EKSPLOITASI INSTRUCTION-LEVEL PARALELLISM (ILP) PADA UNIPROCESSOR

Kuspriyanto, Rustam Effendi
Departemen Teknik Elektro, Institut Teknologi Bandung
email :kuspriyanto@yahoo.com, rustamfellowship@yahoo.com

Abstrak

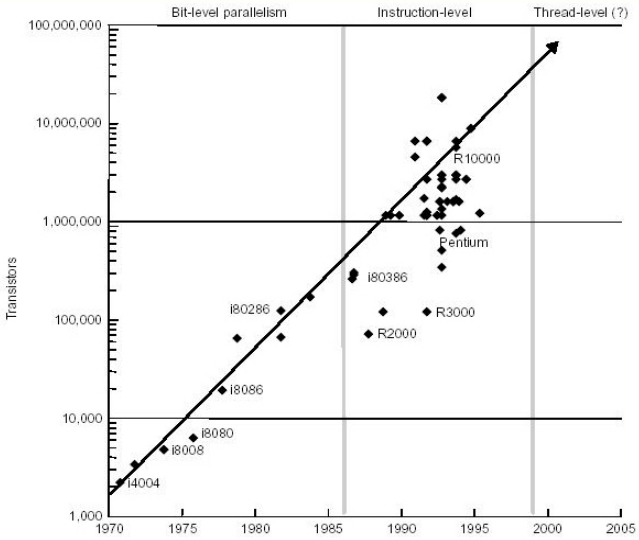
Saat ini para ilmuwan melakukan banyak penelitian dalam rangka meningkatkan performa komputer. Beberapa diantaranya mengkhususkan diri dalam mengembangkan parallelism baik pada Bit-Level Parallelism, Instruction-Level Parallelism maupun dengan konsep baru yang disebut Thread-Level Parallelism. Dengan dua sudut pandang yang berbeda, Instruction-Level Parallelism menghadapi kendala yang dalam eksploitasinya. Ketika "perseteruan" antara arsitektur dengan multiprocessor dengan perancangan uniprocessor yang sangat cepat masih ada, Instruction-Level Parallelism dapat mengambil peran penting untuk mengoptimisasi keduanya.

1. PENDAHULUAN

Dengan sudut pandang software, Instruction-level parallelism memungkinkan suatu bagian dari instruksi-instruksi yang diperoleh dari suatu program sequential diparalelisasi untuk eksekusi pada banyak unit fungsional pipeline. Hal ini secara mendalam mempengaruhi perancangan banyak institusi perancang microprocessor dan compiler-compilernya. Sampai saat ini pengembangan ILP dianggap masih belum sempurna, dengan demikian penelitian masih berlanjut untuk menemukan optimisasi yang lebih baik untuk digunakan dalam rangka peningkatan performa komputer.

2. PARADIGMA PARALLELISM

Sejak tahun 1970 sampai dengan tahun 1985, paradigma parallelism dimulai dengan perkembangan arsitektur komputer pada level hardware ditandai dengan dibuatnya microprocessor 4 bit, 8 bit, hingga 16 bit. Beberapa ilmuwan menyebut masa ini dengan era *Bit-Level Parallelism*. Selanjutnya, sejak tahun 1985 mulai dikembangkan arsitektur komputer pada level Instruksi yang dikenal dengan era *Instruction-Level Parallelism* sampai hari ini.



Gambar 1. Perkembangan Paralelisasi

Meskipun era ini belum akan ditinggalkan oleh para pengembang, telah muncul suatu paradigma baru mengenai suatu teknologi parallelism mulai dikembangkan yaitu Thread-Level Parallelism.

3. INSTRUCTION-LEVEL PARALLELISM

Walaupun riset-riset terus dilakukan untuk meningkatkan kecepatan clock, sampai saat ini institusi-institusi pengembang microprocessor mendapatkan kenyataan bahwa pendekatan teknologi yang dilakukan masih tetap terbatas pada waktu siklus processor. Sehingga dimulailah pengembangan teknik optimisasi ILP untuk meningkatkan performa komputer dengan konsep umum pengeksekusian lebih banyak instruksi per satu siklus. Untuk mengeksekusi lebih banyak instruksi dalam satu siklus, diperlukan penambahan unit-unit fungsional unit integer, unit floating point, unit load/store dan unit-unit lain. Peluang diterapkannya teknik optimisasi ILP ini pada keadaan di mana beberapa instuksi dalam satu rangkaian (*sequence*) tidak bergantung terhadap hasil eksekusi instruksi yang lain. Sehingga langkah awal yang harus dilakukan dalam penerapan teknik ini adalah menemukan rangkaian instruksi mana yang sesuai dengan keadaan yang telah disebutkan sebelumnya.
Suatu kenyataan yang menggembirakan mengenai optimisasi pada ILP adalah adanya kenyataan bahwa dengan Instruction-Level Parallelism performa komputer meningkat menjadi dua kali lipat dalam 1,5 tahun sejak tahun 1985.

Kunci keberhasilan ILP adalah pada eksploitasi parallelism tanpa meminta programmer menulis ulang program-program yang sudah ada dengan memberikan kemampuan untuk melampaui kecepatan eksekusi dari instruksi-instruksi secara individual. Hal ini akan memberikan kemampuan otomatisasi pada ILP untuk mengoptimisasi kinerja komputer.

Meskipun banyak pembahasan arsitektur komputer mengenai multiprocessor, aplikasi-apliasi yang ada saat ini masih merupakan pemrogramman sequential, dan banyak diantaranya tidak akan pernah ditulis ulang kembali.

4. MACHINE PARALLELISM

Kita harus dapat membedakan konsep machine parallelism dengan ILP. ILP ada ketika instruksi-instruksi dalam suatu bagian dalam rangkaian (sequence) bebas dari bagian yang lainnya. Sedangkan machine parallelism adalah ukuran dari kemampuan processor untuk memanfaatkan optimisasi ILP. Machine parallelism ditentukan oleh jumlah instruksi yang dapat di ambil (*fetch*) dan dieksekusi (execute) pada saat yang besamaan (jumlah parallel pipeline) dengan kecepatan mesin mutakhir di mana processor digunakan untuk menemukan instruksi-instruksi yang bebas (*independent*). Ada beberapa keterbatasan dalam teknik paralelisasi yang harus dapat dipecahkan dengan ILP yaitu:

a. Instruksi-instruksi percabangan menghalangi determinasi dari instruksi-instruksi untuk dieksekusi (*control dependences*)
b. Analisis yang tidak sempurna mengenai pengalamatan memori menghalangi penataan ulang operasi-operasi memori (*ambiguous memory dependences*)
c. Aliran data program (*algorithm*) menghalangi parallelism (*true dependences*)
d. Memperuncing dampak tersembunyi dari kebergantungan (*dependences*)

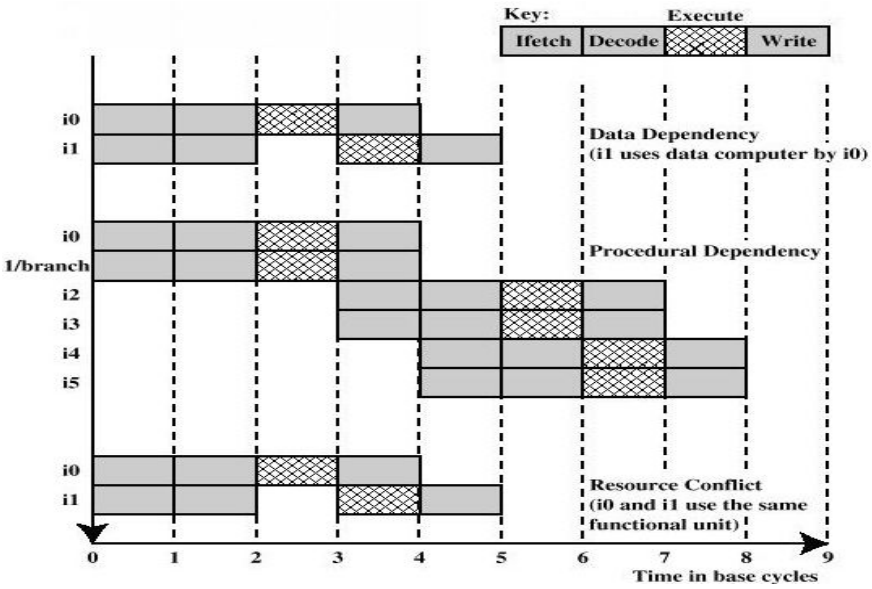
Sebagai contoh dapat diperhatikan pada dua contoh kode dengan penggunaan beberapa register dalam satu sequence berikut ini.

```
Load R1 ← R2          Add R3 ← "1"
Add R3 ← R2, "1"      Add R4 ← R3, R2
Add R4 ← R4, R2       Store [R4] ← R0
```

Dapat dilihat bahwa tiga instruksi di sebelah kiri bebas, di mana pada masing-masing instruksi, register yang satu tidak memiliki ketergantungan terhadap register lainnya dalam satu rangkaian. Ini berarti ketiga instruksi ini dapat dieksekusi dalam waktu yang bersamaan dalam suatu paralelisasi. Sedangkan pada tiga instruksi di sebelah kanan, instruksi ke-dua tidak dapat dieksekusi bersama-sama dengan instruksi sebelumnya karena jika hal ini dilakukan

akan terjadi kesalahan operasi dimana isi register R3 belum terisi nilai yang "benar". Keterbatasan yang sama juga terjadi pada instruksi ke-tiga di mana instruksi tidak dapat dieksekusi bersama-sama dengan instruksi sebelumnya karena jika dilakukan akan mengakibatkan penggunaan sumber (*resources*) yang sama yaitu register R4.

Beberapa kendala kebergantungan lain yang berkaitan dengan penerapan *Instruction-Level Parallelism* pada satu processor (*uniprocessor*) dapat digambarkan sebagai berikut.



Gambar 2. Akibat dari Kebergantungan (*Dependencies*)

Machine Paralellism dapat diimplementasikan pada sistem yang menggunakan satu processor (*uniprocessor*) dan banyak processor (*multiprocessor*).

5. UNIPROCESSOR VS. ON-CHIP MULTIPROCESSOR

Dalam arsitektur *on-chip multiprocessor* dilakukan pengurangan unit-unit fungsional pada setiap processornya dengan maksud untuk melakukan pendistribusian tugas pada setiap processornya. Berbeda dengan arsitektur *on-chip multiprocessor*, dalam arsitektur uniprocessor, unit-unit fungsionalnya berjumlah lebih banyak.

Jika penambahan processor pada suatu arsitektur pada titik tertentu tidak lagi memberikan peningkatan performa yang signifikan, maka perlu dipikirkan suatu teknik lain yang sekiranya dapat menjawab tantangan ini.

Tabel 1. Laporan peningkatan kecepatan

Sumber	Kecepatan
Horst, Harris, and Jardine [1990	1,37
Wang and Wu [1988]	1,70
Smith, Johnson, and Horowitz [1989]	2,30
Murakami et al. [1989	2,55
Chang et al. [1991]	2,90
Jouppi and Wall [1989]	3,20
Lee, Kwok, and Briggs [1991]	3,50
Wall [1991]	5
Melvin and Patt [1991]	8
Butler et al. [1991]	17+

Processor *Superscalar* adalah processor yang mengimplementasikan optimisasi ILP pada arsitektur sequential. Tidak ada bukti yang dapat dipakai mengenai kemungkinan diterapkannya paralelisasi, sehingga penerapan ILP harus dilakukan oleh hardware, di mana selanjutnya harus dibangun suatu perencanaan untuk aksi-aksi eksploitasi paralelisasi. Tabel 1 adalah laporan mengenai peningkatan kecepatan mesin Superscalar.

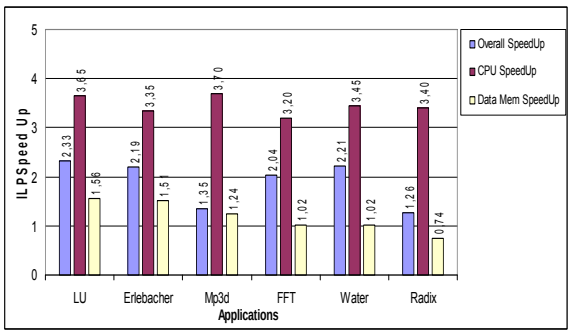
Processor *Very Long Instruction Word (VLIW)* adalah suatu contoh arsitektur di mana program memiliki informasi terinci yang diharapkan untuk paralelisasi. Program mengidentifikasi kemungkinan paralelisasi dalam program dan mengkomunikasikannya kepada hardware dengan menyebutkan operasi mana yang bebas dari operasi lainnya. Informasi ini langsung diberikan untuk hardware, di mana selanjutnya tanpa pengecekan lebih lanjut operasi-operasi ini akan mulai dieksekusi pada siklus yang sama.

Explicitly *Parallel Instruction Computing (EPIC)* [4] adalah model arsitektur yang merupakan suatu evolusi dari VLIW di mana telah mengadopsi banyak ide-ide terbaik dari processor superscalar.

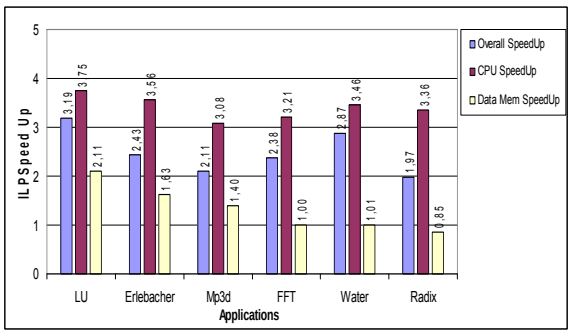
Pada Tabel 2 ditunjukkan hasil suatu riset mengenai penerapan optimisasi ILP pada mutiprocessor dan uniprocessor yang menggunakan program simulasi yang bernama Rice Simulator for ILP Multiprocessors (RSIM). RSIM memodelkan processor, sistem memori, dan jaringan secara detail beserta semua isinya (unit fungsionalnya). Kode-kode operasi dan sub sistem cache untuk pengujian model processor ini melakukan simulasi siklus-persiklus dan terhubung dengan simulator pengendali aksi-aksi untuk model jaringan dan model memori. Aplikasi ini di-compile dengan SPARC V9 gcc yang dimodifikasi untuk mengurangi terjadinya penundaan pada percabangan (branch) dan dibatasi untuk kode 32 bit. Berikut adalah beberapa aplikasi yang digunakan untuk beban pengujian model processor ini di mana aplikasi-aplikasi ini mempunyai karakteristik yang berbeda satu sama lainnya. Hasil pengujian dengan simulasi ini diperlihatkan pada Gambar 3 dan 4.

Tabel 2. Hasil riset penerapan optimisasi ILP pada mutiprocessor dan uniprocessor

Aplikasi	Jumlah Siklus
LU	$1{,}03 \times 10^8$
FFT	$3{,}67 \times 10^7$
Radix	$3{,}15 \times 10^7$
Mp3d	$8{,}82 \times 10^6$
Water	$2{,}68 \times 10^8$
Erlebacher	$7{,}62 \times 10^7$



Gambar 3. Performa ILP pada multiprocessor



Gambar 4. Performa ILP pada uniprocessor

Berdasarkan Gambar 3 dan 4, peningkatan performa ILP pada *multiprocessor* lebih kecil dibandingkan pada *uniprocessor*.

6. EKSPLOITASI INSTRUCTION-LEVEL PARALLELISM

Usaha untuk meningkatkan performa sistem pada sistem processor tunggal secara umum telah fokuskan dengan menambah kemampuan processor. Dengan pendekatan ini, perancangan processor telah difokuskan pada pembuatan processor yang memungkinkan untuk mengeksekusi instruksi-instruksi lebih cepat melampaui kecepatan tinggi clock, ILP dan cache serta kecepatan akses memori. Sehingga untuk optimisasi ILP yang lebih sempurna diperlukan teknik yang dapat memanfaatkan kemampuan yang lebih dari processor cepat ini. Satu teknologi terbaru untuk menjawab tantangan ini adalah teknologi hyper-threading. Teknologi ini meningkatkan performa sistem dengan mengijinkan aplikasi yang mempunyai banyak aliran berpikir multiple thread untuk dijalankan pada sebuah processor tunggal pada suatu waktu dan berbagi sumber daya. Secara konsep, suatu aplikasi multiple thread adalah aplikasi yang mempunyai kemampuan memilah program menjadi tugas-tugas menjadi beberapa bagian kecil yang terdiri dari rangkaian instruksi-instruksi sehingga dapat dieksekusi secara paralel. Hal ini biasanya dikenal dengan Thread-Level Parallelism. Secara arsitektur, suatu processor dengan teknologi hyper-threading membentuk dua processor secara logic di mana setiap processornya secara individu mampu dihentikan, diinterupsi, dan dapat langsung mengeksekusi suatu thread tertentu. Setiap processor logic mengatur set bagian arsitektur yang utuh. Bagian arsitektur yang utuh secara logic ini terdiri atas register-register, register APIC (advanced programmable interupt controller), dan beberapa bagian register mesin lain. Dari sudut pandan software, jika bagian arsitektur yang secara logic ini digandakan, processor seakan-akan menjadi dua processor. Jumlah transistor untuk penyimpanan bagian arsitektur logic akan menjadi pecahan yang kecil dari total keseluruhan transistor. Processor logic ini akan berbagi sumber daya procesor secara fisik seperi cache, unit eksekusi, dan bus. Setiap processor logic mempunyai pengendali interupsi sendiri dan ditangani sendiri hanya oleh processor logic.

Mengenai optimisasi ILP, ada beberapa fase yang dilakukan yang erat kaitannya dengan compiler.

a. Analisis Program
   Fase ini menampilkan aliran control (*control flow*), aliran data (*data flow*), dan analisis kebergantungan data (*data dependece analysis*) untuk mengetahui pada bagian mana dari program yang memiliki peluang untuk di paralelisasi.
b. Restrukturisasi Program
   Pada fase ini dilakukan peralihan (*converting*) dari program input ke suatu bentuk kode program paralel di mana instruksi-instruksinya dapat dieksekusi secara paralel..
c. Pembangkitan Kode Paralel
   Pada fase ini dilakukan peralihan (*converting*) kode secara serial menjadi kode dengan banyak thread (*multithreaded*)

Di dalam optimisasi ILP dikenal suatu penjadwalan yang diimplementasikan pada suatu modul penjadwal (scheduler) yang dapat di digabungkan ke dalam suatu program preprocessor untuk menghasilkan suatu penjadwalan eksekusi statik (static scheduling). Keputusan scheduler akan direpresentasikan sebagai suatu sequence dari persoalan instruksi. Setiap persoalan menyatakan permulaan dari suatu tugas atau instruksi. ILP juga menggunakan penjadwalan dinamik (dynamic scheduling).

Dua macam penjadwalan yang digunakan dalam ekspoitasi ILP [6] adalah sebagai berikut:

a. *Static scheduling* yang terdiri atas :
   - *simple scheduling*
   - *loop enrolling*
   - *loop enrolling + scheduling*
   - *software pipelining*
b. *Dynamic scheduling* yang terdiri atas :
   - *out of order execution*
   - *dataflow computers*

Pada level perangkat keras, paralelisasi bertautan (*sychronous parallelism*) dapat menaikkan tingkat produksi. Hal ini terutama bermanfaat jika program melakukan operasi-operasi yang sama pada semua komponen dari struktur umum. Pada *multiprocessor* penggunaan *synchronous parallelism* dimaksudkan untuk meningkatkan eksekusi dari jenis-jenis masalah ini.

7. KESIMPULAN

Parallelism masih akan terus dikembangkan baik pada tingkat *Bit-Level Parallelism*, *Instruction-Level Parallelism* maupun *Thread-Level Parallelism*. Peningkatan performa suatu arsitektur dengan *Machine Parallelism* bukan berarti bahwa arsitektur *Uniprocessor* akan ditinggalkan begitu saja, melainkan sebaliknya dapat dilakukan optimisasi untuk peningkatan performanya. Peningkatan performa suatu arsitektur dapat terus dipacu dengan dukungan teknologi *Hyper-Thread Parallelism* serta pengembangan terus-menerus terhadap teknik-teknik di dalam optimisasi *Instruction-Level Parallelism* baik mengenai analisis program, restrukturisasi program, pembangkitan kode paralel, penjadwalan (*static scheduling* dan *dynamic scheduling*), prediksi percabangan (*branch prediction*), serta teknik-teknik lain di dalamnya. Masalah terbesar penggunaan *multiprocessor* secara fisik adalah ketika biaya yang diperlukan untuk suatu arsitektur multiprocessor tidak cukup sepadan dengan performa yang ditingkatkannya. Sehingga pengembangan paralelisasi pada arsitektur *uniprocessor* sangat logis untuk dipertahankan.

DAFTAR PUSTAKA

[1] Laxmi Narayan Bhuyan, **"Parallel Processing"**, http://www.cs.ucr.edu/~bhuyan/CS213/ 2004/ LECTURE1.ppt
[2] Michael S. Schlansker, B. Ramakrishna Rau, **"EPIC: An Architecture for Instruction-Level Parallel Processors"**, www.hpl.hp.com
[3] Richard Y. Kain, **"Advanced Computer Architecture"**, International Edition, Prentice Hall, 1996.
[4] Vijay S. Pai, Parthasarathy Ranganathan, and Sarita V. Adve, **"The Impact of Instruction-Level Parallelism on Multiprocessor Performance and Simulation Methodology"**, International Edition, Prentice Hall, 2003.
[5] William Stallings, **"Computer Organization and Architecture, Designing for Performance"**, Fifth Edition, Prentice Hall, 1996.
[6] *……,* **"Intel Hyper-Threading Technology"**, *Technical User's Guide*, http://www.intel.com, 2003.
[7] *……,* **"Why Parallel Architecture?"**, http://personal.denison.edu/~bressoud/ cs400-f04/lectures/01_WhyParallel.pdf
[8] *……,* **"Microprocessor Evolution: Past, Present, and Future"**,http://www.mtl.t.u-tokyo.ac.jp/~sakai/sohi/ talk1.pdf
[9] *……,* **"Exploitation of Instruction-Level Parallelism"**, http://www.capsl.udel.edu/ courses/eleg652/2004/ slides/Topic3-652.pdf