Penggunaan Algoritma Paralel dalam Optimasi Prosesor Multicore

Rafi Ramadhan 13512075
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13512075@std.stei.itb.ac.id

***Abstract*—Algoritma merupakan hal yang fundamental dalam dunia informatika. Hingga saat ini, jenis algoritma yang umum digunakan adalah algoritma sekuensial. Namun, kemajuan teknologi memerlukan algoritma yang lebih efisien dari segi kompleksitasnya. Algoritma paralel mungkin bisa menjadi salahsatu solusi. Hanya dengan satu perintah, algoritma ini mampu menyelesaikan beberapa hal sekaligus. Oleh karena itu, penulis ingiin membahas algoritma paralel secara detil beserta aplikasinya dalam prosesor multicore.**

Kata kunci: Algoritma, Algoritma Paralel, Kompleksitas, dan Prosesor Multicore .

I. PENDAHULUAN

1.1 Algoritma Paralel

Kata paralel dalam konteks 'algoritma paralel' menjelaskan bahwa algoritma tersebut menyelesaikan sebuah permasalahan tidak secara bertahap. Lain halnya dengan algoritma sekuensial. Untuk menyelesaikan persoalan, ada tahapan-tahapan yang harus dilalui satu per satu. Misalnya saja dalam algoritma penjumlahan larik dengan n buah elemen. Algoritma sekuensial akan melakukan penjumlahan dengan menelusuri seluruh elemen larik sedangkan algoritma paralel memiliki cara lain untuk melakukannya. Sebagai contoh, dimisalkan setiap elemen berindeks genap dipasangkan dengan elemen setelahnya yang berindeks ganjil kemudian dijumlahkan. $N_{[0]}$ dengan $N_{[1]}$, $N_{[2]}$ dengan $N_{[3]}$, dan seterusnya sehingga terbentuk urutan baru sebanyak (n/2) elemen. Jika dibandingkan, langkah dalam algoritma sekuensial akan diulang sebanyak n-1 kali sedangkan proses dalam algoritma paralel berulang sebanyak $^{2}\log n$ kali. Jelas terlihat bahwa untuk sebuah persoalan yang sama, algoritma paralel memerlukan waktu yang lebih singkat daripada algoritma sekuensial.

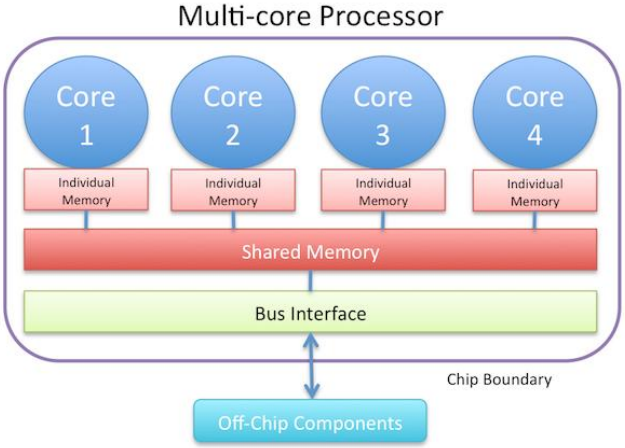
Munculnya algoritma paralel merupakan salahsatu akibat dari adanya hukum yang diperkenalkan oleh Gordon E. Moore pada tahun 1975 (Hukum Moore). Hukum ini menyatakan bahwa tingkat kompleksitas mikroprosesor akan meningkat dua kali lipat dalam jangka waktu 18 bulan. Sekarang, hukum Moore dijadikan parameter untuk perkembangan prosesor. Semua industri yang bergerak di bidang ini berusaha untuk mewujudkan hukum Moore dalam produk ciptaannya. Dengan meningkatnya kompleksitas, kecepatan komputasi pun meningkat. Di sinilah algoritma yang tepat diperlukan untuk melakukan optimasi terhadap prosesor.

Masalah yang belum terselesaikan hingga saat ini adalah bentuk pemodelan dari algoritma paralel. Tidak seperti komputasi sekuensial, pemodelan komputasi paralel lebih rumit karena organisasinya yang lebih beragam. Sebagai usaha untuk menemukan model yang tepat, para ahli melakukan penelitian secara besar-besaran. Meskipun belum ada kesepakatan mengenai bentuk pemodelan yang tepat, setidaknya ada dua jenis pemodelan yang dianggap cukup baik. Pemodelan pertama adalah multiprocessor model sedangkan pemodelan kedua adalah work-depth model.

1.2 Prosesor *multicore*

Istilah prosesor mungkin sudah tidak asing lagi di telinga kita. Prosesor adalah sebuah komponen yang menjadi 'otak' dari sebuah sistem operasi komputer. Komponen ini mengatur seluruh proses dalam komputer sehingga komputer mampu menjalankan instruksi yang diberikan pengguna.

Lebih dari 30 tahun, industri komputer terus berusaha untuk meningkatkan kecepatan kinerja prosesor unicore. Namun, hal tersebut tidak akan berkembang lebih jauh karena adanya keterbatasan. Ukuran transistor, disipasi panas, dan daya adalah faktor yang membatasi perkembangan tersebut. Hal



inilah yang menjadi latar belakang terciptanya prosesor multicore.

Prinsip dari multicore adalah sebuah IC (integrated circuit) yang dilengkapi dengan beberapa prosesor independen yang disebut *core*.
Gambar di atas adalah ilustrasi dari arsitektur sebuah prosesor multicore.

Prosesor multicore memiliki fungsi yang beragam dalam komputasi. Contoh penerapan prosesor multicore tersebut dapat dilihat di CPU multicore, GPU(Graphic Processing Unit), dan FPGA(Field Programmable Gate Array). Pada awalnya, GPU didesain khusus untuk grafik tiga dimensi. Namun, pada awal tahun 1999, GPU juga diterapkan untuk aplikasi komputasi ilmiah lainnya. FPGA adalah perangkat keras khusus yang dapat diprogram untuk menggunakan beberapa mikroprosesor dalam satu *board*. Contoh dari prosesor multicore juga dapat kita lihat dalam bentuk lain, seperti DSP(Digital Signal Processor), konsol *games*, dan perangkat keras jaringan.

II. TEKNIK DAN PEMODELAN ALGORITMA PARALEL

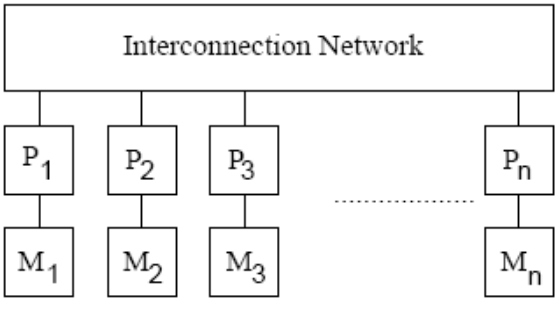
2.1 Pemodelan Algoritma Paralel

Sebelumnya telah dijelaskan bahwa hingga saat ini pemodelan algoritma paralel yang tepat belum disepakati oleh para pakar. Sejauh ini, ada dua pemodelan yang mendekati ideal, yaitu *multi-processor model* dan *work-depth model.*

2.1.1 Multi-processor Model

Pemodelan ini merupakan bentuk generalisasi dari model RAM dalam komputasi sekuensial. Pemodelan ini dapat diklasifikasikan menjadi tiga jenis, yaitu model mesin memori lokal, mesin memori modular, dan mesin PRAM (*Parallel Random Access Memory*). Setiap jenis multiproseor tersebut memiliki perbedaan baik dari stuktur maupun cara aksesnya.

Model mesin memori lokal tersusun atas n buah prosesor yang masing-masing memiliki memori lokalnya sendiri. Karena tiap prosesor memiliki memori internal, mesin ini mampu mengakses secara langsung data yang disimpan di memorinya sendiri. Untuk mengambil data dari prosesor lain, perlu dikirim sebuah sinyal permintaan melalui jaringan. Durasi yang diperlukan untuk mengakses data dari prosesor lain sangat bergantung kepada kemampuan jaringan untuk menyampaikan informasi dan pola akses memori tiap-tiap prosesor.



local memory machine model

Tidak seperti mesin memori lokal, memori dan prosesor di dalam model mesin memori modular memiliki jumlah yang berbeda. Model ini terdiri atas m buah memori dan n buah prosesor yang terhubung dalam sebuah jaringan yang sama. Akses prosesor terhadap memori dilakukan dengan mengirimkan permintaan melalui jaringan. Lagi-lagi durasi yang diperlukan bergantung pada tingkat komunikasi jaringan dan pola akses memori. Namun, kecepatannya akan cenderung seragam karena memori tersusun sedemikian rupa sehingga akses prosesor ke memori manapun membutuhkan waktu yang sama.

Bentuk lain dari pemodelan multiprosesor adalah mesin PRAM. PRAM terdiri atas sekumpulan prosesor dan semuanya terhubung ke sebuah memori yang sama. Tentu saja arsitektur semacam ini akan mempercepat waktu akses bagi semua prosesor. Hanya dengan satu instruksi, semua prosesor dapat bekerja secara paralel. Hal itulah yang membuat PRAM menjadi sebuah kontroversi. Pada kenyataannya, tidak ada mesin yang dapat memberikan kondisi ideal seperti itu.

Tujuan dari pemodelan bukanlah serta merta harus diimplementasikan secara fisik. Pemodelan digunakan untuk mempermudah para *programmer* untuk mendesain algoritma yang efisien. Jika algoritma tersebut dapat beroperasi secara efisien dalam komputer/mesin, pemodelan dianggap berhasil. Oleh karena itu, pemodelan yang abstrak seperti PRAM seharusnya tidak menjadi masalah.

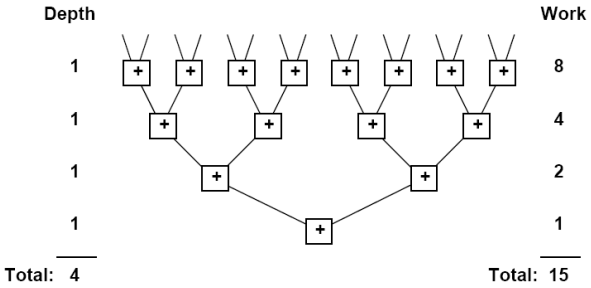
2.1.2 Work-depth Model

Pemodelan ini merupakan salahsatu alternatif yang dapat digunakan. Jika pemodelan sebelumnya lebih fokus terhadap mesin/komputer, fokus yang digunakan dalam pemodelan ini adalah bentuk algoritma. Algoritma tersebut dapat dinyatakan dengan sebuah perbandingan/rasio yang biasa disebut rasio paralelisme P = (W/D). Kerja algoritma (W) dinyatakan oleh hasil kali jumlah prosesor dengan waktu yang diperlukan algoritma untuk menyelesaikan eksekusi sedangkan kedalaman (D) dinyatakan oleh waktu total yang diperlukan algoritma untuk menyelesaikan eksekusi. Pemodelan ini juga diklasifikasikan menjadi tiga jenis, yaitu model mesin vektor, model sirkuit, dan model berbasis bahasa (language-based).

Model mesin vektor menyatakan algoritma sebagai urutan langkah-langkah. Algoritma tersebut melakukan operasi vektor terhadap nilai masukan sehingga menghasilkan vektor. Kerja algoritma (W) dihitung dari jumlah total kerja di setiap langkah sedangkan kedalaman (D) ditentukan oleh jumlah langkah dalam operasi vektor.

Model sirkuit hampir menyerupai graf berarah.

Model in tersusun atas node dan busur. Node berfungsi sebagai tempat melakukan operasi sedangkan busur menghubungkan node-node tersebut. Berdasarkan fungsinya, busur terbagi menjadi dua, yaitu busur input dan busur output. Busur input mengirim nilai masukan ke dalam sirkuit dan tidak berpangkal dari sebuah node. Sebaliknya, busur output mengirimksn hasil proses dari sirkuit dan berasal dari sebuah node. Berikut adalah representasi fisik algoritma tersebut.



Gambar tersebut adalah algoritma penjumlahan 16 angka. *Depth* merupakan kedalaman algoritma dan *work* adalah jumlah kerja algoritma. Nilai setiap data akan dijumlahkan saat mencapai node. Dari algoritma tersebut didapat bahwa W = n – 1 dan D = ^{2}log n.

Lain halnya dengan model berbasis bahasa (language-based). Algoritma dalam model ini hanya melakukan operasi pemanggilan fungsi. Oleh karena itu, jumlah kerja algoritma ditentukan oleh kerja masing-masing fungsi dan kedalamannya juga bergantung kepada fungsi-fungsi yang dipanggil.

Jika dibandingkan dengan *multi-processor model*, *work-depth model* memiliki struktur yang lebih abstrak. Dari ketiga jenis model algoritma yang telah dijelaskan, model sirkuit adalah yang paling abstrak. Tingkat abstraksi yang lebih tinggi menjadi keunggulan dari pemodelan ini karena algoritma yang dibuat tidak harus berpatokan kepada arsitektur mesin. Akibatnya, *programmer* bisa lebih leluasa dalam mendesain algoritmanya.

2.2 Teknik Algoritma Paralel

Seperti halnya dalam algoritma sekuensial, algoritma paralel memiliki teknik-teknik umum yang dapat digunakan untuk menyelesaikan berbagai persoalan. Sebagian besar teknik merupakan hasil modifikasi dari algoritma sekuensial sedangkan beberapa lainnya teknik yang khusus untuk dipakai dalam algoritma paralel. Berikut ini beberapa teknik yang digunakan dalam algoritma paralel.

2.2.1 Divide and Conquer

Prinsip dari teknik ini adalah membagi-bagi sebuah persoalan menjadi beberapa sub persoalan yang lebih sederhana. Pada akhirnya, solusi untuk setiap sub persoalan digabungkan untuk menyelesaikan persoalan utama. Teknik ini terbukti sangat efisien dalam algoritma sekuensial dan pastinya akan mempunyai peranan yang lebih penting dalam algoritma paralel. Untuk meningkatkan efisiensi, proses pemecahan dan penggabungan kembali perlu dilakukan secara paralel. Selain itu, sangat penting dalam algoritma paralel jika persoalan utama dibagi-bagi sebanyak mungkin menjadi sub persoalan.

Salahsatu contoh dari penerapan teknik ini adalah algoritma mergesort. Algoritma ini menerima serangkaian n buah masukan dan mengembalikan nilai masukan tersebut dalam keadaan terurut. Serangkaian nilai masukan dibagi menjadi dua (n/2 elemen). Kemudian, proses pengurutan dilakukan secara rekursif. Kompleksitas algoritma tersebut dapat dinyatakan dengan

$$T(n) = \begin{cases} 2T(n/2) + O(n) & n > 1 \\ O(1) & n = 1 \end{cases}$$

Sebenarnya, langkah-langkah tersebut berlaku untuk algoritma sekuensial. Perlu dilakukan modifikasi agar algoritma ini sesuai untuk algoritma paralel. Berikut langkah-langkah yang dapat diterapkan

```
1  if (|A| = 1) then return A
2  else
3     in parallel do
4        L := MERGESORT(A[0..|A|/2))
5        R := MERGESORT(A[|A|/2..|A|))
6     return MERGE(L, R)
```

2.2.2 Randomization

Teknik ini sangat diperlukan untuk memastikan prosesor mampu menentukan keputusan lokal yang tepat. Hal ini sangat penting karena akan memengaruhi performansi algoritma paralel secara keseluruhan.

Randomisasi biasa dilakukan dalam metode *sampling*. Dalam proses ini, randomisasi digunakan untuk menentukan sampel representatif dari setiap himpunan data. Sampel tersebut akan diolah dan hasilnya akan dijadikan solusi umum persoalan. Misalnya akan dilakukan pengurutan terhadap data. Data-data terxebut dipisahkan menjadi beberapa bagian dan jumlahnya harus sama. Sampel acak akan menentukan batas interval tiap-tiap bagian. Selain itu, sampel acak juga digunakan dalam komputasi geometri, graf, dan algoritma perbandingan *string*.

Metode lain yang juga memerlukan randomisasi adalah *symmetry breaking*. Metode ini biasa dipakai dalam operasi graf. Misalnya, menentukan simpul independen. Pemeriksaan harus dilakukan terhadap semua simpul. Apabila ada dua simpul yang tidak saling bertetangga, maka simpul tersebut independen. Hal tersebut cukup rumit jika dilakukan terhadap jumlah simpul yang banyak. Oleh karena itu, digunakanlah metode *symmetry breaking*.

Terakhir, randomisasi juga biasa dipakai dalam metode *load balancing*. Metode ini dipakai untuk melakukan partisi data yang berukuran sangat besar. Randomisasi dilakukan dengan memasukkan elemen secara acak ke dalam kotak-kotak partisi. Metode ini akan bekerja dengan baik jika ukuran tiap kotak partisi sama.

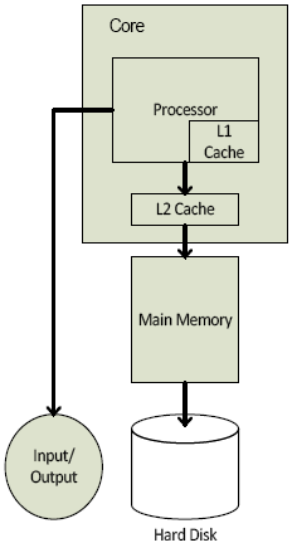
2.2.3 Parallel Pointer

Sebagian besar teknik mengelola list, pohon, dan graf dalam algoritma sekuensial tidak dapt diterjemahkan untuk penggunaan algoritma paralel. Contohnya, menelusuri setiap elemen list, mengunjungi simpul pohon secara inorder, dan menelusuri graf. Namun, aksi-aksi tersebut dapat digantikan dengan aksi lain yang setara. Aksi-aksi pengganti tersebut diantaranya adalah *pointer jumping*, *euler tour*, *graph contraction*, dan *ear decomposition*.

III. MULTIPROSESOR

3.1 Arsitektur Multicore

Deskripsi yang akan dijelaskan dalam bagian ini tidak merujuk kepada model prosesor manapun tetapi lebih menekankan terhadap desain dasar. Berikut ini adalah sketsa arsitektur dari sebuah mikroprosesor.



Di dalam prosesor, terdapat *cache* level 1 (L1) yang memiliki kecepatan paling tinggi. *Cache* ini berfungsi untuk menyimpan informasi yang sering digunakan oleh prosesor. Sama halnya seperti L1, level 2 *cache* (L2) juga memiliki fungsi yang sama. Perbedaan antara L1 dan L2 adalah kapasitas memorinya. *Main memory* memiliki kapasitas lebih besar tetapi kecepatan aksesnya lebih lambat. Bagian ini biasanya dipakai untuk menyimpan data file. *Hard Disk* dipakai untuk menyimpan data lain yang tidak dimuat dalam L1, L2 , maupun *main memory*.

Sebagian besar sistem operasi memiliki main memory dengan kapasitas yang berkisar antara 1 hingga 4 GB dengan kapasitas L1 sebesar 32 KB dan kapasitas L2 sebesar 2 MB. Sistem ini masih menggunakan prosesor *singlecore*. Jika dua *core* yang diletakkan bersebelahan, seperti itulah kira-kira gambaran dari prosesor *multicore*. Dengan susunan seperti itu, tentu dibutuhkan mekanisme komunkasi antara *core* dengan *main memory*. Biasanya, persoalan ini diselesaikan dengan penerapan jalur komunikasi tunggal atau jaringan interkoneksi. Jalur komunikasi tunggal menggunakan pendekatan *shared memory* sedangkan jaringan interkoneksi menggunakan pendekatan *distributed memory*.

3.2 Paralelisme dalam Multicore

Karena prosesor multicore bekerja secara paralel, sangat penting bagi kita untuk dapat memahami tipe-tipe dasar paralelisme. Ada 3 jenis tipe, yaitu *instruction-level paralellism* (ILP), *thread-level paralellism (TLP)*, dan *data-level paralellism* (DLP). Ketiga tipe tersebut akan memengaruhi performansi prosesor secara berbeda jika diterapkan dalam sistem.

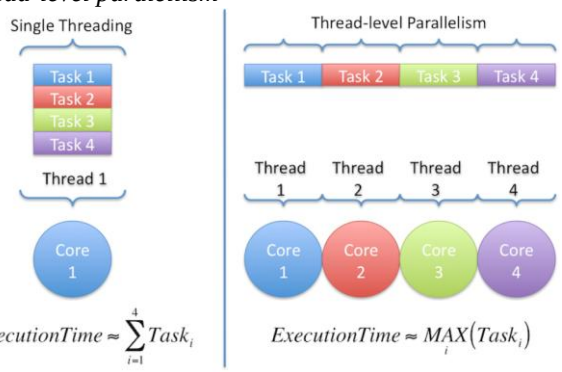
Dalam *instruction-level paralellism*, sejumlah instruksi tertentu akan dijalankan dalam waktu yang bersamaan. CPU modern menggunakan teknik ini untuk *pipelining*, prediksi, dan spekulasi alamat. Sebagai contoh sederhana, akan ditampilkan tabel berikut.

Sequential Execution	Instruction-Level Parallelism
1. a = 10 + 5 2. b = 12 + 7 3. c = a + b Instructions: 3 Cycles: 3	1.A. a = 10 + 5 1.B. b = 12 + 7 2. c = a + b Instructions: 3 Cycles: 2 (-33%)

Karena langkah 1 dan langkah 2 saling lepas satu sama lain, kedua langkah tersebut dapat dijalankan pada waktu yang bersamaan. Dapat kita lihat bahwa hal tersebut mengurangi siklus program sebesar 33%. Namun, langkah 3 harus tetap dilakukan secara sekuensial karena bergantung kepada dua langkah sebelumnya. Contoh ini tidak hanya merepresentasikan kelebihan dari *instruction-level paralellism* tapi juga keterbatasannya.

Thread yang bersifat individual akan diproses dalam waktu yang bersamaan jika sistem tersebut menerapkan *thread-level paralellism*. *Thread* adalah istilah yang digunakan untuk proses/instruksi yang sedang berjalan. Tipe paralelisme ini dipengaruhi oleh cukup banyak faktor, misalnya spesifikasi *hardware*, spesifikasi *thread-implementation*, dll.

Gambar di bawah ini adalah ilustrasi yang menggambarkan perbedaan antara *single-thread* dan *thread-level paralellism*



Dari gambar tersebut dapat dilihat bahwa prosesor multicore tidak memproses *thread* dengan menggunakan *stack*. Setiap *core* dalam prosesor akan menampung *thread* dan mengeksekusinya secara paralel. Dengan demikian, performansi sistem akan meningkat. Namun, tingkat performansi sistem sangat dipengaruhi oleh kemampuan *core*. Oleh karena itu, kita perlu mempertimbangkan batasan sebuah *core* supaya efisiensi dapat mencapai optimal.

Data-level paralellism menggunakan prinsip koherensi memori dalam setiap proses agar proseor dapat berbagi data yang sama. Hal ini akan menambah efisiensi karena mengurangi waktu untuk mengakses memori.

Performansi diharapkan meningkat karena suatu salinan data dapat dipakai oleh beberapa prosesor dalam waktu yang sama. Jika sesuai dengan harapan, waktu eksekusi akan berkurang begitu pula halnya dengan penyalinan data. Keuntungan tidak akan didapat jika setiap thread tidak mengakses data yang sama. Kerugian pun mungkin terjadi apabila permintaan data dari prosesor kepada memori melebihi batas yang seharusnya.

Operasi penulisan juga memiliki pengaruh yang sama terhadap performansi sistem. Dalam kasus penulisan di lokasi memori yang sama, penulisan tidak dapat dilakukan sekaligus karena ada data konflik yang harus ditangani. Biasanya, hal ini ditangani oleh sebuah skema yang bernama *spin-locks*. Performansi akan ditentukan oleh skema dan frekuensi terjadinya data konflik. Oleh karena itu, pilihan terbaik adalah mencari alternatif lokasi memori yamg dapat digunakan.

Selain kedua operasi tersebut, performansi dalam *data-level paralellism* juga dipengaruhi oleh ukuran *cache*, kapasitas *bandwidth*, dan beberapa faktor lainnya. Observasi terhadap jenis paralelisme ini dalam aplikasi tertentu sangatlah penting. Hal ini disebabkan performansi yang seringkali kurang optimal karena faktor keterbatasan memori.

IV. PEMBAHASAN

Perkembangan pesat yang terjadi terhadap teknologi rangkaian listrik menjadi faktor utama dalam desain prosesor. Seperti yang telah kita ketahui, prosesor tersusun atas beberapa transistor. Kemajuan teknologi mampu memperkecil ukuran transistor. Akibatnya, jumlah transistor yang dapat dipasang menjadi lebih banyak. Jumlah transistor berbanding lurus dengan kecepatan komputasi prosesor. Semakin banyak jumlah transistor, semakin cepat pula komputasinya. Hal inilah yang menjadi ide dasar lahirnya prosesor multicore.

Sebenarnya, multicore bukanlah sebuah gagasan baru. Konsep ini telah diterapkan dalam embedded system dan beberapa aplikasi dengan tujuan khusus. Namun, belakangan ini tampaknya multicore tampaknya akan dijadikan sebagai desain umum untuk prosesor. Perusahaan Intel contohnya, memperkenalkan prosesor multicore dan menjualnya kepada masyarakat umum. Para ahli berpendapat bahwa pada tahun 2017, CPU komputer akan mampu menampung hingga 512 core. Jumlah ini terhitung sangat banyak jika dibandingkan dengan tahun 2008 karena CPU komputer hanya mampu menampung 2 atau 4 core.

Secara teori, menambahkan sebuah *core* ke dalam prosesor akan menambah performansi sebesar dua kali lipat dan mengurangi disipasi panas. Artinya, tingkat kenaikan yang signifikan dapat diperoleh tanpa menambah konsumsi energi. Dapat dibayangkan bagaimana konsep ini tampaknya akan menjadi sebuah inovasi yang luar biasa. Namun, teori tersebut belum bisa direalisasikan. Hal ini disebabkan kecepatan masing-masing *core* dalam prosesor multicore masih kalah cepat dengan prosesor *singlecore*.

Selain itu, masih ada beberapa hal lainnya yang perlu diperhatikan dalam prosesor multicore. Pertama, masalah manajemen memori. Meskipun di bagian sebelumnya telah dibahas mengenai koherensi memori, tetap saja hal tersebut masih menjadi tantangan. Kedua, adanya *multi-threading*. Tentu saja masalah ini memaksa para *programmer* untuk mendesain algoritma yang mampu menangani hal tersebut.

Seiring dengan perkembangan prosesor, sangat penting bagi perangkat untuk mengikutinya sesuai dengan hukum Moore. Dalam hal ini, perangkat lunak harus mampu menangani paralelisme sebanyak dua kali lipat dari jumlah sebelumnya dalam jangka waktu 2 tahun. Gagasan ini tentu saja menimbulkan beberapa pertanyaan. Misalnya, bagaimana *programmer* memastikan *thread* dengan prioritas tinggi dapat diprioritaskan di skala prosesor, bukan hanya dalam skala core? Masalah yang tidak kalah pentingnya adalah *debugging*. Bagaimana programmer bisa menjamin sistem berhenti secara keseluruhan, tidak hanya *core* yang sedang menjalankan aplikasi?

Di sisi lain, algoritma paralel belum terlalu banyak pemakaiannya. Kebanyakan algoritma yang dipakai saat ini adalah algoritma sekuensial. Padahal, prinsip paralel sangat cocok dengan cara kerja prosesor multicore. Metode ini pun tampaknya dapat menangani masalah *multi-threading* dalam prosesor multicore. Oleh karena itu, konsep algoritma paralel perlu lebih dimatangkan. Selain itu, usaha ini harus diikuti dengan pelatihan pemrograman paralel yang baik. Apabila *programmer* sudah menguasai dasar-dasar pemrograman paralel, mengembangkan perangkat lunak sesuai dengan hukum Moore bukanlah hal yang sulit.

V. KESIMPULAN

Adanya beberapa *core* dalam satu prosesor tentu saja memunculkan beberapa masalah dan tantangan. Suplai energi dan pengaturan suhu adalah dua hal yang akan bertambah secara eksponensial seiring dengan penambahan core. Selain itu, koherensi memori/*cache*

menjadi tantangan lainnya. L1 dan L2 harus mampu berkoordinasi dengan arsitektur yang menggunakan pola akses distribusi. Terakhir, memanfaatkan potensi yang dimiliki prosesor multicore secara optimal adalah tantangan lainnya. Jika *programmer* mendesain perangkat lunak tanpa adanya pemanfaatan multicore, tidak akan ada keuntungan yang diperoleh. Malah, dalam beberapa kasus hal tersebut akan menimbulkan kerugian. Oleh karena itu, algoritma paralel dapat menjadi solusi yang efektif untuk menjawab tantangan tersebut. Algoritma ini mengarah kepada penggunaan prosesor multicore secara optimal.

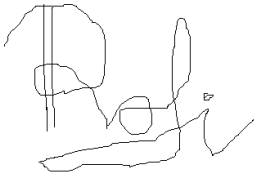
REFERENCES

[1] http://www.cs.cmu.edu/~guyb/papers/BM04.pdf
[2] http://www.mcs.anl.gov/~itf/dbpp/text/node14.html.
[3] http://wwwusers.cs.umn.edu/~karypis/parbook/Lectures/AG/chap1_slides.pdf
[4] http://www.csa.com/discoveryguides/multicore/review.pdf
[5] http://www.cs.wustl.edu/~jain/cse567-11/ftp/multcore

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 17 Desember 2013



ttd

Rafi Ramadhan - 13512075