saluran elektrik akan menghalangi dimilikinya 145 prosesor yang berasosiasi dengan saluran memori tunggal. Meskipun, dengan implementasi DRAM yang bertambah baik, model akan mengijinkan analisa dari konfigurasi alternatif.

Salah satu catatan yang penting pada table 7 adalah jumlah maksimum dari pin I/O (yaitu jumlah pin data untuk saluran memori dan saluran I/O) tidak melampaui 400. bahkan ketika menambah pin yang perlu untuk pensinyalan, kontrol dan *power*, jumlah pin tidak lebih diatas teknologi pengepakan.

5 KESIMPULAN

Sebuah model analitik dari sistem yang telah ditunjukkan yang mencerminkan *power* komputasi per area dari sebuah *cluster* dan total *power* pemrosesan dari sebuah ASIC. Menggunakan statistik aplikasi dari pengukuran telekomunikasi, sebuah beban kerja telah ditentukan dan digunakan pada proses optimalisasi. hasil dari *cluster* yang bermacam-macam dan konfigurasi ASIC telah ditunjukan dan dianalisa.

Berikut kunci teknologi dari desain System-On-aChip dapat diperoleh :

- Frekuensi *clock* prosesor memiliki pengaruh yang signifikan pada konfigurasi dengan *cache* SRAM *on-chip*. Untuk *cache* DRAM *on-chip*, tidak meningkatkan performansi untuk kecepatan *clock* lebih tinggi dari kecepatan akses memori.

- *Bandwitdh* saluran memori yang lebih tinggi meningkatkan konfigurasi untuk kedua SRAM dan DRAM. Pengaruhnya adalah lebih tinggi untuk konfigurasi SRAM. Beban saluran memori optimal untuk *cache* SRAM adalah antara 65% sampai 85% dan untuk *cache* DRAM antara 70% sampai 92%.
- Konfigurasi *cache* DRAM yang optimal diperoleh pada rata-rata hanya setengah dari energi pemrosesan konfigurasi *cache* SRAM, namun, dengan teknologi sekarang, batasan implementasi lain sepertinya membuat mereka sebagai alternatif yang lebih sedikit diinginkan dibanding dengan SRAM.
- Trend tradeoff adalah sama untuk kedua beban kerja dipertimbangkan. Hal ini menandakan bahwa mereka berdiri sendiri dari beban kerja yang khusus dimana suatu sistem dioptimasi.

6 DAFTAR PUSTAKA

Wolf, Tilman.; Franklin, Mark, 2000, *Commbench – A Telecommunications Benchmark for Network Processor,* Proc. Of IEEE International Symposium on Performance Analisys of Systems and Software (ISPASS), Austin, TX, Apr. 200, pp 154-162

Wolf, Tilman, 2002, *Design TradeOffs for Embedded Network Processor,* Springer Verlag, Karlsruhe, Germany, Apr. 2002, pp 149-164

Wolf, Tilman, 2001, *Design Issues for high performance active router,* IEEE Journal on Selected Area in Communication, Volume 19, Issue 3, Mar 2001, pp 404-409