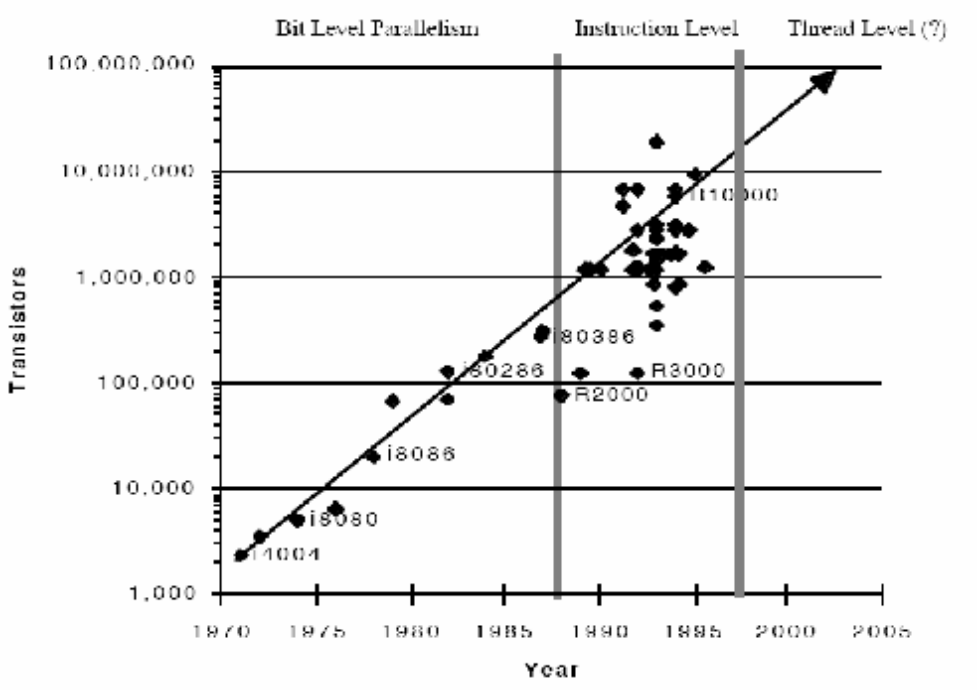
BAB II
LANDASAN TEORI

2.1 Komputasi Paralel

Teknologi komputasi paralel sudah berkembang lebih dari dua dekade, penggunaannya semakin beragam mulai dari kebutuhan perhitungan di laboratorium fisika nuklir, simulasi pesawat luar angkasa, hingga prakiraan cuaca. Komputasi paralel didefinisikan sebagai penggunaan sekumpulan sumberdaya komputer secara simultan untuk menyelesaikan permasalahan komputasi.[4] Secara prinsip komputer paralel membagi permasalahan sehingga menjadi lebih kecil untuk dikerjakan oleh setiap prosesor (CPU) dalam waktu yang bersamaan/simultan (*concurrent*). Prinsip ini disebut paralelisme.



Gambar.2.1. Tren Paralelisme Prosesor [4]

Paralelisme dalam komputasi paralel merupakan hal yang diciptakan dan dimanfaatkan. Sebenarnya prinsip paralelisme juga sudah diterapkan dalam komputer serial misal dengan *pipelining* dan *superscalar*-nya namun demikian tidak memberikan solusi terbaik dalam hal meningkatkan performansi dikarenakan terbatasnya kemampuan untuk menambah kecepatan prosesor dan

fenomena *memory bottleneck*. Perkembangan penerapan paralelisme pada prosesor dari masa ke masa ditunjukkan pada Gambar 2.1. Dari gambar tersebut kita dapatkan beberapa tingkat paralelisme dalam komputasi khususnya pada prosesor, di antaranya :

1) Paralelisme *bit-level*. Contoh : prosesor 32 bit dan prosesor 64 bit.
2) Paralelisme *instruction set-level*. Contoh : CISC dan RISC.
3) Paralelisme *thread-level*. Contoh : Intel *hyperthreading*.

Paralelisme lain yang juga berkembang dalam komputasi paralel adalah paralelisme data dan paralelisme fungsi (*task*).

Perkembangan teknologi prosesor memberikan pengaruh yang besar pada komputasi paralel. Mulai dari prosesor *singlecore superscalar*, *chip multiprocessor*, prosesor *multicore*, hingga prosesor *cell* memberikan kontribusi terhadap peningkatan performansi komputer paralel. *Supercomputer* seperti *Roadrunner* misalnya menggunakan teknologi multiprosesor, prosesor cell, atau gabungan dari keduanya (*hybrid system*)[5]. Jumlah prosesor yang dipakai HPC juga semakin tidak terbatas sehingga arsitekturnya disebut *Massively Parallel Processing* (MPP). Namun demikian penggunaan *cluster* PC menjadi tren dalam komputasi paralel karena faktor biaya dan skalabilitas. Dari Tabel 2.1. diperoleh data bahwa *cluster* menjadi pilihan terbanyak para pengembang HPC.

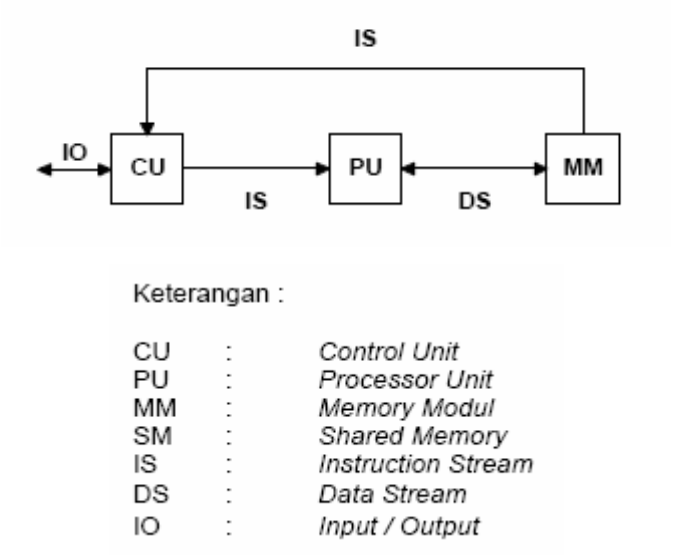
Tabel 2.1. Komposisi Arsitektur HPC Tahun 2009. [1]

Arsitektur	Jumlah	Persentase	Rmax (Gflops)	Jumlah Core
Cluster	417	83,4 %	16.916.825	2.553.904
Constellations	2	0,4 %	94.970	17.648
MPP	81	16,2 %	10.939.467	2.090.251
Total	500	100 %	27.951.300	4.661.803

2.1.1 Arsitektur Komputer Paralel

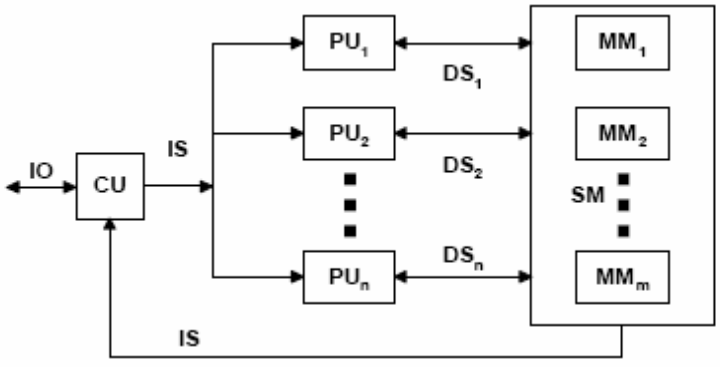
Berdasarkan jumlah dan prinsip kerja prosesor pada komputer paralel, A.J. Van der Steen dan J. Donggara menyebutkan terdapat empat arsitektur utama komputer paralel menurut Flynn (1972) yaitu [6]:

1) SISD (*Single Instruction – Single Data*). Komputer ini memiliki hanya satu prosesor dan satu instruksi yang dieksekusi secara serial. Komputer ini adalah tipe komputer konvensional. Menurut mereka tipe komputer ini tidak ada dalam praktik komputer paralel karena bahkan mainframe pun tidak lagi menggunakan satu prosesor. Klasifikasi ini sekedar untuk melengkapi definisi komputer paralel. Skema SISD ditunjukkan pada Gambar 2.2.



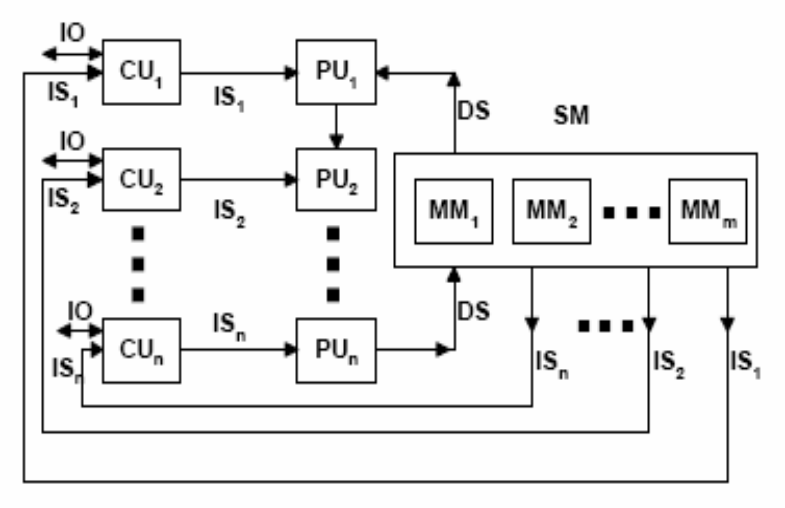
Gambar 2.2. Skema SISD [7]

2) SIMD (*Single Instruction – Multiple Data*). Komputer ini memiliki lebih dari satu prosesor, tetapi hanya mengeksekusi satu instruksi secara paralel pada data yang berbeda pada level *lock-step*. Komputer vektor adalah salah satu komputer paralel yang menggunakan arsitektur ini. Skema SIMD ditunjukkan pada Gambar 2.3.



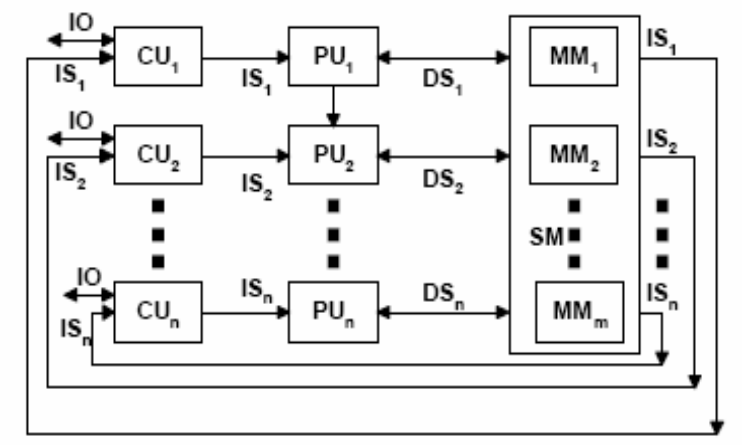
Gambar 2.3. Skema SIMD [7]

3) MISD (*Multiple Instructions – Single Data*). Teorinya komputer ini memiliki satu prosesor dan mengeksekusi beberapa instruksi secara paralel tetapi praktiknya tidak ada komputer yang dibangun dengan arsitektur ini karena sistemnya tidak mudah dipahami. Skema MISD ditunjukkan pada Gambar 2.4.



Gambar 2.4. Skema MISD [7]

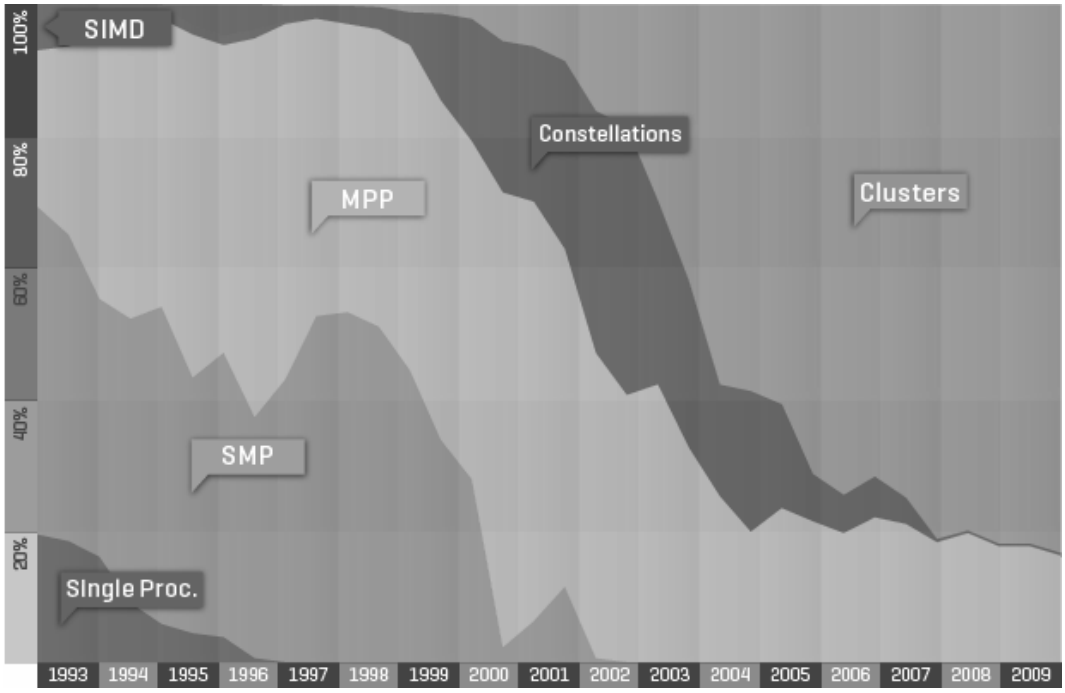
4) MIMD (*Multiple Instructions – Multiple Data*). Komputer ini memiliki lebih dari satu prosesor dan mengeksekusi lebih dari satu instruksi secara paralel. Tipe komputer ini yang paling banyak digunakan untuk membangun komputer paralel, bahkan banyak *supercomputer* yang menerapkan arsitektur ini. Skema MIMD ditunjukkan pada Gambar 2.5.



Gambar 2.5. Skema MIMD [7]

Sistem komputer paralel dibedakan dari cara kerja memorinya menjadi *shared memory* dan *distributed memory*. *Shared memory* berarti memori tunggal diakses oleh satu atau lebih prosesor untuk menjalankan instruksi sedangkan *distributed memory* berarti setiap prosesor memiliki memori sendiri untuk menjalankan instruksi. Menurut A.J. Van der Steen dan J. Donggara baik sistem *shared memory* maupun *distributed memory* merupakan SIMD atau MIMD. [6]

Top500 Team membagi arsitektur komputer paralel dalam 6 kelompok berdasarkan daftarnya sejak tahun 1993 yaitu : SIMD, *Single Processor*, SMP, MPP, *Constellation* dan *Cluster* [1]. Dari keenam arsitektur tersebut hanya 3 kelompok yang masih bertahan dalam daftar di Nopember 2009 seperti ditunjukkan dalam Gambar 2.6.

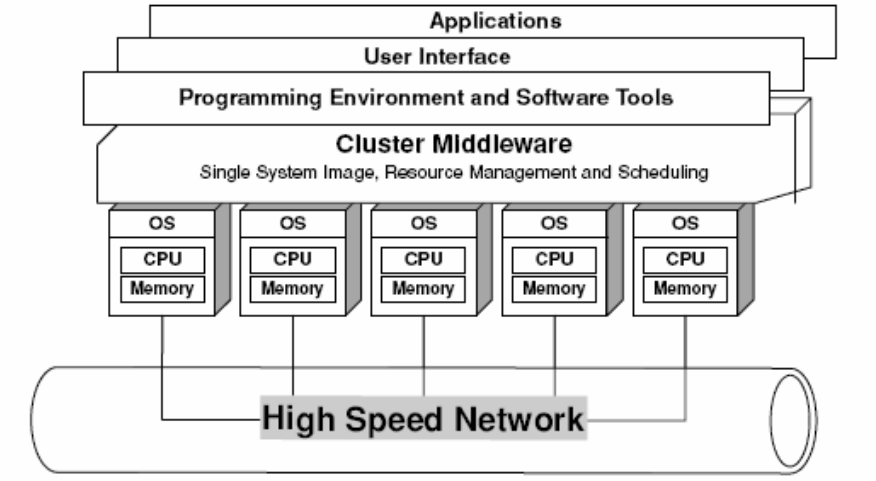


Gambar 2.6. Arsitektur HPC Tahun 1993 - 2009 [1]

Pada penelitian ini arsitektur yang digunakan adalah *cluster* PC *multicore* yang merupakan penerapan arsitektur MIMD dengan *distributed shared memory*. Skema arsitektur ini ditunjukkan pada Gambar 2.7. Adapun komponen-komponen utama dari arsitektur komputer paralel *cluster* PC antara lain [8] :

1) Prosesor (CPU). Bagian paling penting dalam sistem, untuk *multicore* terdapat lebih dari satu core yang mengakses sebuah memori (*shared memory*).

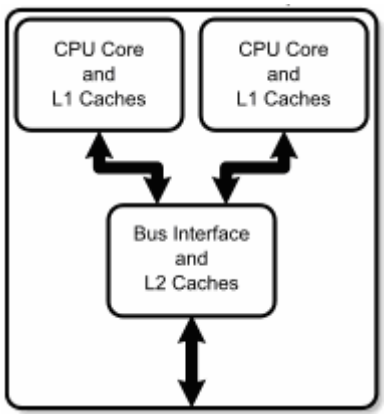
2) Memori. Bagian ini dapat diperinci lagi menjadi beberapa bagian penyusunnya seperti RAM, *cache memory* dan memori eksternal.
3) Sistem Operasi. *Software* dasar untuk menjalankan sistem komputer.
4) *Cluster Middleware*. Antarmuka antara *hardware* dan *software*.
5) *Programming Environment* dan *Software Tools*. *Software* yang digunakan untuk pemrograman paralel termasuk *software* pendukungnya.
6) *User Interface*. *Software* yang menjadi perantara *hardware* dengan *user*.
7) Aplikasi. *Software* berisi program permasalahan yang akan diselesaikan.
8) Jaringan. Penghubung satu PC (prosesor) dengan PC yang lain sehingga memungkinkan pemanfaatan sumberdaya secara simultan.



Gambar 2.7. Arsitektur *Cluster* Komputer [9]

2.1.1.1 Komputer Multicore

Pada saat ini PC secara umum telah menggunakan paralelisme *thread-level* dengan berkembangnya teknologi *multicore*. Teknologi ini menempatkan lebih dari satu *general purpose processor* pada satu keping mikroprosesor sehingga memungkinkan lebih dari satu *thread* bekerja secara simultan. Prosesor *multicore* umumnya menggunakan prinsip *shared-memory*, dimana masing-masing *core processor* terhubung dengan sebuah memori internal (*cache memory*) yang menjadi perantara baginya dengan memori eksternal seperti RAM. Hal ini ditunjukkan dengan Gambar 2.8. Namun demikian adapula prosesor multicore yang menggunakan prinsip message passing untuk berkomunikasi antar core [10].



Gambar 2.8. Skema Prosesor *Multicore* [10]

Paralelisme fisik pada prosesor *multicore* sebenarnya tidak serta merta mendorong peningkatan performansi komputer. Jumlah *core* yang banyak tidak akan berpengaruh bila tidak didukung dengan algoritma dan aplikasinya. Program paralel dengan berbagai level paralelisme dapat diterapkan untuk mengoptimalkan kinerja prosesor, mulai dari *instruction-level parallelism* (ILP) hingga *thread-level parallelism* (TLP).

Keuntungan teknologi multicore di antaranya adalah [10]:

1) Percepatan proses dengan adanya *cache coherency circuitry*.
2) Penghematan ruang yang berarti semakin kecil ukuran fisik mikroprosesor.
3) Penghematan energi karena daya yang digunakan oleh dua *singlecore* lebih besar dibandingkan dengan sebuah *dualcore*.

Selain keuntungan, prosesor multicore juga tidak lepas dari kekurangan yaitu [10]:

1) Perlu pembaruan software sehingga mendukung kinerja *core* yang tersedia.
2) Kesulitan mengelola panas yang dihasilkan oleh kerja prosesor.
3) Kesulitan meningkatkan performansi prosesor karena keterbatasan bandwidth memori dan bus sharing.

2.1.2 Algoritma Paralel

Komputasi paralel digunakan untuk menyelesaikan permasalahan komputasi yang besar atau komplek. Program paralel dibuat khusus atau dimodifikasi dari program serial. Algoritma paralel digunakan untuk menggantikan algoritma serial menyesuaikan arsitektur komputer yang digunakan. Berdasarkan klasifikasi

arsitektur komputer paralel di atas komputer paralel *shared memory* dan *distributed memory* tentu saja menggunakan algortima paralel yang berbeda.

Algoritma paralel menjelaskan langkah-langkah yang ditempuh oleh komputer paralel dalam menyelesaikan permasalahan [8]. Hal-hal yang ada dalam algoritma paralel meliputi :

1) Identifikasi terhadap beban permasalahan yang akan dikerjakan secara paralel.
2) Pemetaan porsi pekerjaan yang dibebankan kepada tiap-tiap proses.
3) Distribusi data input, output dan perantara yang terkait dengan program.
4) Pengaturan data yang diakses bersamaan oleh beberapa prosesor.
5) Menyelaraskan fungsi prosesor pada setiap langkah pekerjaan.

Ada beberapa model algortima paralel, di antaranya : *data parallel*, *task graph*, *work pool*, *master-slave* (*message passing*), dan *pipeline* [11]. Algoritma paralel yang digunakan untuk arsitektur *cluster* PC umumnya menggunakan *message passing*, di mana data dari memori sebuah PC dikirimkan ke memori PC lain melalui suatu jaringan. Beberapa konsep penting terkait dengan hal di atas adalah dekomposisi, *mapping*, dan granularitas.

2.1.2.1 Dekomposisi (pembagian beban kerja)

Pembagian beban pekerjaan adalah hal utama dalam algoritma paralel, karena tujuan utama komputasi paralel adalah mempercepat proses dengan mengerjakan permasalahan menggunakan sumberdaya yang dimiliki secara bersamaan. Berdasarkan obyek yang dibagi, dekomposisi dibedakan menjadi dekomposisi data (*domain*) dan dekomposisi fungsi [11].

Dekomposisi menyesuaikan permalasahan yang dikerjakan, semisal untuk permasalahan yang melibatkan pengulangan (iterasi) input data yang besar di mana fungsi yang digunakan sulit untuk diparalelkan maka dekomposisi data lebih baik digunakan. Untuk permasalahan lain dengan fungsi yang beragam bisa menggunakan dekomposisi fungsi. Pemilihan dekomposisi ini sangat berpengaruh pada performansi komputer paralel.

2.1.2.2 *Mapping* (pemetaan beban kerja)

Mapping terkait erat dengan arsitektur dan kapasitas komputer paralel yang digunakan. *Mapping* juga terkait erat dengan dekomposisi, di mana beban pekerjaan yang dibagi-bagi akan diberikan pada proses-proses secara paralel.

Pada arsitektur homogen di mana kemampuan dan kapasitas setiap *node* sama maka mapping menjadi lebih mudah, setiap *node* mendapatkan porsi yang sama untuk dekomposisi data. Untuk komputer paralel heterogen, *mapping* akan dilakukan dengan terlebih dahulu mengukur kapasitas dari setiap *node*, sehingga *mapping* beban kerja memberikan porsi yang berbeda-beda kepada setiap *node*.

2.1.2.3 Granularitas

Faktor penting lain dalam algoritma paralel adalah perbandingan komputasi dan komunikasi. Komputasi di sini diartikan proses yang dilakukan pada setiap prosesor sedangkan komunikasi adalah proses pertukaran informasi yang dilakukan antar prosesor. Permasalahan dengan fungsi sederhana biasanya memberikan porsi komputasi lebih besar dari pada komunikasi (*coarse grain*), sebaliknya permasalahan dengan banyak fungsi menyebabkan porsi komunikasi hampir sama dengan porsi komputasinya (*fine grain*).

Algoritma paralel mengatur granularitas sehingga tidak terjadi defisiensi proses baik komputasi maupun komunikasi. Porsi komputasi dan komunikasi juga disesuaikan dengan arsitektur komputer paralel.

2.1.3 Algortima *Multithreading*

Definisi komputasi paralel memberikan pemahaman bahwa komponen-komponen dalam komputer menjadi bagian yang dieksploitasi untuk meningkatkan performansi terutama kecepatannya. Selain pengembangan *hardware* seperti prosesor dan memori, yang juga dikembangkan adalah eksploitasi terhadap pemrograman aplikasinya (*software*). Eksploitasi *software* merupakan solusi yang mungkin dilakukan dengan biaya ringan, salah satunya penggunaan teknik pemrograman paralel dengan *multithreading*.

Komputasi paralel sangat dipengaruhi oleh perkembangan teknologi komputer, terutama prosesor. Komputer *singlecore* telah menggunakan teknik pemrograman paralel untuk meningkatkan kinerjanya, sebagai contoh penggunaan *hyperthreading* pada prosesor Intel. Sedangkan untuk komputer *multicore* ada beberapa teknik yang diterapkan, salah satunya adalah *multithreading*. Bila dalam sistem operasi kita mengenal *task*, *process* dan *thread*, maka *multithreading* adalah paralelisme *thread-level* sebagaimana penerapan *multitasking* dan

multiprocessing. Sebuah *thread* adalah sebuah aliran kendali tunggal di dalam suatu program [8] sehingga teknik *multithreading* diartikan sebagai teknik yang memanfaatkan lebih dari satu aliran kendali di dalam suatu program.

Teknik *multithreading* berdasarkan survey [12] bila dipadukan dengan teknologi *multicore* terbukti meningkatkan performansi CPU. Dalam beberapa penelitian termasuk [13], penggunaan *thread-level programming* menjadi solusi bagi peningkatan performansi komputer paralel, sehingga keunggulan *message passing* dipadukan di antara dua pustaka paralel seperti MPI dan OpenMP.

2.1.3.1 *Static Threading*

Teknik ini biasa digunakan untuk komputer dengan *chip multiprocessors* dan jenis komputer *shared-memory* lainnya. Teknik ini memungkinkan thread berbagi memori yang tersedia, menggunakan program counter dan mengeksekusi program secara independen. Sistem operasi menempatkan satu *thread* pada prosesor dan menukarnya dengan *thread* lain yang hendak menggunakan prosesor itu [14].

Mekanisme ini terhitung lambat, karenanya disebut dengan *static*. Selain itu teknik ini tidak mudah diterapkan dan rentan kesalahan. Alasannya, pembagian pekerjaan yang dinamis di antara thread-thread menyebabkan *load balancing*-nya cukup rumit. Untuk memudahkannya programmer harus menggunakan protokol komunikasi yang kompleks untuk menerapkan *scheduler load balancing*. Kondisi ini mendorong pemunculan *concurrency platforms* yang menyediakan *layer* untuk mengkoordinasi, menjadwalkan, dan mengelola sumberdaya komputasi paralel. Sebagian platform dibangun sebagai *runtime libraries* atau sebuah bahasa pemrograman paralel lengkap dengan *compiler* dan pendukung *runtime*-nya [14].

2.1.3.2 *Dynamic Multithreading*

Teknik ini merupakan pengembangan dari teknik sebelumnya yang bertujuan untuk kemudahan karena dengannya programmer tidak harus pusing dengan protokol komunikasi, load balancing, dan kerumitan lain yang ada pada static threading [14]. *Concurrency platform* ini menyediakan *scheduler* yang melakukan *load balacing* secara otomatis. Walaupun platformnya masih dalam pengembangan namun secara umum mendukung dua fitur : *nested parallelism* dan *parallel loops*.

Nested parallelism memungkinkan sebuah *subroutine* di-*spawned* (ditelurkan dalam jumlah banyak seperti telur katak) sehingga program utama tetap berjalan sementara *subroutine* menghitung hasilnya. Sedangkan parallel loops seperti halnya fungsi for namun memungkinkan iterasi loop dilakukan secara bersamaan. Salah satu contoh paltform ini adalah Cilk++.

2.2 Aplikasi Pengujian

Pada subbab ini akan dibahas tentang aplikasi yang digunakan dalam pengujian algortima paralel yaitu perkalian matrik dan pengurutan data.

2.2.1 Perkalian Matriks

Perkalian matriks adalah sebuah operasi dasar pada berbagai aplikasi aljabar linier [12]. Perkalian matrik digunakan di banyak penelitian yang berhubungan dengan pengujian kinerja komputasi paralel. Ukuran-ukuran matriks yang digunakan merupakan *range* sampel pengujian algoritma paralel.

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \bullet \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$
$$= \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix} \quad (2.1)$$

Untuk mengalikan dua matriks n x n digunakan algoritma rekursif. Misal matriks C adalah hasil perkalian matriks A dan matriks B seperti ditunjukkan oleh Persamaan 2.1 maka operasi yang dilakukan adalah 8 perkalian dan 4 penjumlahan dari submatriks (n/2) x (n/2).[14]

2.2.2 Pengurutan Data (*Sorting*)

Pengurutan data (*sorting*) adalah operasi sederhana yang melibatkan iterasi yang besarnya tergantung pada jumlah dan komposisi datanya. Operasi ini juga digunakan dalam berbagai penelitian sebagai aplikasi pengujian komputasi paralel. Panjang rangkaian data acak yang digunakan merupakan *range* sampel pengujian algoritma paralel. Ilustrasi *sorting* abjad ditunjukkan pada Gambar 2.9.

P	K	G	B	H	N	J	D	C	O	F	L	M	A	E	I
K	P	B	G	N	H	D	J	C	O	F	L	A	M	E	I
K	B	P	G	N	D	H	C	J	F	O	L	M	A	E	I

.... dan seterusnya sampai ...

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P

Gambar 2.9. Ilustrasi *Sorting*

Algortima sorting yang dikenal ada beberapa di antaranya : *insertion sort*, *merge sort*, dan *quick sort*. Adapun secara umum algoritma sorting menggunakan prinsip *divide and conquer*.

2.2.3 Algoritma Serial Aplikasi Pengujian

Kedua aplikasi tersebut biasa dikerjakan secara serial (sekuensial) dengan algortima seperti terlihat pada Gambar 2.10 dan Gambar 2.11. Dari algoritma tersebut diperoleh jumlah operasi untuk aplikasi perkalian matriks yang harus diproses adalah berjumlah n x n x n perkalian dan n penjumlahan atau total $n^3 + n^2$ operasi.

```
Input   : A[n][n], B[n][n]
Ouput  : C[n][n]
for i = 1 to n do
        for j = 1 to n do
                (1) Cij = 0
                (2) for k = 1 to n
                        Cij = Cij + (Aik x Bkj)
                    end for
        end for
end for.
```

Gambar 2.10. Algoritma Serial Perkalian Matriks

Sedangkan untuk aplikasi *sorting* jumlah operasi yang harus diproses adalah berjumlah n log n (dengan asumsi menggunakan algoritma *quick sort*).

```
Input   : S[n]
Ouput  : S'[n]
for i = 1 to n-1 do
        S'[n] = 0
        for j = i+1 to n do
                If S[j] > S[i] then
                        Swap (S[j],S[i])
        end for
        if S[j] < S[i] then
        S'[i] = S[j]
end for.
```

Gambar 2.11. Algoritma Serial Pengurutan Data