BAB II
LANDASAN TEORI

2.1 Algoritma Rivest Shamir Aldeman (RSA)

Algoritma Rivest Shamir Aldeman (RSA) adalah algoritma untuk enkripsi kunci publik (*public-key encryption*).Algoritma ini adalah algoritma pertama yang diketahui paling cocok untuk menandai (*signing*) dan untuk enkripsi (*encryption*) dan salah satu penemuan besar pertama dalam kriptografi kunci publik. RSA masih digunakan secara luas dalam protokol-protokol kemanan SSL/TLS, SET, SSH, S/MIME, PGP,DNSSEC (Frederico 2003).

Algortima RSA dijabarkan pada tahun 1977 oleh Ron Rivest, Adi Shamir dan Len Adleman dari *Massachusetts Institute of Technology* (MIT). Huruf RSA itu sendiri juga berasal dari inisial nama mereka (Rivest—Shamir— Adleman). Clifford Cocks, seorang matematikawan Inggris yang bekerja untuk GCHQ, menjabarkan tentang sistem equivalen pada dokumen internal di tahun 1973. Penemuan Clifford Cocks tidak terungkap hingga tahun 1997 dikarenan alasan top-secret classification. Algoritma tersebut dipatenkan oleh Massachusetts Institute of Technology pada tahun 1983 di Amerika Serikat sebagai U. S. Patent 4405829. Paten tersebut berlaku hingga 21September 2000. Semenjak Algoritma RSA dipublikasikan sebagai aplikasi paten, regulasi disebagian besar negara-negara lain tidak memungkinkan penggunaan paten. Hal ini menyebabkan hasil temuan Clifford Cocks dikenal secara umum, paten di Amerika Serikat tidak dapat mematenkannya.

2.1.1 Pembuatan Kunci Publik dan Kunci Privat

Algoritma kriptografi RSA didesain sesuai fungsinya sehingga menghasilkan kunci yang digunakan untuk enkripsi berbeda dari kunci yang digunakan untuk dekripsi pesan. Algoritma RSA disebut menggunakan kunci publik karena kunci enkripsi yang dibuat boleh diketahui semua orang, dan juga

bisa melakukan enkripsi pesan tersebut. Sedangkan yang dimaksud kunci privat adalah kunci untuk melakukan dekripsi pesan tidak semua orang boleh mengetahuinya, hanya orang tertentu yang berhak saja untuk melakukan dekripsi pesan. Keamanan algoritma RSA didasarkan pada sulitnya memfaktorkan bilangan besar menjadi factor-faktor primanya (Lubis et al. 2013). Adapun langkah – langkah pembuatan kunci antara lain :

1. Pilih dua buah bilangan prima misal p dan q adapun p ≠ q dan secara acak dan terpisah untuk tiap-tiap p dan q. Hitung nilai N dimana nilai N adalah hasil perkalian antara bilangan p dan q, $N = p \,.\, q$
2. Hitunglah nilai $\varphi = (p - 1).(q - 1)$
3. Pilihlah bilangan bulat (*integer)* antara satu dan φ, $(1 < e < \varphi)$ yang juga merupakan bilangan koprima dari φ.
4. Hitunglah nilai d hingga d e ≡ 1 (mod φ), adapun untuk mencari d bisa menggunakan rumus

$$\boldsymbol{d = \frac{1 + k\,N}{e}} \qquad (2.1)$$

Adapun nilai k adalah hasil percobaan nilai dari 1,2,3, … sehingga menghasilkan nilai d merupakan nilai bulat

Adapun kunci publik antara lain :

1. Nilai bilangan N
2. Serta bilangan e (digunakan untuk proses enkripsi pesan)

Adapun kunci privat antara lain :

1. Nilai bilangan N
2. Serta bilangan d (digunakan untuk proses dekripsi pesan)

2.1.2 Proses Enkripsi dan Dekripsi Pesan

Dalam melakukan proses enkripsi maupun dekripsi pesan algoritma RSA memiliki rumus yang berbeda dalam merubah pesan yang diterimanya. Adapun rumus yang digunakan untuk melakukan enkripsi pesan adalah :

$$C = Plaintext^{e}\ Mod\ N \quad (2.2)$$

Keterangan :

Plaintext : Merupakan text asli yang akan dirubah

e : Merupakan kunci publik yang digunakan untuk enkripsi

N : Merupakan perkalian dua buah bilangan pirma p dan q

Sedangkan untuk merubah *chipertext* kedalam bentuk semula memerlukan rumus yang berbeda dengan proses enkripsi, rumus dekripsi pesan adalah sebagai berikut:

$$Plaintext = Chipertext^{d}\ Mod\ N \quad (2.3)$$

Keterangan:

Chipertext : Merupakan pesan yang akan dirubah kebentuk asli

d : Merupakan kunci privat yang digunakan untuk dekripsi

N : Merupakan perkalian dua buah bilangan pirma p dan q

2.1.3 Contoh Proses Algoritma RSA

Diketeahui telah dipilih dua buah bilangan prima yaitu p = 19 dan q = 41. Kemudian cari nilai N dengan mengkalikan kedua bilangan tersebut sehingga N = p.q = 19 . 41 = 779. Setelah itu carilah nilai φ dengan menggunakan rumus $\varphi = (p-1).(q-1)$ sehingga menghasilkan $\varphi = (19-1).(41-1)$ sehingga menghasilkan nilai φ = 720. Selanjutnya yang perlu dilakukan adalah mencari bilangan e dan d. Untuk mencari bilangan e dimana nilai e merupakan koprima dari nilai φ salah satu cara untuk mencai bilangan e yaitu menggunakan cara GCD. adapun hasil pencarian nilai e dapat dilihat pada tabel 2.1

Tabel 2.1 Contoh Pencarian Nilai e

Mulai dari	GCD (720,e)
e = 2	720 mod 2 = 0 GCD (2,720) = 2
e = 3	720 mod 3 = 0 GCD (3,720) = 3
e = 4	720 mod 4 = 0 GCD (4,720) = 4
e = 5	720 mod 5 = 0 GCD (5,720) = 5
e = 6	720 mod 6 = 0 GCD (6,720) = 6
e = 7	720 mod 7 = 6 7 mod 6 = 1 6 mod 1 = 0 GCD(7,720) = 1

Untuk mencari nilai e adalah nilai yang menhasilkan GCD bernilai 1, untuk nilai e itu sendiri bisa dicoba dari nilai 2,3,4,… dst. Untuk mencari nilai d bisa menggunakan rumus $= \frac{1+k\,N}{e}$. Adapun untuk contoh pencarian nilai d dapat dilihat pada tabel 2.2

Tabel 2.2 Contoh Pencarian Nilai d

Nilai K	Persamaan = $d = \frac{1+k\,N}{e}$	Hasil
K = 1	$d = \frac{1 + 1\,.\,720}{7}$	103
K = 2	$d = \frac{1 + 2\,.\,720}{7}$	205.857142857142857
K = 3	$d = \frac{1 + 3\,.\,720}{7}$	308.71428571428571

…	…	…

Untuk nilai k pada pencarian nilai d bisa dilakukan percobaan dari 1,2,3, … dst hingga hasil dari persamaan tersebut menghasilkan bilangan bulat. Pada contoh ini nilai d didapat pada k = 1 dengan hasil 103. Kemudian misalkan pesan asli berupa "Tekno" akan dirubah kedalam bentuk ASCII menjadi 87-101-107-110 yang kemudian akan dibagi menjadi P1 – P4 dan tiap Pi akan di rubah menjadi *chipertext* menggunakan rumus sehingga dapat dilihat pada tabel 2.3

Tabel 2.3 Contoh Proses Enkripsi

Pi	Enkripsi $C = Plaintext^7 \ Mod \ 779$
87	46
101	552
107	31
110	507

Sehingga pesan tersebut berubah menjadi 46-552-31-507. Untuk melakukan dekripsi pesan *chipertext* yang didapatkan dipisah menjadi beberapa bagian dalam contoh ini dibagi menjadi 4 bagian C1 – C4 untuk proses dan hasil dekripsi pesan dapat dilihat pada tabel 2.4

Tabel 2.4 Contoh Proses Dekripsi

Ci	Dekripsi Plaintext $=$ $Chipertext^{103} \ Mod \ 779$
46	87
552	101
31	107
507	110

Pada contoh tersebut didapatkan hasil 87-101-107-110 kemudian nilai tersebut dirubah menjadi nilai ASCII sehingga menghasilkan teks berupa "Tekno".

2.1.4 Kekuatan Kunci RSA

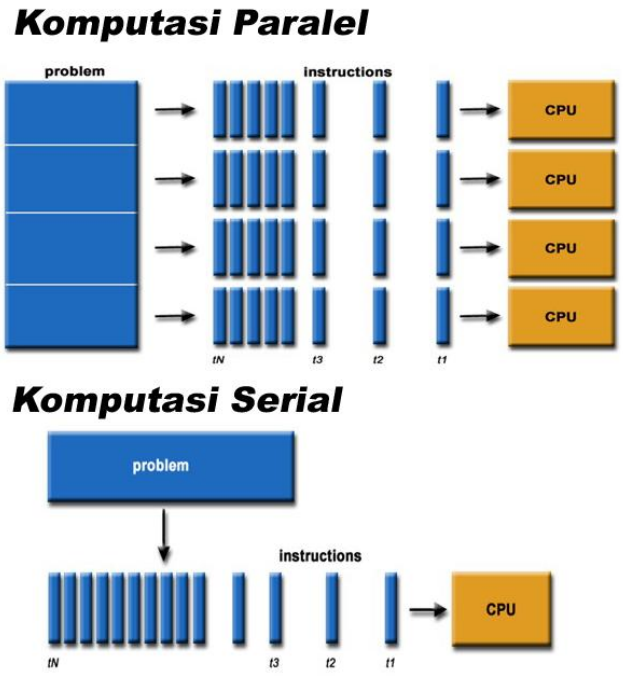
Keamanan algoritma RSA terletak pada kekuatan kunci yang dimiliki nya, hal ini disebabkan karena sulitnya mefaktorkan bilangan non prima menjadi bilangan prima yaitu $r = p \times q$. Ketika r berhasil difaktorkan menjadi p dan q, maka $\phi(r) = (p - 1)(q - 1)$ dapat dihitung. Selanjutnya, karena kunci enkrispi *PK* diumumkan (tidak rahasia), maka kunci dekripsi *SK* dapat dihitung dari persamaan $PK \cdot SK \equiv 1 \pmod{\phi(r)}$. Penemu algoritma *RSA* menyarankan nilai p dan q panjangnya lebih dari 100 digit. Dengan demikian hasil kali $r = p \times q$ akan berukuran lebih dari 200 digit. Menurut Rivest dan kawan-kawan, uasaha untuk mencari faktor bilangan 200 digit membutuhkan waktu komputasi selama 4 milyar tahun (dengan asumsi bahwa algoritma pemfaktoran yang digunakan adalah algoritma yang tercepat saat ini dan komputer yang dipakai mempunyai kecepatan 1 milidetik).Namum demikian berdasarkan paper yang berjudul Fault Based Attack of RSA Authentication yang di tulis oleh dan Andrea Pellegrini dengan rekan nya mereka mengklaim bahawa kunci private 1024bit RSA telah mereka pecahkan dengan kurun waktu 104 jam. Hal ini telah membuat kunci privat 1024 bit tidak dirasa aman lagi. Namun demikian pihak RSA tetap menyatakan bahwa mereka tetap merekomendasikan 1024 bit untuk digunakan di berbagai perusahan dan juga menyarankan penggunaan kunci 2048 bit untuk penggunaan kunci khusus tambahan (RSA Laboratories n.d.).

2.2 Komputasi Paralel

Istilah dimana bersatu kita teguh dan bercerai kita runtuh atau istilah *divide and conquer* merupakan gambaran umum dari pengertian komputasi paralel. Komputasi paralel adalah sebuah bentuk komputasi dimana melakukan banyak perhitungan pada saat yang bersamaan. Prinsip dari komputasi paralel adalah membagi suatu masalah yang kompleks atau besar kedalam bentuk yang lebih

kecil yang kemudian diselesaikan secara bersama-sama Komputasi paralel merupakan sebuah teknologi yang dapat mendayagunakan seluruh kemampuan *multiprocessor* dalam suatu aplikasi. Walaupun sudah cukup banyak aplikasi yang dapat mengeksploitasi kemampuan *multiprocessor* tetapi komputer paralel ini belum diadaptasi secara luas utuk aplikasi-aplikasi umum.

Selain komputasi paralel juga terdapat istilah komputasi sekuensial (serial) menurut Blaise Barney komputasi serial adalah pengeksekusian permasalahan pada sebuah alur proses dengan sebuah instruksi pada satu waktu yang dijalankan pada sebuah prosesor atau komputer. Pada komputasi sekuensial (serial) masukan diterima dan keluaran dikirim melalui memori internal atau perangkat antarmuka. Sementara pada komputasi paralel sebuah prosesor dapat menerima masukan atau mengembalikan keluarannya melalui 2 shared memory atau prosesor lain melalui bus interkoneksi antar prosesor atau jaringan komputer (Barney & Lawrence Livermore National Laboratory 2016).



Gambar 2.1 Perbedaan Komputasi Paralel dan Serial

Berdasarkan gambar 2.1 menunjukan gambaran perbandingan antara komputasi paralel dengan serial. Permasalahan yang dimiliki pada komputasi paralel dibagi menjadi bagian-bagian kecil kemudian dikimkan keberbagai komputer untuk dilakukan proses eksekusi secara bersamaan. Sedangkan pada proses komputasi serial permasalahan yang dikirimkan pada sebuah komputer untuk dilakukan proses eksekusi.

Berdasarkan perangkat yang mendukung komputasi paralel dibagi menjadi beberapa kelas antara lain:

1. *Multicore Computing*

   Multicore computing adalah sebuah processor yang dapat melakukan banyak eksekusi unit (core) pada chip yang sama.

2. *Symmetric multiprocessing*

   Symmetric multiprocessing adalah pendekatan kedua untuk *Multiprocessor Scheduling*. Pada metode ini setiap prosesor menjadwalkan dirinya sendiri (*self scheduling*). Penjadwalan terlaksana dengan menjadwalkan setiap prosesor untuk memeriksa antrian ready dan memilih suatu proses untuk dieksekusi.

3. *Distributed computing*

   Distributed computing atau yang dikenal juga dengan distribusi memory pada *multiprocessor* adalah sebuah sistem memori komputer yang terdistribusi dimana memproses element menggunakan sebuah jaringan. Adapun komputer terdistribusi ini terbagi menjadi beberapa bagian antalain

   a. *Cluster* Computing

      adalah suatu sistem perangkat keras dan perangkat lunak yang menggabungkan beberapa komputer dalam suatu jaringan dimana komputer tersebut dapat bekerjasama dalam pemrosesan suatu masalah. dari penggabungan beberapa komputer dalam satu jaringan tentu komputer tersebut dapat menghasilkan kecepatan yang sangat tinggi dalam prosesnya.

   b. *Massive Parallel Processing*

hampir sama dengan *cluster* komputer tetapi jumlah komputer yang dimiliki jauh lebih banyak dibandingkan dengan *cluster* komputer.

c. Grid Computing

merupakan salah satu dari tipe Komputasi Paralel, adalah penggunaan sumber daya yang melibatkan banyak komputer terpisah secara geografis namun tersambung via jalur komunikasi (termasuk Internet) untuk memecahkan persoalan komputasi skala besar.

4. Specialized parallel computers

Specialized parallel computers merupakan komputasi paralel tetapi menggunakan beberapa jenis perangkat tertentu yang ditujukan untuk area tertentu.

2.3 *Cluster Beowulf*

Cluster (klaster komputer) adalah kumpulan dari beberapa komputer yang dapat beroperasi secara mandiri, yang disatukan dengan jaringan komunikasi data dan mendukung perangkat lunak yang memungkinkan pengaturan rutin beban komputasi secara bersamaan yang bertujuan untuk mengerjakan satu rutin komputasi yang lebih besar(Gebali 2011).

Cluster dibangun untuk mengerjakan berbagai tugas khusus yang akan memakan waktu relatif lebih lama bila dikerjakan dengan satu komputer. Menurut (Proboyo et al. 2010) berikut beberapa contoh cluster dengan fungsi khusus:

1. Columbia (NASA) yang dibangun oleh Silicon Graphics, dipakai untuk mensimulasikan tabrakan antar galaksi spiral yang kemudian membentuk galaksi elips.
2. Roadrunner milik Departemen Energi Amerika Serikat yang dirancang IBM, terletak di Los Alamos National Laboratory, New Mexico, USA. Roadrunner dipergunakan untuk mensimulasikan proses pembelahan bahan nuklir pada senjata-senjata nuklir milik Amerika Serikat.

3. Belle, superkomputer catur yang dirancang Joe Condon dan Ken Thompson di Laboratorium Bell pada tahun 1970-1980. Belle adalah superkomputer catur terhebat di masanya dengan nilai USCF 2250.
4. Gravity Pipe, disingkat GRAPE, adalah superkomputer yang digunakan untuk menghitung efek gravitasi untuk besar massa tertentu. GRAPE dirancang dengan gaya Beowulf. GRAPE terletak di Universitas Tokyo.
5. Dalam bidang kriptografi, EFF DES cracker (disebut juga Deep Crack) adalah superkomputer yang dikhususkan untuk melakukan pencarian kunci cipher DES dari segala kemungkinan kunci DES (2^{56} kemungkinan).

Cluster Beowulf adalah *cluster* yang merupakan suatu komoditas komponen komputer yang terhubung kedalam sebuah jaringan local dengan memasangkan sebuah program kedalamnya yang memungkinkan untuk dijalankan oleh setiap komputer yang terhubung. Hasilnya adalah sebuah high-performance komputasi paralel dengan harga yang murah, karena cluster Beowulf bersifat consumer-grade adapun komponen yang digunakan merupakan komponen komputer yang sudah pernah digunakan (Aliyansyah et al. 2013).

Nama *Beowulf* merupakan rujukan dari nama komputer yang dibuat pada tahun 1994 oleh Thomas Streling dan Donald Becker di NASA. Secara normal dapat berjalan pada sistem operasi UNIX seperti BDS, solaris, Linux dsb. *Beowulf* merupakan sebuah arsitektur multi-computer yang mana digunakan untuk melakukan komputasi paralel. Pada bentuk yang standar sistem hanya memiliki satu buah server node dan satu atau lebih *client node* yang terhubung kedalam menggunakan *Ethernet*. Sistem *Beowulf* terbuat dari menggunakan bahan bahan yang murah seperti komputer yang dapat menjalankan sistem operasi UNIX dengan standar *Ethernet adapter* dan switch. Server node dapat mengontrol seluruh client yang ada serta menyediakan data kepada client node yang ada. Salah satu perbedaan mendasar pada dengan *cluster of workstation* (COW) adalah *Beowulf* menyerupai sebuah satu buah komputer ketimbang beberapa komputer. Pada beberapa kasus client node tidak memiliki keyboard maupun monitor tetapi hanya diakses melalui remote accsess. *Beowulf* bisa berisi satu

paket komputer yang ada atau juga bisa berupa *CPU* dan memori yang terhubung dengan *motherboard* pada client nodenya. Pada gambar 2.1 merupakan contoh *cluster beowulf* yang dibagun berdasarkan komputer-komputer bertipe *consumer-grade.*

Gambar 2.2 Contoh Cluster Beowulf

Pada penjelasan sebelumnya dibangun menggunakan komputer berjenis consumer-grade menitik beratkan harga yang murah dengan tujuan peningkatan komputasi. Adapun juga sistem operasi yang di gunakan bertipe FOSS (Free Open Source Software). Cluster Beowulf memiliki beberapa keuntungan karena struktur dan metode implementasianya antara lain: skalabilitas, konvegerensi, fleksibilitas konfigurasi, kemudahan, dan memiliki harga implementasi yang murah di bandingkan cluster beritpe konvesional (Wilkinson & Allen 2005).

Menurut (Sterling 2001) dalam membangun *cluster* berjenis *beowulf* memiliki beberapa karateristik, antara lain:

1. Komponen komputer yang di gunakan menggunakan komponen berjenis consumer-grade yaitu komponen komputer yang dibeli sacara bebas di pasaran serta memiliki harga yang relatif lebih murah untuk mencapai

performa komputasi terbaik, bukan komponen komputer yang di pergunakan untuk sebuah industri besar, seperti komputer server yang memiliki harga yang relatif lebih mahal. Komponen ini meliputi prosesor, *memory*, dan *network interfaces.*

2. Dalam membangun sebuah *cluster* setidaknya memiliki 1 buah head-node dan 1 buah atau lebih node pekerja, dengan lokasi geografis yang saling berdekatan.
3. Setiap node dalam cluster saling terkoneksi menggunakan saluran komunikasi data. Secara umum *cluster beowulf* menggunakan fast-ethernet ataupun gigabit-ethernet switch untuk saling menghubung kan setiap node yang ada.
4. Pada komputer node tidak dibutuhkan monitor, keyboard, maupun mouse. Hal ini karena proses yang akan dijalankan akan di remot oleh head node.
5. Pada umumnya komputer head node memiliki spesifikasi lebih baik dibandingkan komputer node. Hal ini dimaksudkan agar komputer head node dapat mengawasi kinerja node secara visual.
6. Secara umum sistem operasi yang di gunakan bertipe FOSS (Free Open Source Software). Namun tidak ditutup kemungkin untuk menggunakan sistem operasi berbayar seperti windows.
7. Terdapatnya *network file system* pada komputer head node yang berfungsi untuk membagikan file yang akan diproses oleh setiap node pada *cluster.*
8. Terdapatnya sebuah aplikasi yang dapat mengatur proses pertukaran, pengolahan, penjadwalan data (*message-passing-interface)* pada setiap node didalam *cluster.*

Komputasi dengan cluster Beowulf melibatkan 4 hal dalam pertimbangan pembangunan sistem.

1. Sistem perangkat keras,
2. Manajemen sumberdaya komputasi,
3. pustaka pemograman paralel,

4. Algoritma paralel

2.4 Message Passing Interface (MPI)

Message passing interface atau MPI merupakan sebuah sistem standar message-passing yang di rancang oleh sekelompok peneliti dan akademisi dari berbagai industri yang berfungsi pada berbagai macam komputer paralel. Standarisasi ini mendefenisikan berbagai macam syntax yang ada kedalam beberapa bahasa pemograman seperti C/C++, Java, Pyhton dan dan berbagai macam bahasa pemograman lain (Kurniawan 2010). Adapun tujuan utama dari MPI antara lain:

1. Desain sebuah antarmuka aplikasi programing (Tidak membutuhkan compiler, atau sistem implementasi library tambahan).
2. Memungkinkan komunikasi yang efisien (Menghindari menyalin data dari memori ke memori, mengizinkan adanya tumpang tindih komputasi dan komunikasi)
3. Memungkingkan untuk di implementasikan pada lingkungan yang berbeda.
4. Mempermudah melakukan pemograman dalam berbagai bahasa pemograman.
5. Antarmuka komunikasi yang handal (pengguna tidak perlu mengatasi kegagalan komunikasi. Kegagalan tersebut dapat ditangani oleh subsistem komunikasi.)
6. Antarmuka yang dapat di implementasikan pada berbagai macam vendor,tanpa adanya perumbahan yang signifikan dalam komunikasi dan perangkat lunak.
7. Semantik antarmuka harus bahasa independen.
8. Antarmuka di rancang untuk mengatasi keamanan pada proses.

MPI sebagai bahasa independen komunikasi yang di gunakan pada pemograman komputer paralel. MPI bertujuan untuk memberikan dasar virtual topologi, sinkronisasi, serta fungsi komunikasi antara serangkaian proses yang

telah di petakan ke node/serve dengan bahasa yang independen dengan sintaks dan yang tersendiri.

Secara singkat MPI merupakan sebuah library yang di gunakan pada pemograman paralel yang nantinya akan mengatur dan mengurusi bagaimana proses pembagian beban antara node, penjadwalan tugas, serta masalah komunikasi data yang tidak di sediakan pada bahasa pemograman biasa.

MPI sebagai library memiliki beberapa jenis tersendiri terganrung penggunaan bahasa pemogramannya sendiri. Untuk bahasa pemograman C/C++ dan R terdapat beberapa library yang bisa di gunakan antara lain LAMP/MPI, Open MPI, MPICH2, Microsoft MPI, HP MPI, Intel MPI. Sedangkan untuk bahasa pemograman Java terdapat library seperti The HP Java Project, dan Java MPI. Untuk bahasa pemograman .NET terdapat library MPI.NET. Untuk bahasa pemograman matlab bisa menggunakan library MPI PVM.

Untuk bahasa pemograman python terdapat beberapa library yang bisa di gunakan yaitu, MPI4PY, pyMPI, PyPar, Scientific Python. MPI4PY sebagai salah satu library yang di gunakan pada bahasa pemograman python untuk melakukan proses *messege parsing interface* memiliki beberapa kelebihan diantara library lainnya antaralain :

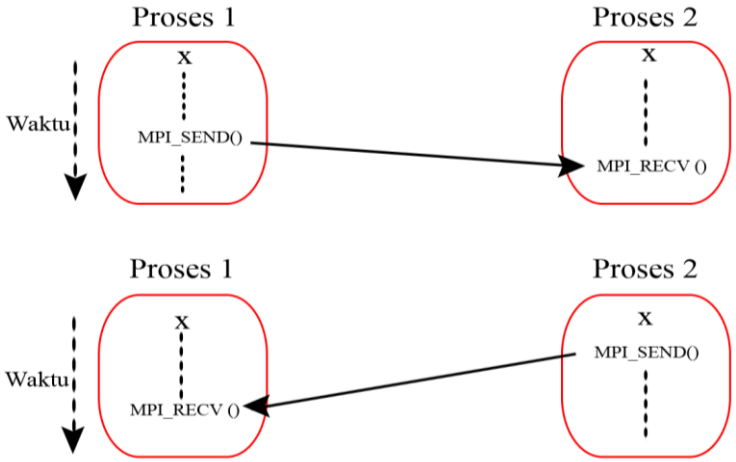
1. MPI4PY menyediakan interface yang serupa dengan MPI-2 yang merupakan standar C++ interface.
2. Pada MPI4PY berfokus untuk menterjemahkan syntax MPI kedalam bahasa pemograman python sehingga mempermudah bagi user untuk mengembangkan sebuah aplikasi berbasi MPI.
3. Syntax yang dimiliki oleh MPI4PY memiliki kesamaan dan memiliki ratusan syntax MPI sehingga ketika sudah mengetahui syntax MPI akan sanggat mudah di terapkan pada bahasa pemograman python.
4. MPI4PY dapat berkumunikasi dengan object di dalam bahasa pemograman python.

2.4.1 Rutin-rutin Dasar Message Passing

Dalam proses komputasi parallel terdapat sebuah mekanisme dalam mengirimkan pesan kepada masing masing node, dalam mengirimkan pesan terdpat dua buah mekanisme dasar yaitu *point to point* dan *collective communication*.

A Komunikasi Point to point

Mekanisme dasar sistem komunikasi pada MPI adalah proses bertukaran data pada sepasang proses dimana satu sebagai pengirim data dan satunya sebagai penerima seperti yang di jelaskan pada gambar 2.2, tahapan ini yang disebut dengan komunikasi point to point. Hampir sebagian besar komunikasi yang terjadi pada MPI di dasarkan pada komunikasi point to point.



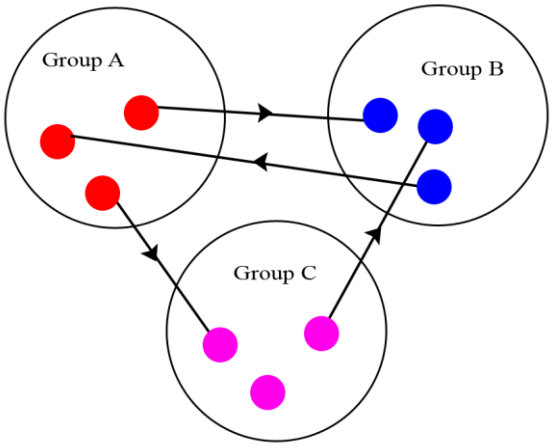
Gambar 2.3 Ilustrasi point to point communication

MPI menyediakan beberapa fungsi untuk mengirim dan menerima data baik secara bloking maupun non-bloking. Blocking send/receive adalah proses yang tidak mengembalikan nilai sampai buffer sudah penuh dengan data yang akan di kirimkan/diterima dan selanjutnya dikirim/diterima data. Ini artinya kode selanjutnya setelah blocking send/receive tidak akan di eksekusi sebelum proses data di peroses selesai. sedangkan non-blocking send/receive akan mengembalikan nilai walaupun data yang dikirm/diterima belum di eksekusi.

MPI_Send() merupakan sebuah fungsi MPI yang digunakan untuk mengirimkan data kepada komputer tujuan. MPI_Recv() merupakan sebuah fungsi MPI yang digunakan untuk menerima data yang telah dikirimkan oleh fungsi MPI_Send() yang selanjutnya data yand di tampung oleh MPI_Recv() bisa di gunakan oleh Node tujuan untuk melakukan proses.

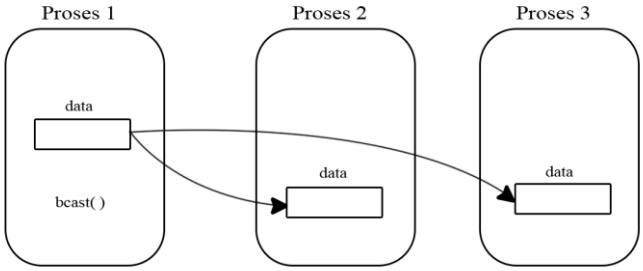
B Collective Communication

Selain *point to point communication* di dalam proses pengiriman dan penerimaan data terdapat juga *collective communication*. *Collective communication* atau komunikasi kolektif dapat di defenisikan sebagai sebuah komunikasi yang melibatkan seluruh group atau kumpulan group dari proses yang ada. Komunikasi kolektif memungkinkan untuk beberapa proses dalam komunikator yang sama untuk bertukar pesan dan melakukan proses. Dalam Proses komunikasi kolektif terdapat beberapa fungsi yang sering di gunakan antara lain *Broadcast*, *Gather*, dan *Scatter*.



Gambar 2.4 Ilustrasi Komunikasi Kolektif

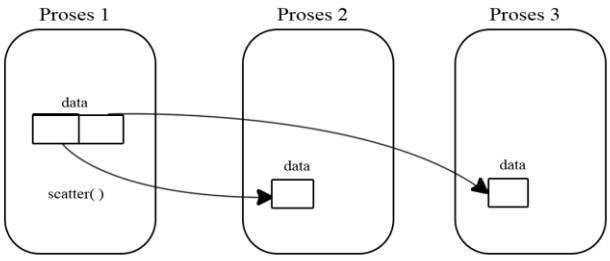
Broadcast merupakan sebuah fungsi MPI yang memungkinkan kita untuk mengirimkan data kesemua proses pada group tertentu. Operasi broadcast hanya dapat mengirimkan satu buah data yang sama ke seluruh proses dimana semua proses mendapatkan data yang sama yang di kirimkan oleh *root* proses nya.



Gambar 2.5 Ilustrasi Proses *Broadcast*

Pada gambar 2.4 menunjukan data yang dimiliki oleh proses 1 atau *root* proses akan di kirimkan keseluruh proses yang ada pada *communicator* yang spesifik yang nantinya data yang di terima oleh proses lain selain *root* proses bernilai sama.

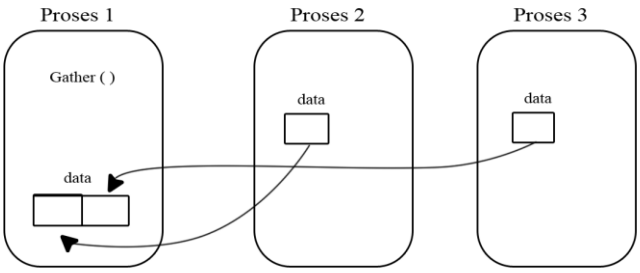
Scatter merupakan sebuah fungsi MPI yang berfungsi untuk mengirimkan data pada elemen array pada proses *root* yang ada ke proses-proses secara terpisah. Isi yang terdapat pada data elemen array-i akan dikirimkan ke proses-i, berbeda dengan proses broadcast adapun data yang sama akan dikirmkan ke seluruh proses yang ada pada sepesifik *communicator* tetapi *scatter* akan mendistribusikan data pada elemen array ke proses yang berbeda.



Gambar 2.6 Ilustrasi Proses *Scatter*

Pada gambar 2.5 menunjukan data yang dimiliki oleh proses satu atau *root* proses bersifat array dengan nilai array-i akan di kirimkan kemasing-masing proses-i sehingga data yang di terima oleh setiap proses berbeda.

Gather merupakan sebuah fungsi MPI yang berfungsi untuk mengambil semua data yang ada suatu proses pada spesifik *communicator*. *Gather* biasanya di gunakan setelah data yang di terima telah di proses komputasi. Pada dasarnya *Gather* merupakan kebalikan dari *Scatter*, data yang dikirmkan oleh *Gather* akan disimpan kedalam elemen array sesuai urutan proses yang ada pada *root* proses.



Gambar 2.7 Ilustrasi Proses Gather

Pada gambar 2.6 menunjukan proses *gather* merupakan proses kebalikan dari *scatter* adapun pada proses ini data yang dimiliki oleh proses lain selain proses *root* akan di kirimkan menuju *root* proses yang kemudian disimpan kedalam sebuah data yang bersifat array dengan nilai yang di terima nantinya akan disimpan pada array-i yang di terima oleh proses-i.

2.5 High Performance Linpack

Linpack merupakan sebuah kumpulan subroutine fortran yang dapat menyelesaikan dan menganalisa persamaan linear. Linpack berguna untuk mengukur kemampuan suatu komputer dalam melakukan komputasi. Linpack di perkenalkan oleh Jack Dongarra berfungsi untuk mengukur kecepatan komputer yang dimiliki nya dalam menyelesaikan permasalahan nilai kepadatan n dari sistem persamaan linear $Ax = b$ yang mana permasalahaan ini merupakan permasalahan umum diantara para insinyur.

Adapun target dari Linpack adalah memperkiran seberapa cepat sebuah komputer dapat menyelesaikan permasalahan nyata. Hal terserbut merpakan permasalahan dimana tidak ada satupun proses komputasi dapat memberikan hasil

yang akurat pada sebuah sistem komputer. Meskipun demikian Linpack dapat memberikan data akurat nilai tertinggi performa suatu sistem komputer. Puncak dari performa adalah nilai maximal yang dimiliki oleh nilai teori yang dapat di capai, hal ini berkaitan dengan perhitungan ferkuensi mesin yang dimiliki, jumlah cycle perdetik yang dimiliki, dan nilai waktu yang dimiliki pada setiap cycle perdetiknya. Meskipun demikin pada kenyataan nya nilai performa nyata selalu lebih rendah dari pada nilai puncak yang di miliki.

Linpack juga di gunakan pada standarisasi supercomputer yang ada di dunia. Top500 merupakan sebuah project pengukuran performa *supercomputer* dunia dibuat (TOP500 n.d.). Tujuan project ini adalah menyediakan sebuah basis pengukuran standarisasi yang handal untuk melacak dan mendeteksi trend dari high-performance computing di dunia. Dalam Pengujian nya HPL menggunakan persamaan Strassen algorithm. Adapun persamaan nya antara lain:

$$\frac{\|\boldsymbol{Ax-b}\|}{\|\boldsymbol{A}\|\|\boldsymbol{x}\|\boldsymbol{n}\epsilon} \leq \boldsymbol{O}(\mathbf{1}) \qquad (2.4)$$

High Performance Linpack adalah sebuah paket software yang dapat menyelesaikan permasalahan linear dengan ketepatan 64 Bits pada sebuah cluster komputer. Adapun paket yang di sediakan oleh HPL seperti menguji waktu sebuah program ketika menjalan sekumpulan tugas. Adapun variable yang dibutuhkan HPL dalam menguji sebuah performa komputer paralel antara lain :

1. N : Nilai n merupakan nilai jumlah seberapa besar permasalahan yang akan di ujikan terhadap sistem. Nilai n juga berguna untuk mengetes seberapa besar performa yang dimiliki komputer. Penggunaan nilai N yang disarankan adalah 80% dari jumlah memory yang digunakan pada sistem. Jika jumlah permasalahan (N) yang di pilih terlalu besar maka proses tersebut akan memakan memori swap yang ada dan dapat menyebabkan penurunan performa (Innovative Computing Laboratory 2016). Adapun rumus untuk menghitung nilai N adalah :

$$\mathbf{N} = \sqrt{\frac{\boldsymbol{jumlah\ memory\ yang\ di\ gunakan\ (GB)} * \mathbf{1024^3}}{\mathbf{8}}} \qquad (2.5)$$

2. NB : nilai nb merupakan nilai block size yang digunakan untuk pendistribusian data, semakin kecil suatu nilai NB semakin besar load balance yang di miliki tetapi jika nilai NB terlalu kecil maka waktu komputasi akan menjadi lebih lama. Adapun nilai NB yang di rekomendasi kan antara lain berkisar 32 – 256.
3. PxQ : merupakan nilai dari ukuran dari jumlah processor yang dimiliki oleh cluster. Nilai P sebaiknya memiliki lebih kecil sedikit dari nilai Q.

Sebagai contoh terdapat sebuah cluster yang terdiri dari 512 node dengan masing-masing node memiliki 12 GB RAM dan menggunakan processor Intel Nehalem yang memiliki kecepatan 2,93 GHZ serta memiliki core sebanyak 8 buah dan memiliki *operation/cycle* sebanyak 4. Adapun cara untuk mengetahui nilai N, NB, dan PxQ antaralain:

1. Nilai N adalah nilai dari rumus (2.5) sebagaimana diketahui pada contoh soal memory yang di gunakan adalah sebanyak 512*12 yaitu 6144 GB

$$N = \sqrt{\frac{6144 * 1024^3}{8}}$$

$$N = \sqrt{\frac{6597069766656}{8}}$$

$$N = \sqrt{824633720832}$$

$$N = 908093,45$$

Adapun nilai N yang digunakan adalah dengan effisien 80% dari kinerja yang ada sehingga menjadi.

$$N = 908093,45 * 0,8$$

$$N = 726475$$

2. Nilai NB merupakan nilai block size yang digunakan untuk pendistribusian data, nilai NB sendiri berkisar dari 96,104,112,120,128 … 256. Nilai NB sendiri bebas untuk di pilih berdasarkan range tersebut.

3. Nilai PxQ merupakan nilai dari jumlah core yang dimiliki oleh cluster, seperti di ketahui jumlah core pada cluster adalah 512*8 yaitu 4096. Untuk memilih nilai PxQ dimana nilai p dan q hampir sama tetapi nilai p lebih kecil dibandingkan nilai q dan jika di kalikan maka hasil nya sama dengan jumlah core yang ada pada cluster.

Tabel 2.5 Perhitungan Nilai PxQ

P	*	Q
1	*	4096
2	*	2048
4	*	1024
8	*	512
16	*	256
32	*****	**128**
64	*	64

Pada Tabel 2.5 terlihat nilai PxQ adalah 32 dan 128 dimana nilai tersebut hampir sedikit sama tetapi nilai p lebih kecil dibandingkan dengan nilai q.

Nilai yang sudah di dapatkan tersebut bisa di pergunakan pada file konfigurasi HPLinpack yang akan di jalankan nanti nya pada saat melakukan pengukuran performa cluster, adapun contoh file konfigurasi HPLinpack bisa dilihat pada gambar 2.7.

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL-13.out      output file name (if any)
file            device out (6=stdout,7=stderr,file)
1               of problems sizes (N)
829248           Ns
1               of NBs
224              NBs
0               PMAP process mapping (0=Row-,1=Column-major)
1               of process grids (P x Q)
32               Ps
128              Qs
16.0            threshold
1               of panel fact
0               PFACTs (0=left, 1=Crout, 2=Right)
1               of recursive stopping criterium
4               NBMINs (>= 1)
1               of panels in recursion
2               NDIVs
1               of recursive panel fact.
0               RFACTs (0=left, 1=Crout, 2=Right)
1               of broadcast
0               BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1               of lookahead depth
0               DEPTHs (>=0)
2               SWAP (0=bin-exch,1=long,2=mix)
128             swapping threshold
0               L1 in (0=transposed,1=no-transposed) form
0               U  in (0=transposed,1=no-transposed) form
1               Equilibration (0=no,1=yes)
8               memory alignment in double (> 0)
```

Gambar 2.8 Contoh file konfigurasi HPLinpack

Pada gambar 2.7 bagian yang berwarna merah merupakan nilai yang dapat di ganti dengan nilai N,NB, dan PxQ yang telah di cari. Adapun output keluaran dari HPLinpack bisa kita lihat pada gambar 2.8 berikut.

```
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR00L2L4      829248   224    32   128            9103.52             4.177e+04
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0005126 ...... PASSED
================================================================================
```

Gambar 2.9 Contoh Output HPLinpack

Pada gambar 2.8 terutama pada bagian merah merupakan hasil atau nilai dari cluster tersebut. Satuan yang di gunakan pada output tersebut menggunakan flops. Flops atau *floathing points per seconds* merupakan satuan jumlah perhitungan yang dapat di lakukan oleh sebuah perangkat komputasi terhadap bilangan *floathing point* pada satuan waktu, *floathing point* merupakan format bilangan yang merepresentasikan nilai yang sangat besar maupun kecil. FLOPS merupakan satuan ukuran kecepatan kinerja suatu mikroprosesor dalam satu aplikasi ilmiah. Pada contoh gambar di atas nilai kecepatan berkisar 41,77 TFlops, jadi cluster tersebut bisa melakukan perhitungan algoritma persamaan Strassen sebanyak 41 kali teraflop perdetik nya.

2.6 Kajian Penelitian Terkait

Berikut adalah beberapa penelitian terkait tentang algoritma RSA dan juga komputasi paralel menggunakan *:*

1. Penelitian tentang penggunaan algortima paralel pada AES yang berjudul "*Parallel AES Algorithm for Fast Data Encryption on GPU"* oleh Deguang Le, Jinyi Chang, Xingdou Gou, Ankang Zhang, Conglan Lu tahun 2010. Penggunaan algoritma AES pada CPU menghasilkan performa yang buruk oleh karena itu digunakan lah Graphics Processing Units (GPU) untuk memenuhi kebutuhkan akan kecepatan data enkripsi selain itu untuk meningkatkan kecepatan proses digunakanlah pemerosesan paralel pada algoritma AES. Kesimpulan dari penelitian tersebut adalah hasil implementasi algoritma paralel AES menghasilkan 7x lebih cepat daripada penggunaan algoritma AES pada sebuah komputer.
2. Penelitian tentang penggunaan algortima paralel pada AES yang berjudul *"Improvising performance of Advanced Encryption Standard Algorithm"* oleh Vandan Pendli, Mokshitha Pathuri, Subhakar Yandrathi, Abdul Razaque tahun 2016. Terdapatnya batasan performa pada algoritma AES seperti penggunaan memory yang besar serta waktu komputasi yang lama memerlukan sebuah solusi untuk mengatasi nya. Penggunaan pemerosesan paralel merupakan salah satu solusi untuk mengatasi permasalahan tersebut. Kesimpulan dari penelitian tersebut adalah penggunaan pemerosesan paralel pada algoritma AES dapat mempercepat 40-45% dari pada penggunaan pemerosesan serial.
3. Penelitian tentang penggunaan algoritma paralel pada blowfish yang berjudul *"Enhancing Blowfish File Encryption Algorithm through Parallel Computing on GPU"* oleh Tejal Mahajan, Shraddha Masih tahun 2015. Semakin besarnya file yang digunakan pada proses kriptografi membuat waktu komputasi yang dihasilkan semakin

besar, penggunaan CPU dalam memproses enkripsi file sangat lamban oleh karena itu digunakanlah GPU karena memiliki performa yang lebih baik. CUDA merupakan sebuah *programing* model yang dapat mengatur penggunaan sumber daya GPU. Blowish digunakan sebagai algoritma kriptografi karena bersifat *open source*. Selain itu digunakan juga pemerosesan paralel untuk meningkatkan waktu komputasi menggunakan algoritma blowfish tersebut . Kesimpulan dari penelitian tersebut adalah penggunaan pemerosesan paralel pada algoritma blowfish dapat menurunkan waktu proses enkripsi dan dekripsi. Penggunaan GPU memberikan peforma yang lebih baik dibandingkan implementasian menggunakan CPU.

4. Penelitian tentang penggunaan algoritma paralel pada algoritma RSA berjudul *"A Promising Parallel Algorithm to Manage the RSA Decryption Complexity"* oleh Abu Asaduzzaman, Deepthi Gummadi, and Puskar Waichal tahun 2015. Dalam menyediakan kemanan yang lebih pada algoritma RSA membutuhkan daya komputasi yang besar untuk mengatasi nya digunakanlah pemerosesan paralel yang dijalankan pada GPU selain itu digunakan CUDA sebagai salah satu *programing model* yang digunakan untuk mengatur penggunan sumber daya GPU. Kesimpulan dari penelitian tersebut adalah pengguna workstation memiliki performa yang lebih baik dari pada PC dan Laptop karena workstation memiliki jumlah core lebih banyak dibandingkan dengan PC dan Laptop dimana dapat menangani lebih banyak thread sehingga terjadi pengingkatan kinerja RSA dalam proses dekripsi dan sebagian dari proses enkripsi.
5. Penelitian tentang algoritma RSA yang berjudul "*A Study of Encryption Algorithms AES, DES, and RSA for Security"* oleh Dr. Prerna Mahajan dan Abhishek Scahdeva Tahun 2013. Kesimpulan dari penelitian tersebut adalah berdasarkan pengujian menggunakan file text, algoritma AES menggunakan waktu tersingkat untuk

melakukan enkripsi, sedangkan untuk algoritma RSA menggunakan waktu terlama untuk melakukan enkripsi.

6. Penelitian tentang algoritma RSA yang berjudul "*A Modified RSA Encryption Technique Based on Multiple Public Keys"* Tahun 2013 oleh Amare Anagaw Ayele dan Dr. Vuda Sreenivasaro. Kesimpulan dari penelitian tersebut adalah penggunaan 2 kunci public yang dikirim secara terpisah membuat penyerang tidak dapat mendapatkan cukup pengetahuan tentang kunci serta tidak dapat melakukan dekripsi pesan. Serta tujuan dari algoritma RSA digunakan untuk sistem yang membutuhkan kemanan tingkat tinggi dengan mengeyampingkan kecepatan yang tinggi.
7. Penelitian tentang algoritma RSA yang berjudul "*Studi Kasus Penggunaan Algoritma RSA Sebagai Algoritma Kriptografi yang Aman"* tahun 2009 oleh Zainal arifin memiliki kesimpulan adalah sebagai berikut :
    a. RSA merupakan salah satu solusi yang baik untuk mengatasi masalah keamanan dan kerahasiaan data dengan tingkat kemanan yang tinggi karena belum ditemukan algoritma yang efisien untuk memecahkanya.
    b. Semakin panjang suatu kunci publik, maka usaha yang harus dikeluarkan untuk memecahkan kunci tersebut akan lebih lama. Pada saat ini dianjurkan untuk menggunakan sistem RSA dengan kunci publik 1024 bit.
8. Penelitian tentang algoritma RSA yang berjudul "*Penggunaan Algritma RSA dengan Metode The Sieve Of Eratoshenes dalam Enkripsi dan Dekripsi Perngiriman Email"* tahun 2013 oleh Muhammad Safri Lubis, Mohammad Andri Budiman, dan Karina Lolo Manik menghasilkan kesimpulan kunci privat yang dibangkitkan selalu berubah-ubah nilainya walau terkadang nilai kunci public yang dihasilkan nilainya sama. Pembangkitan bilangan acak diimplementasikan dalam suatu fungi *Math.random*. Sementara

pada proses penentuan bilangan prima digunakan fungsi khusus *Math.sqrt* yang sesuai dengan metode *The Sieve of Eratosthenes* untuk menentukan nilai akar kuadrat dari suatu bilangan sehingga di harapkan pengecekan bilangan prima bisa dilakukan lebih cepat dan menghemat memori.

9. Penelitian tentang algoritma RSA yang berjudul "*Analysis of RSA Algorithm Using GPU Programing"* Tahun 2014 oleh Sonam Majan dan Maninder Singh menghasilkan kesimpulan penggunaan bilangan prima dengan nilai yang kecil membuat algoritma RSA menjadi rentan terhadap serangan, sedangkan penggunaan bilangan prima yang besar akan membuat waktu komputasi semakin melambat dan meningkatkan biaya.
10. Penelitian tentang komputasi paralel dengan judul "*Implementasi Portal Komputasi GRID untuk Multi Kluster"* tahun 2013 oleh Benediktus Anindito dan F.X Arunanto memiliki kesimpulan pada pengimplementasian menggunakan Gridsphere dan Vine Tollkit, portal yang dapat terhubung ke berbagai kluster sekaligus. Pemerograman paralel yang disediakan infrastruktur grid ini dapat mempercepat waktu eksekusi suatu program dengan data masukan yang cukup besar.
11. Penelitian tentang komputasi paralel dengan judul *"Membangun (High Performance Computing)*" tahun 2010 oleh Ardrian Proboyo, Wawan Yunanto, dan Yuli Fitrisia menghasilkan kesimpulan :
    a. Penerapan cluster Beowulf yang telah dibangun untuk menjalankan aplikasi benchmark LINPACK terbukti mempercepat penyelesaian masalah yang diberikan, sampai jumlah node tertentu.
    b. Dalam menjalankan aplikasi benchmark LINPACK, penambahan komputer tidak selalu meningkatkan kinerja cluster.

c. Perbedaan tipe prosesor tidak memberikan perbedaan kinerja yang signifikan dalam sistem cluster Beowulf saat menjalankan aplikasi benchmark LINPACK sesuai metode pengujian.
d. Proses render menggunakan Blender yang dijalankan pada cluster Beowulf yang telah dibangun menunjukkan proses render berlangsung lebih cepat setiap pertambahan compute node, meski percepatan tersebut tidak sama besarnya.

12. Penelitian tentang komputasi paralel dengan judul *"Komputasi Paralel Integral Definit Rangkap Tiga dengan Metode Monte Carlo di Cluster Bewoulf"*tahun 2013 oleh Muhammad Zulhaj Aliyansyah, Mahfudz Shidiq, dan Muhammad Aswin menghasilkan kesimpulan:
    a. Peningkatan kecepatan komputasi integral definit rangkap tiga metode Monte Carlo meningkat seiring pertambahan cacah proses jika cacah proses tidak melebihi cacah core, sementara kesangkilannya menurun secara step saat cacah proses melebihi kelipatan cacah core dalam node komputasi primer.
    b. Peningkatan kecepatan meningkat seiring pertambahan cacah node dalam komputasi integral definit rangkap tiga metode Monte Carlo. Kesangkilan komputasi integral definit rangkap tiga metode Monte Carlo konstan saat menggunakan node komputasi primer dan menurun saat menurutsertakan node komputasi sekunder.