# BAB III
# PERANCANGAN SISTEM

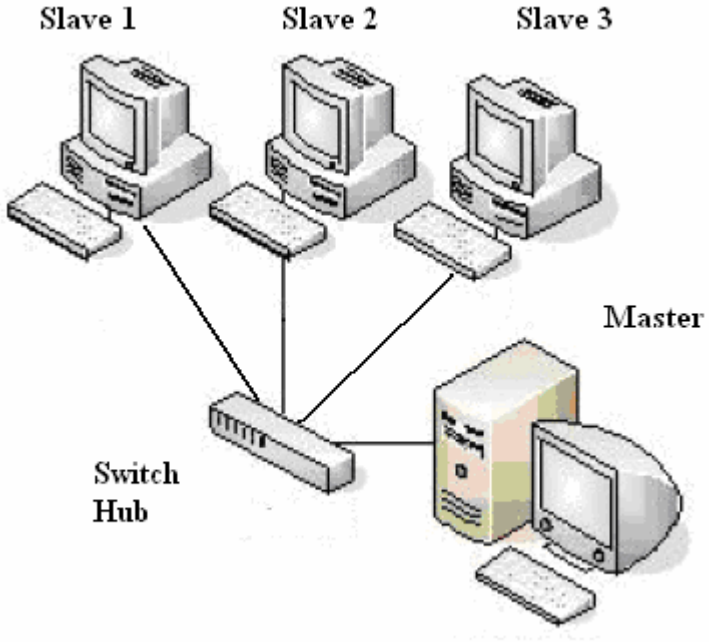
## 3.1 Konfigurasi *Cluster* PC *Multicore*

Penelitian ini bertujuan untuk mengetahui pengaruh algortima paralel pada kinerja komputasi paralel. Untuk itu konfigurasi *hardware* disusun dengan lebih dari satu PC agar dapat diukur kinerja komputasi secara paralel antar PC, selain itu masing-masing PC menggunakan prosesor *multicore* agar dapat diukur kinerja komputasi antar *core* dalam PC. Setiap PC terhubung satu sama lain melalui sebuah *switch hub* sehingga membentuk sebuah jaringan kecil.

Untuk dapat menjalankan program aplikasi paralel maka *cluster* PC tersebut dilengkapi dengan *software* mulai dari sistem operasi, editor, pustaka paralel dan bahasa pemrograman penunjang. *Cluster* yang sudah lengkap dengan software bisa berkomunikasi satu sama lain, mengirim pesan dan berbagi pakai (*shared*) memori dan prosesor.

### 3.1.1 Konfigurasi *Hardware*

Penelitian ini menggunakan komputer paralel jenis *cluster* yang secara fisik tampak pada Gambar 3.1.




Gambar 3.1. *Cluster* 4 PC *Multicore* [15]

Adapun spesifikasi dari keempat PC *multicore* dalam *cluster* tersebut adalah sebagai berikut :

1) Prosesor : Intel Core2 Quad Processor, 2,4 GHz
2) Memori standar : 2 GB (2 x 1 GB) DDR-2 SDRAM
3) Memori maksimal : 4 GB (2 DIMMs)
4) Harddisk : 320 GB Serial ATA-II/300, 7200 RPM
5) *Switch hub* : 1 GB Ethernet 8 port

### 3.1.2 Konfigurasi *Software*

*Software* yang digunakan untuk memfungsikan cluster sebagai komputer paralel meliputi :

1) Sistem operasi : Linux Slackware 12.2 [16]
2) Pustaka paralel : MPICH 2.1.2
3) Multithreading : Cilk++
4) Bahasa pemrograman : C dan C++

Untuk dapat menjalankan program aplikasi paralel, *cluster* dilengkapi dengan *software* yang dikonfigurasi dengan beberapa program sistem dengan penjabaran sebagai berikut :

Sistem Operasi

Sistem operasi adalah dasar bagi berfungsinya komputer baik *hardware* maupun *software*. Sistem operasi yang digunakan pada konfigurasi *cluster* ini adalah Linux Slackware 12.2. Penulis tidak memilih sistem operasi karena sudah terinstal dalam setiap PC yang akan digunakan. Namun demikian penggunaan Linux pada komputer paralel mempunyai maksud di antaranya : *free software* yang bisa didapatkan secara gratis tanpa membayar, *open source* yang berarti semua *listing program* (*source code*) dapat diubah sesuai kebutuhan, dan *cross platform* yang memungkinkan kompatibilitas yang luas untuk hampir semua *hardware*.

Sistem Interkoneksi

*Cluster* yang sudah terhubung satu sama lain melalui sebuah *switch hub* harus diatur sehingga dapat terkoneksi dengan baik. Interkoneksi pada konfigurasi *cluster* ini menggunakan protokol TCP/IP. Dilengkapi dengan mengatur IP

*address* secara manual pada setiap PC terhubung. Sistem interkoneksi merupakan prasyarat terhubungnya semua PC baik untuk kebutuhan komunikasi maupun untuk pertukaran data.

Pelayanan Informasi Jaringan (Network Information Service/NIS)

NIS berfungsi untuk memberikan informasi mengenai *host* (PC) yang terhubung dan membuat *map* konfigurasi.

Sistem File Jaringan (Network File System/NFS)

NFS berfungsi untuk membagi pakai (*sharing*) direktori yang akan digunakan secara bersama oleh semua PC. Hal ini dikarenakan untuk menjalankan MPICH semua PC menggunakan *source program* yang sama sehingga memerlukan mekanisme *file sharing* ini. Tanpa NFS, konfigurasi manual harus dilakukan terhadap direktori yang harus di-*sharing* sehingga MPICH bisa dijalankan.

Sistem Komunikasi

Sistem komunikasi yang dibutuhkan oleh MPICH adalah SSH/RSH tanpa password, di mana *master node* bisa mengakses semua *slave node* tanpa harus menggunakan *password* koneksi dan sebaliknya. Hal ini dimaksudkan untuk memungkinkan komunikasi terbatas yang terbuka hanya untuk PC yang sudah dikonfigurasi. SSH/RSH menggunakan *public-private key* untuk menjamin keamanan komunikasinya.

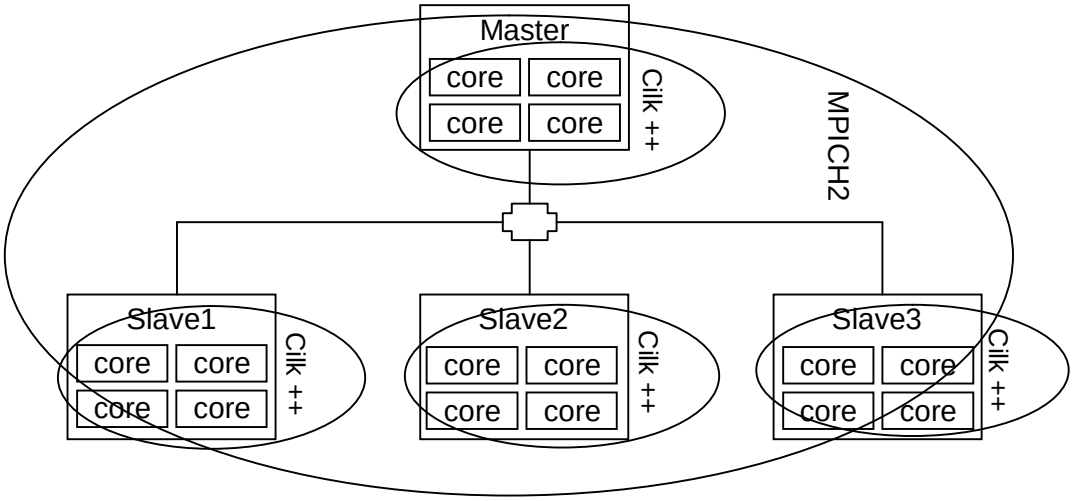
MPICH2

MPICH2 adalah software/pustaka paralel pengembangan *Message Passing Interface* (MPI). MPICH2 terdiri dari beberapa komponen utama yaitu : PMI (Process Manager Interface), ADIO dan ADI-3.

Bahasa Pemrograman

Bahasa pemrograman dibutuhkan untuk membuat aplikasi pengujian komputasi paralel. Untuk MPICH2, digunakan bahasa C, C++, atau Fortran sedangkan untuk *multithreading* dibutuhkan bahasa Cilk++, C++ dan atau C.

Adapun konfigurasi *software* yang digunakan untuk menjalankan algoritma paralel adalah seperti tampak pada Gambar 3.2.




Gambar 3.2. Skema *software* pada *cluster* PC *multicore*

## 3.2 Program Aplikasi Paralel

Pengujian komputasi paralel menggunakan aplikasi-aplikasi perkalian matrik dan pengurutan data. Adapun algoritma serialnya telah dibahas pada Bab II, maka di sini akan dibahas algoritma paralel masing-masing aplikasi hingga implementasinya dalam program.

### 3.2.1 Algoritma Paralel dengan MPICH2

Fungsi MPI yang digunakan untuk implementasi aplikasi ini meliputi :

- MPI_Init(), untuk menginisialisasi program MPI
- MPI_Comm_size(), untuk mendapatkan jumlah proses
- MPI_Comm_rank(), untuk mendapatkan *rank* tiap proses
- MPI_Bcast(), untuk mendistribusikan data ke tiap proses
- MPI_Reduce(), untuk mengumpulkan data dari semua proses
- MPI_Scatter(), untuk mendistribusikan data ke tiap proses
- MPI_Send(), untuk mengirimkan data
- MPI_Recv(), untuk menerima data
- MPI_Barrier(), untuk menunggu semua proses mendapatkan hasil
- MPI_Wtime(), untuk mencatat waktu eksekusi
- MPI_Finalize(), untuk mengakhiri program MPI

Adapun fungsi dari pustaka C yang digunakan meliputi :

- malloc(), untuk mengalokasikan memori secara dinamis

- free(), untuk membebaskan memori yang telah dialokasikan
- qsort(), untuk mengurutkan data

Aplikasi pengujian komputasi paralel yang pertama adalah perkalian matriks. Algoritma paralel untuk aplikasi tersebut terlihat pada Gambar 3.3.

```
Main:
local variables
initMatrix(A,B,C)
if MPI_Init
        if (master)     input matrixSize
                        createMatrix(A,B,C)
        MPI_Bcast(matrixSize)
        MPI_Bcast(matrix B)
        for (i=1 to procSize)
                MPI_Send(matrixRow A)
        doMatrixMultiply(A,B,C)
        for (i=1 to procSize)
                MPI_Recv(matrixRow C)
else
        MPI_Bcast(matrixSize)
        createMatrix(A,B)
        MPI_Bcast(matrix B)
        MPI_Recv(matrix A)
        matrixMultiply(A,B,C)
        MPI_Send(matrix C)
freeMatrix(&A,&B,&C)
MPI_Barrier
MPI_Finalize
```

Gambar 3.3. Algoritma Paralel Perkalian Matriks dengan MPICH2

Sedangkan algoritma paralel untuk aplikasi *sorting* tampak pada Gambar 3.4.

```
Main:
local variables
if MPI_Init
        if (master)
                input(elementSize)
                rand(data)
        parallelBinarySort
MPI_Barrier
MPI_Finalize
```

Gambar 3.4. Algoritma Paralel *Sorting* dengan MPICH2

### 3.2.2 Algoritma Paralel dengan Cilk++

Untuk implementasi Cilk++ pada aplikasi perkalian matriks ditunjukkan pada Gambar 3.5.

```
Mult(C,A,B,n)
    if n =1then C[1,1] ← A[1,1] · B[1,1]else allocate a temporary matrix T[1
      ..n,1 ..n]
            partition A,B,C and T into (n/2) × (n/2) submatricesspawn
            Mult(C11,A11,B11,n/2)spawn Mult(C12,A11,B12,n/2)spawn
            Mult(C21,A21,B11,n/2)spawn Mult(C22,A21,B12,n/2)spawn
            Mult(T11,A12,B21,n/2)spawn Mult(T12,A12,B22,n/2)spawn
            Mult(T21,A22,B21,n/2)spawn
            Mult(T22,A22,B22,n/2)syncspawn Add(C,T,n)sync

Add(C,T,n)
    if n =1then C[1,1] ← C[1,1] + T[1,1]else partition C and T into (n/2) ×
      (n/2) submatrices
            spawn Add(C11,T11,n/2)spawn Add(C12,T12,n/2)spawn
            Add(C21,T21,n/2)spawn Add(C22,T22,n/2)sync
```

Gambar 3.5. Algoritma Paralel dengan Cilk++ untuk Perkalian Matriks [14]

Sedangkan untuk aplikasi *sorting*, algoritma paralel dengan Cilk++ ditunjukkan pada Gambar 3.6.

```
P-Merge(A[1..l], B[1..m], C[1..n])
if m> l
      then spawn P-Merge(B[1 ..m],A[1 .. l],C[1 .. n])
elseif n =1
      then C[1] ←A[1]
elseif l =1
      then if A[1] ≤B[1]
            then C[1] ←A[1]; C[2] ←B[1]
            else C[1] ←B[1]; C[2] ←A[1]
      else find j such that B[j] ≤A[l/2] ≤B[j + 1] //using binary search
            spawn P-Merge(A[1..(l/2)],B[1..j],C[1..(l/2+ j)])
            spawn P-Merge(A[(l/2+1)..l],B[(j +1)..m],C[(l/2+ j +1)..n])
      sync
```

Gambar 3.6. Algoritma Paralel dengan Multithreading Pengurutan Data