PERBANDINGAN PRESTASI ANTARA MESSAGE PASSING INTERFACE JAVA (MPIJAVA) DAN JAVA OBJECT PASSING INTERFACE (JOPI)

Zafril Rizal bin M.Azmi, Kamalrulnizam bin Abu Bakar, Mohd Seth bin Yaacob.
Jabatan Sistem dan Komunikasi Komputer, Fakulti Sains Komputer dan Sistem Maklumat,
Universiti Teknologi Malaysia.
zafrilrizal@yahoo.com, knizam@utm.my, seth@utm.my

ABSTRAK

Pengkomputeran selari merupakan satu kaedah alternatif yang murah bagi membangunkan satu sistem atau untuk tujuan penyelidikan yang memerlukan komputer berprestasi tinggi. Pengaturcaraan yang dibuat di dalam persekitaran selari memerlukan pustaka seperti MPI sebagai tapak kepada pembangunannya. Namun begitu, disebabkan bahasa pengaturcaraan Java semakin popular dikalangan pembangun sistem, pustaka seperti MPIJAVA dan JOPI yang menyokong pembangunan sistem selari semakin gemar digunakan. Walaubagaimanapun, pembangun sistem selari sering mengalami masalah untuk memilih pustaka berorientasikan objek yang sesuai. Oleh itu, kajian yang bertajuk "*Perbandingan prestasi antara Message Passing Interface Java (MPIJAVA) dan Java Object-Passing Interface (JOPI).*" dibuat untuk membandingkan prestasi antara MPIJAVA dan JOPI. Perbandingan dibuat berdasarkan kepada masa komunikasi, kadar *bandwitdh*, *overhead* perisian dan masa keseluruhan pemprosesan yang diuji menerusi beberapa siri pengujian. Hasil daripada kajian ini mendapati MPIJAVA sesuai digunakan sekiranya saiz data adalah kecil, tetapi untuk saiz data yang lebih besar, JOPI lebih efektif digunakan berbanding MPIJAVA walaupun ia mempunyai overhead perisian yang lebih tinggi berbanding MPIJAVA. Kajian ini dapat membantu pembangun sistem selari untuk memilih perpustakaan yang terbaik untuk sistem selari yang mereka dibangunkan.

Kata kunci: Pengkomputeran selari, Java, MPIJAVA, JOPI, prestasi.

ABSTRACT

The parallel computing is a cheap alternative for system development or for research that needed to be done by a super computer. Coding in parallel environment will need a library such as MPI as a platform to the development process. But recently, because of Java programming language has becoming popular among the system developer, library such as MPIJAVA and JOPI that support the development of parallel system have been increasingly used. Unfortunately, parallel system developer always facing a problem in deciding which object orientation library is suitable for their system. That is why this research titled "*Performance comparison between Message Passing Interface Java (MPIJAVA) and Java Object-Passing Interface (JOPI).*" is done to compare the performance between MPIJAVA and JOPI. The comparison is done based on the communication time, the effectiveness of bandwidth, software overhead and overall processing time that have been gain from a series of testing. From the result, it is clearly shown that MPIJAVA is suitable to be use if the size of data is small. In the other hand, JOPI is more effective if the size of data is big although it has an overhead that is bigger than MPIJAVA. The result from this research will help the parallel system developer to choose the best library for their parallel system.

Key words: Parallel computing, Java, MPIJAVA, JOPI, performance.

1.0 Pengenalan

Peningkatan permintaan terhadap pengkomputeran berkuasa dan berprestasi tinggi telah mendorong kepada peningkatan di dalam bidang pemprosesan selari (Nader *et al.* 2002; Baker *et al*. 1999). Pemprosesan selari merupakan satu teknik alternatif selain daripada penggunaan komputer termaju atau *super computer* untuk menyelesaikan satu masalah pengiraan dalam persekitaran pemprosesan yang pantas. Selain boleh meningkatkan keupayaan untuk memproses sesuatu program, teknik pemprosesan selari juga menawarkan kos yang lebih rendah berbanding komputer termaju. Salah satu daripada cara perlaksanaan kaedah pemprosesan selari adalah dengan menggunakan sekumpulan stesen kerja yang terdiri daripada nod-nod pemprosesan yang akan berkongsi bebanan kerja. Nod-nod ini tidak bergantungan antara satu sama lain dari aspek ingatan dan unit pemprosesan. Ia hanya berhubungan antara satu sama lain untuk menyelesaikan

satu masalah yang telah dipecahkan dan dikhususkan kepada nod tertentu sahaja. Ini bermaksud nod-nod ini mempunyai ingatan dan unit pemprosesan sendiri tetapi berkongsi bebanan kerja untuk menyelesaikan sesuatu masalah dengan kepantasan yang lebih berbanding teknik pemprosesan sesiri.

Oleh kerana penggunaan Java juga semakin popular dikalangan pembangun sistem, semakin ramai pengkaji cuba membangunkan dan menerokai kebarangkalian untuk menggunakan Java bagi sistem berbilang pemproses. Di antara perpustakaan penghantaran mesej terbaru untuk pengaturcaraan selari adalah JOPI. JOPI baru diperkenalkan dan masih baru dalam bidang pengkomputeran selari berdasarkan kepada maklumat yang ditemui termasuk manual pengguna versi beta 1.0. Di antara kelebihan JOPI adalah nahu untuk kaedah komunikasinya adalah lebih mudah dan senang diingat. Struktur data yang kompleks juga boleh dihantar dengan menggunakan langkah-langkah yang ringkas. Selain daripada itu, penggunaan objek untuk penghantaran struktur data yang kompleks dan besar di antara nod selari menjadikan saiz kod yang dibangunkan lebih kecil. Akhir sekali, pengasingan antara masalah-penyelesaian daripada proses penselarian menjadikannya mudah untuk pemeliharaan dan pemuliharaan (Nader *et al.* 2002).

Berbanding dengan JOPI, MPIJAVA telah agak lama bertapak dalam bidang pengkomputeran selari dan telah banyak eksperimen dan pengujian dilakukan ke atasnya (Al-Jaroodi *et al*. 2001, 2002, 2003a, 2003b). Berdasarkan kajian, versi terbaru MPIJAVA di pasaran adalah 2.0. Walaubagaimanapun, dari segi teori, MPIJAVA mempunyai kelebihan orientasi objek yang sama dengan JOPI, namun dari sudut implimentasinya, mungkin jauh berbeza. Tiada lagi perbandingan secara terus dibuat antara JOPI dan MPIJAVA, namun perbandingan antara JOPI dengan MPI + C dan antara MPIJAVA dengan MPI + C pernah dibuat oleh pengkaji-pengkaji luar.

Sebagai contoh, di dalam penulisan Nader *et al.* (2002), terdapat satu seksyen di mana perbandingan prestasi komunikasi, aplikasi pendaraban matriks dan masalah Travelling Salesman Problem (TSP) telah dibuat antara JOPI dan MPI + C. Secara rawak, prestasi MPI + C lebih baik dari sudut masa. Dari segi perbandingan antara MPIJAVA dan MPI + C pula, di dalam eksperimen yang dijalankan Rizal, (2005), didapati keupayaan MPI + C juga lebih baik berbanding MPIJAVA. Namun begitu, MPI + C bukanlah antara muka yang berorientasikan

objek. Oleh itu, tujuan kajian ini dijalankan adalah untuk membandingkan antara dua antara muka yang berorientasikan objek.

2.0 Pengkomputeran Selari

Secara amnya, perisian komputer yang ditulis untuk pengkomputeran sesiri hanya boleh diproses oleh satu komputer yang mempunyai satu unit pemproses (CPU). Masalah diselesaikan oleh satu set arahan, yang dijana satu persatu oleh CPU. Hanya satu arahan boleh diproses dalam satu-satu masa (Westein *et al*. 2003).

Dewasa ini, telah wujud peningkatan permintaan terhadap pengkomputeran selari. Menurut DeCegama (1989), dua faktor utama yang menyumbang kepada pemilihan pengkomputeran selari adalah penjimatan masa dan juga keupayaannya dalam menyelesaikan masalah-masalah yang besar. Grove *et al*. (2004), dalam penulisannya juga menyatakan pendapat yang sama berkenaan dengan penjimatan masa. Beliau berpendapat bahawa sebab utama kepada penggunaan kaedah pengkomputeran selari adalah untuk mengurangkan masa pengiraan yang diperlukan. Penggunaan kaedah yang boleh memendekkan masa pengiraan adalah penting untuk memastikan sesuatu program itu tidak menjadi program yang memerlukan masa larian yang panjang sebelum dapat menghasilkan input. Menyedari kepentingan ini, ramai pengkaji berlumba-lumba untuk membangunkan seni bina, platform dan juga algoritma yang dapat meningkatkan prestasi komunikasi dan pengiraan data secara selari (Nader *et al*. 2002).

Secara umumnya, pengkomputeran selari adalah penggunaan serentak pelbagai sumber pemprosesan untuk menyelesaikan satu masalah pengiraan (Guster *et al*. 2003). Sumber-sumber pemprosesan termasuklah komputer dengan berbilang pemproses, sekelompok komputer yang dihubungkan oleh rangkaian dan kombinasi antara kedua-duanya. Sumber-sumber ini menggunakan teknik atau kaedah pengkomputeran teragih (Hennessy dan Patterson, 2002) di mana masalah pengiraan yang dulu hanya boleh diselesaikan oleh *super computer*, kini boleh dipecah-pecahkan kepada sekumpulan kerja yang lebih kecil (Guster *et al*. 2003).

Masalah yang diselesaikan oleh pengkomputeran selari kebiasaannya mempunyai ciri-ciri tidak bergantungan atau boleh dipecahkan kepada beberapa set kerja yang boleh diselesaikan secara serentak dan juga boleh diselesaikan dengan masa yang lebih pantas berbanding jika

menggunakan teknik pemprosesan sesiri. Di antara contoh-contoh implimentasi pemprosesan selari adalah pangkalan data selari, perlombongan data (*data mining*) (Zomaya *et al*. 1999; Dhillon *et al*. 1999), enjin carian web (Orlando *et al*, 2001) dan pemprosesan imej dan grafik (Seinstra *et al*. 2002; Gennart *et al*. 1999).

2.1 Antara muka Penghantaran Mesej (MPI)

Untuk memastikan pengkomputeran selari boleh diimplimentasikan pada sekelompok komputer atau komputer *cluster*, pustaka yang mengandungi rutin-rutin yang mengawal aktiviti pengkomputeran selari harus dipasang pada sistem komputer. Salah satu daripada pustaka yang popular dan sering digunakan adalah MPI (Baker *et al*. 1999; Baiardi *et al*. 2004; Carpenter *et al*. 2000).

Antara muka Penghantaran Mesej atau MPI (*Message Passing Interface*) merupakan satu antara muka yang membolehkan komputer-komputer yang berlainan seni bina digabungkan untuk melakukan aktiviti pemprosesan selari. MPI dibangunkan untuk tujuan menjadi piawai kepada pengaturcaraan pemprosesan selari. Dengan erti kata lain, ianya haruslah fleksibel serta praktikal dan boleh digunakan oleh semua jenis seni bina komputer.

Sejarah pembangunan MPI bermula pada tahun 1993-1994 oleh sekumpulan pengkaji yang mewakili bidang industri, kementerian dan akademik daripada Amerika dan Eropah. Melalui kajian mereka, maka terhasil MPI yang merupakan piawai berasaskan konsep penghantaran mesej (*message passing*) (Dagum dan Menon, 1998).

MPI mempunyai beberapa ciri istimewa yang menyebabkan penggunaannya digemari di dalam pembangunan aplikasi selari. Antaranya adalah aturcara yang dibangunkan boleh digunakan pada sebarang seni bina komputer. MPI juga menjadikan operasi komunikasi lebih mudah dimana sebarang operasi penukaran data dan pengurusan protokol komunikasi di antara komputer dilakukan oleh MPI tanpa perlu campur tangan pengguna. Sumber-sumber aplikasi MPI juga boleh diperolehi dengan mudah serta percuma terutamanya melalui internet. Selain daripada itu fungsi-fungsi MPI boleh dipanggil dalam aturcara C, C++ dan FORTRAN (Carpenter *et al*. 2000; Guster D *et al*. 2003) .

MPI menggunakan konsep ingatan teragih (*distributed memory*). Konsep ingatan teragih ini diterangkan oleh Palis *et al.* (1996) sebagai satu strategi yang digunakan untuk membahagikan satu program atau data kepada modul-modul atau kerja dan menjadualkan kerja-kerja ini kepada pemproses-pemproses yang ada di dalam sistem komputer. Di dalam ingatan teragih, kerja-kerja yang telah diagihkan kepada pemproses-pemproses yang berlainan ini berkomunikasi dengan menggunakan konsep penghantaran mesej. Govett (2003), menyatakan konsep ingatan teragih semakin banyak digunakan untuk menyelesaikan masalah kerana ia menawarkan kos yang lebih murah berbanding penggunaan konsep ingatan terkongsi (*shared memory*). Menurut Gibbons (1996), konsep ingatan terkongsi adalah apabila pemproses-pemproses berkomunikasi dengan cara membaca dan menulis pada lokasi di dalam ingatan yang dikongsi bersama. Semua pemproses mempunyai keupayaan akses yang sama.

Walaubagaimanapun, beberapa kajian telah dibuat yang menunjukkan MPI boleh diimplimenkan secara ingatan terkongsi. Sebagai contoh, Joubert *et al.*(2004), melalui kajiannya menunjukkan bahawa penggunaan ingatan terkongsi secara maya boleh dibuat pada persekitaran ingatan teragih dengan beberapa modifikasi. Selain itu, Behrens *et al.* (2004), membuktikan bahawa konsep ingatan terkongsi ini boleh digunakan untuk menggantikan konsep perkongsian teragih sedia ada. Kajian-kajian ini dibuat untuk mengatasi kelemahan komunikasi yang wujud disebabkan *overhead* (David *et al.*2002; Al-Jaroodi *et al.*2002) oleh penggunaan konsep ingatan teragih.

Untuk memastikan MPI boleh berfungsi dengan baik, rutin-rutin tertentu perlu dipanggil sewaktu proses pengaturcaraan. Rutin-rutin ini dipanggil berdasarkan kesesuaian aktiviti yang ingin dibuat. Antara rutin-rutin MPI yang sering digunakan oleh pengaturcara untuk menyelesaikan sesuatu masalah adalah seperti dalam Jadual 2.1.

Jadual 2.1 : Jadual rutin asas MPI (berdasarkan MPICH (iaitu MPI dengan bahasa pengaturcaraan C atau C++)) (Gropp *et al*. 1996).

Fungsi MPI	**Kegunaan**
MPI_INIT( )	Menandakan permulaan aktiviti MPI
MPI_COMM_SIZE	Menentukan bilangan pemproses yang terlibat.
MPI_COMM_RANK	Menetapkan id setiap pemproses
MPI_SEND	Menghantar data daripada nod *Slave* ke nod *Master* atau daripada nod *Master* ke nod *Slave*
MPI_RECV	Menerima data daripada nod *Slave* oleh nod *Master* atau daripada nod *Master* oleh nod *Slave*
MPI_FINALIZE( )	Menandakan berakhirnya penggunaan MPI atau aktiviti MPI dalam aturcara.

2.1.1 MPIJAVA

Dewasa ini, penggunaan Java telah menjadi semakin popular di dalam bidang saintifik, komputasi kejuruteraan dan juga pengkomputeran selari (Baker *et al.* 1998). Ini kerana penggunaan Java adalah mudah, efisyen dan boleh diimplimenkan pada sebarang platform (Baker *et al.* 1998; Taboada *et al.* 2003). Selain daripada itu, peningkatan prestasi pada pengkompil dan mekanisma komunikasi juga telah menyumbang kepada peningkatan penggunaan Java dalam bidang komputer berprestasi tinggi (*high performance computing*)(Nelisse *et al.* 2001). Kelebihan yang ada pada Java ini telah menjadikan pustaka penghataran mesej Java satu alternatif untuk membangunkan aplikasi selari dan teragih (Taboada *et al.* 2003)

MPIJAVA merupakan salah satu pustaka berorientasikan objek, yang memenuhi piawaian MPI (Baker *et al.* 1999). Ia mengadaptasikan model penghantaran mesej (Dagum dan Menon, 1998) untuk membangunkan aplikasi pada persekitaran Sistem Ingatan Teragih (Jaroudi *et al*.2003). Antarmuka ini telah dibangunkan sebagai sebahagian daripada projek HPJava. MPIJAVA seharusnya memiliki ciri-ciri seperti serasi dengan sebarang platform yang mempunyai aplikasi untuk pembangunan Java dan juga pada persekitaran MPI.

Secara umumnya pustaka MPIJAVA merupakan satu alternatif kepada MPI iaitu ianya membolehkan teknik-teknik pada MPI diimplemenkan dengan menggunakan bahasa pengaturcaraan Java selain daripada C, C++ dan juga FORTRAN (Carpenter *et al*. 2000; Guster D *et al.* 2003). MPIJAVA tidak boleh berdiri dengan sendiri tanpa kewujudan pustaka MPI di

dalam sesuatu sistem dan juga tanpa pengkompil Java yang sesuai. Pada dasarnya MPIJAVA menggunakan sepenuhnya konsep yang berorientasikan objek, tetapi masih mengekalkan ciri-ciri yang terdapat pada MPI. Oleh itu rutin-rutin yang dipanggil adalah berlainan daripada MPI walaupun mempunyai fungsi yang sama kerana MPIJAVA menggunakan pustaka yang berlainan berbanding MPI. Beberapa rutin asas yang penting serta fungsinya di dalam pembangunan aturcara selari menggunakan MPIJAVA serta perbandingan dan rutin MPI ditunjukkan pada Jadual 2.2.

Jadual 2.2 : Perbandingan antara rutin MPI dan MPIJAVA

Fungsi MPI	**Fungsi MPIJAVA**	**Kegunaan**
MPI_INIT( )	MPI.INIT( )	Menandakan permulaan aktiviti MPI
MPI_COMM_SIZE	MPI.COMM_WORLD.SIZE( )	Menentukan bilangan pemproses yang terlibat.
MPI_COMM_RANK	MPI.COMM_WORLD.RANK( )	Menetapkan id setiap pemproses
MPI_SEND	MPI.COMM_WORLD.SEND	Menghantar data daripada nod *Slave* ke nod *Master* atau daripada nod *Master* ke nod *Slave*
MPI_RECV	MPI.COMM_WORLD.RECV	Menerima data daripada nod *Slave* oleh nod *Maste*r atau daripada nod *Master* oleh nod *Slave*
MPI_FINALIZE( )	MPI.FINALIZE( )	Menandakan berakhirnya penggunaan MPI atau aktiviti MPI dalam aturcara.

2.2 Java Object-Passing Interface (JOPI)

JOPI adalah pengaturcaraan selari selain daripada MPIJAVA yang diimplimentasikan dalam Java dan ia menyediakan fungsi-fungsi yang bersesuaian kepada pengaturcara Java untuk membangunkan aplikasi selari berasaskan objek pada sistem *cluster* (Nader *et al.* 2002). JOPI merupakan satu kaedah yang baru dan sewaktu penulisan ini dibuat, pengkaji yang paling aktif menulis untuk JOPI adalah Jameela Al-Jaroodi (Al-Jaroodi *et al*. 2001, 2002, 2003a, 2003b) bersama-sama dengan kumpulan kajiannya. Ini kerana kebanyakan bahan terbitan berkaitan JOPI tidak banyak dan kebanyakannya adalah daripada Al-Jaroodi ataupun merujuk kepada laporan yang pernah dibuat olehnya. Oleh itu perbincangan tentang JOPI pada seksyen ini akan lebih tertumpu kepada hasil ujikaji yang pernah dibuat oleh Jaroodi.

Menurut Al-Jaroodi *et al.* (2001; 2002), JOPI menyediakan pengguna dengan antara muka yang sama dengan piawaian MPI dan juga MPIJAVA. Walaubagaimanapun, JOPI mengeksploitasikan sifat berorientasikan objek, Java untuk memudahkan penulisan program selari. JOPI dibina menggunakan kelas-kelas *socket programming* di dalam Java. Walaupun ianya mengkomplekskan proses pembangunan, ia menawarkan kawalan penuh ke atas sistem dan fleksibel untuk pengubahsuaian rekabentuk yang sedia untuk penambahbaikan dan pengoptimuman prestasi. Sebagai tambahan, sistem akan bertambah laju dan lebih efisyen.

JOPI juga menyediakan antara muka seperti MPI yang boleh digunakan untuk menukar objek diantara proses (Al-Jaroodi *et al.* 2001). Kelebihan pendekatan ini adalah penggunaan objek dalam penukaran maklumat, yang memudahkan penukaran struktur kompleks dan membolehkan pengaturcara program membezakan antara masalah yang cuba diselesaikan dengan masalah dalam menselarikan proses aplikasi tersebut (Al-Jaroodi *et al.* 2001). Sebagai contoh, pengaturcara sistem boleh menulis kelas yang mengandungi kaedah untuk menyelesaikan masalah tertentu bersama-sama dengan aturcara yang mengandungi kaedah untuk membahagikan masalah tersebut kepada masalah-masalah kecil dan menggabungkan penyelesaian bagi masalah-masalah tersebut sebagai satu penyelesaian terakhir (Al-Jaroodi *et al.* 2001). Pada kelas yang berlainan pula, saiz kepada masalah yang telah dipecah-pecahkan kepada masalah-masalah kecil, jumlah proses yang akan digunakan dan bagaimana setiap proses akan berhubung antara satu sama lain tadi boleh ditetapkan.

JOPI juga menyediakan kemudahan pengaturcaraan selari dalam Java untuk memanfaatkan sumber-sumber yang ada dalam *cluster* dan juga pada sistem rangkaian. Persekitaran masa larian bagi JOPI juga adalah efisyen.

Ciri-ciri utama bagi JOPI adalah:

1) JOPI sesuai untuk *cluster* dan juga pelbagai sistem teragih yang lain.
2) JOPI menggunakan paradigma pengaturcaraan berorientasikan objek untuk pengaturcaraan selari yang memudahkan proses pembangunan.
3) JOPI amat sesuai untuk aplikasi yang besar dan aplikasi-aplikasi yang mempunyai ratio komunikasi-komputasi yang rendah.

2.3 Persamaan di antara JOPI dan MPIJAVA

Secara umumnya *object passin*g sama seperti *message passing* iaitu merupakan model bagi pengaturcaraan selari yang mempunyai kriteria-kriteria seperti berikut:

1) *Multithreading*: Mempunyai berbilang proses yang mempunyai struktur kawalan sendiri bagi setiap pemproses dan boleh melarikan kod aturcara yang berbeza bagi setiap pemproses.
2) *Asynchronous parallelism*: Setiap proses memproses secara berasingan dan memerlukan penyelarasan.
3) Ruang alamat yang berasingan: Setiap proses mempunyai ruang alamatnya yang tersendiri dan bertukar-tukar informasi menerusi fungsi *message-passing*.
4) Interaksi yang jelas: Pengguna harus menyelesaikan sendiri semua isu interaksi seperti komunikasi, penyelarasan dan pengumpulan.
5) Pembahagian yang jelas: Beban kerja dan data dibahagikan kepada proses-proses oleh pengguna.

2.4 Ujian Perbandingan

Ujian perbandingan adalah ujian yang digunakan untuk membandingkan JOPI dan juga MPIJAVA. Kajian ini menekankan empat ujian perbandingan iaitu ujian komunikasi ping, ujian komunikasi ping-pong, ujian *overhead* perisian serta ujian menggunakan satu aplikasi program selari iaitu pendaraban matriks.

2.4.1 Prestasi komunikasi

Komunikasi adalah kaedah bagaimana sekumpulan proses atau nod berinteraksi atau berhubung antara satu sama lain. Komunikasi adalah penting sebagai salah satu elemen yang boleh digunakan sebagai pengukur untuk membuat perbandingan dari segi prestasi komunikasi bagi seni bina komputer yang berbeza ataupun perisian yang berbeza. Menurut Nupairoj dan Lionel, (1995), aktiviti-aktiviti yang melibatkan komunikasi seperti pertukaran data antara proses-proses boleh menyebabkan program terhalang daripada mencapai kelajuan yang ideal. Ini menunjukkan bahawa komunikasi amat penting dalam mempengaruhi masa keseluruhan bagi

program yang melibatkan rangkaian. Oleh yang demikian, di dalam kajian ini, dua ujian komunikasi telah digunakan untuk menguji masa komunikasi bagi MPIJAVA dan JOPI. Ujian tersebut adalah ujian komunikasi ping dan ujian komunikasi ping-pong. Menurut Nupairoj dan Lionel, (1995), untuk membandingkan antara dua sistem komunikasi, beberapa metrik adalah diperlukan. Manakala untuk membandingkan di antara dua perpustakaan komunikasi (JOPI dan MPIJAVA), hanya dua metrik sahaja yang sesuai digunakan iaitu *communication latency* dan *bandwidth*.

2.4.1.1 Metrik Prestasi

Seperti yang telah dijelaskan, metrik prestasi yang sesuai digunakan untuk membandingkan JOPI dan MPIJAVA adalah *communication latency* dan juga *bandwidth*. Kedua-dua metrik ini boleh diperolehi dengan cara membuat pengujian menerusi ujian ping dan ujian ping pong.

2.4.1.1.1 *Communication Latency*

Nupairoj dan Lionel, (1995) telah menyatakan bahawa *communication latency* adalah masa yang diperlukan oleh sesuatu proses untuk menghantar mesej, menerima mesej atau kedua-duanya. *Communication latency* sangat bergantung kepada sifat sesuatu program. Sebagai contoh, apabila satu proses menghantar mesej kepada satu proses yang lain, jika ia perlu menunggu jawapan daripada proses tersebut (ping pong), *overhead* rangkaian dan *overhead* perisian akan termasuk di dalam *communication latency*. *Latency* ini dipanggil *end-to-end delay*. Walaubagaimanapun, jika proses tersebut tidak perlu menunggu (ping), *communication latency* hanyalah pada *overhead* perisian.

2.4.1.1.2 *Bandwidth*

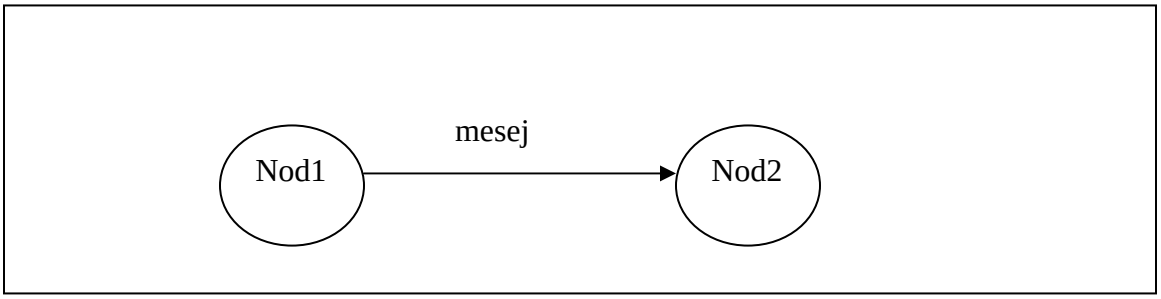
Bandwidth merupakan kadar sejauh mana rangkaian dapat menghantar data (Nupairoj dan Lionel, 1995). Ia digunakan dengan meluas oleh kerana mudah untuk diuji. Metrik ini digunakan apabila perbandingan di antara saiz mesej yang berbeza ingin dibuat. Ini kerana, menurut Ahmad (1997), saiz bagi data memberikan kesan yang besar terhadap sistem

komunikasi. Untuk mengira keberkesanan *bandwidth* daripada data bagi ping pong, Al-Jaroodi *et al*. (2001) telah menyarankan agar rumus seperti berikut digunakan:

$$\textit{Bandwidth}\ (\text{Mbps}) = (8*\text{Saiz mesej}/2^{20})(\text{Masa ping pong}/2) \quad (1)$$

2.4.2 Ujian Komunikasi Ping

Komunikasi ping dibuat untuk mengukur prestasi puncak komunikasi *point-to-point* pada rangkaian komputer *cluster* (Nupairoj dan Lionel, 1995). Selain itu, seperti yang telah diterangkan pada seksyen 2.4.1.1.1 dan 2.4.1.1.2, ping juga boleh digunakan untuk menguji *communication latency* dan juga *bandwidth* rangkaian. Ping berlaku apabila mesej dihantar daripada satu nod kepada nod yang lain. Penghantaran mesej adalah berlaku secara sehala. Nod penghantar terus menghantar mesej melainkan ia dihalang daripada berbuat demikian dan nod penerima akan terus menerima mesej (Nupairoj dan Lionel, 1995; Ishfaq, 1996). Model bagi komunikasi ping boleh dilihat pada Rajah 2.1 (Nupairoj dan Lionel, 1995).

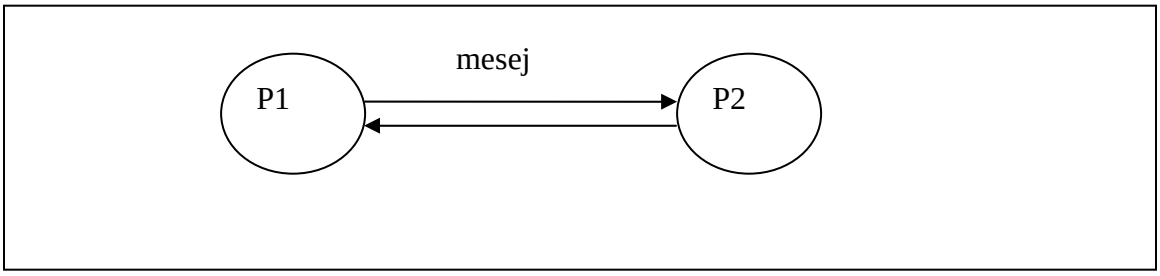


Rajah 2.1 : Model komunikasi ping

2.4.3 Ujian Komunikasi ping-pong

Seperti yang telah dijelaskan pada seksyen 2.4.1.1.1, komunikasi ping-pong pula diuji untuk mengukur *end-to-end delay* yang melibatkan kesan daripada protokol komunikasi (Nupairoj dan Lionel, 1995). Selain itu, ping pong juga boleh digunakan untuk menguji keberkesanan *bandwidth* seperti yang telah dijelaskan dalam seksyen 2.4.1.1.2. Ujian komunikasi ping-pong berfungsi dengan cara menghantar mesej dan menerimanya semula oleh dua nod atau proses (Nader *et al.* 2002). Tidak seperti ping, setiap proses atau nod akan menunggu giliran untuk menjadi penghantar, dan dalam satu masa hanya boleh ada satu penghantar sahaja. Rajah

2.2 menunjukkan model komunikasi ping-pong. Proses P1 menghantar mesej kepada P2 dan menunggu P2 membalas mesej tersebut. Apabila P2 menerima mesej, ia akan menghantar kembali kepada P1 (Nupairoj dan Lionel, 1995).



Rajah 2.2 : Model komunikasi ping pong

2.4.4 Ujian *Overhead* Perisian

Overhead perisian merupakan salah satu elemen yang terdapat di dalam *communication latency* (Nupairoj dan Lionel, 1995). Oleh yang demikian ia juga memberikan kesan kepada prestasi komunikasi. Nupairoj dan Lionel juga menyatakan bahawa *overhed* perisian ini adalah tetap untuk setiap saiz data. Ini bermaksud untuk sebarang saiz data, *overhead* perisian tidak berubah, tetapi *communication latency* tetap berubah kerana wujudnya *overhead* pada rangkaian seperti yang telah dinyatakan dalam seksyen 2.4.1.1.1

Menurut Al-Jaroodi *et al*.(2003), untuk menguji *overhead*, program Java dilaksanakan secara bersendirian (tanpa JOPI) dan kemudian dibandingkan dengan program Java yang dilarikan bersama-sama JOPI. Purata masa bagi kedua-dua program tersebut diambil dan dibandingkan. Teori ini boleh digunakan untuk menguji masa permulaan bagi MPIJAVA. Ini kerana MPIJAVA juga menggunakan perpustakaan yang berasingan dengan perpustakaan Java yang sedia ada. Di dalam kajian ini, program pengiraan Trapezium telah diimplementasikan menggunakan perpustakaan Java sahaja, perpustakaan JOPI dan juga perpustakaan MPIJAVA.

2.4.5 Aplikasi Program Selari

Untuk menguji keberkesanan sesuatu perpustakaan selari, program selari perlu dibangunkan dan diuji. Program selari yang sering digunakan untuk pengujian ini adalah pendaraban matriks. Menurut Guster *et al*.(2003), untuk menguji keefisyenan komunikasi pemproses-ke-pemproses dalam konteks masa pemprosesan, algoritma pendaraban matriks *A x B* boleh digunakan. Masa pemprosesan ini merangkumi proses penghantaran, pengiraan dan penerimaan data oleh pemproses. Pemilihan teori ini juga adalah kerana ianya menpunyai ciri-ciri tidak bergantungan antara setiap langkah dalam proses penyeleaian masalah. Sifat ini membolehkan ianya dipecah-pecahkan kepada masalah-masalah kecil yang seterusnya boleh dilarikan pada persekitaran sistem selari. (Bhandarkar *et al*. 2001; Parent *et al.* 2002)

Di dalam program ini, nod *master* akan mengawal kerja-kerja bagi nod-nod *slave*. Setiap kerja bagi *slave* akan diagihkan ke setiap nod *slave* oleh nod *master*. Program untuk nod *master* akan menghantar semua matriks B kepada setiap *slave* dan kemudian memecahkan matriks A dan menghantar pecahan-pecahan matriks tersebut kepada *slave*.

2.4.5.1 Pendaraban Matriks (Ahmad *et al.* 2003)

Rumus pendaraban matriks telah digunakan secara meluas untuk menguji masa pemprosesan bagi sesuatu pustaka pengkomputeran selari (Al-Jaroodi *et al.* 2003b; Nader *et al.* 2002; Al-Jaroodi *et al.* 2001). Sebagai contoh, Guster *et al.* (2003), telah membuat perbandingan antara MPI dan *Parallel Virtual Machine* (PVM) dari segi masa pemprosesan dengan mengimplimentasikan rumus pendaraban matriks. Di dalam kajian ini, pendaraban matriks turut digunakan sebagai satu kaedah untuk menguji masa pemprosesan bagi MPIJAVA dan JOPI. Matrik C yang terhasil daripada pendaraban matrik A dan B didefinasikan sebagai

$$C_{ik} = a_{ij}\, b_{jk} \qquad (2)$$

Di mana nilai j, i dan k adalah lebih besar daripada 0. Nilai C boleh diperolehi daripada penyelesaian berikut:

$$\begin{bmatrix} C_{11}\ C_{12}.....C_{1p} \\ C_{21}\ C_{22}.....C_{2p} \\ C_{n1}\ C_{n2}.....C_{np} \end{bmatrix} = \begin{bmatrix} a_{11}\ a_{12}.....a_{1m} \\ a_{21}\ a_{22}.....a_{2m} \\ a_{n1}\ a_{n2}.....a_{nm} \end{bmatrix} \begin{bmatrix} b_{11}\ b_{12}.....b_{1p} \\ b_{21}\ b_{22}.....b_{2p} \\ b_{m1}\ b_{m2}.....b_{mp} \end{bmatrix} \quad (3)$$

Di mana,

$C_{11} = a_{11}\, b_{11} + a_{12}\, b_{21} + ... + a_{1m}\, b_{m1}$
$C_{12} = a_{11}\, b_{12} + a_{12}\, b_{22} + ... + a_{1m}\, b_{m2}$
$C_{1p} = a_{11}\, b_{1p} + a_{12}\, b_{2p} + ... + a_{1m}\, b_{mp}$
$C_{21} = a_{21}\, b_{11} + a_{22}\, b_{21} + ... + a_{2m}\, b_{m1}$
$C_{22} = a_{21}\, b_{12} + a_{22}\, b_{22} + ... + a_{2m}\, b_{m2}$
$C_{2p} = a_{21}\, b_{1p} + a_{22}\, b_{2p} + ... + a_{2m}\, b_{mp}$
$C_{n1} = a_{n1}\, b_{11} + a_{n2}\, b_{21} + ... + a_{nm}\, b_{m1}$
$C_{12} = a_{n1}\, b_{12} + a_{n2}\, b_{22} + ... + a_{nm}\, b_{m2}$
$C_{np} = a_{n1}\, b_{1p} + a_{n2}\, b_{2p} + ... + a_{nm}\, b_{mp}$

2.4.5.1.1 Kaedah Penjadualan Data Selari

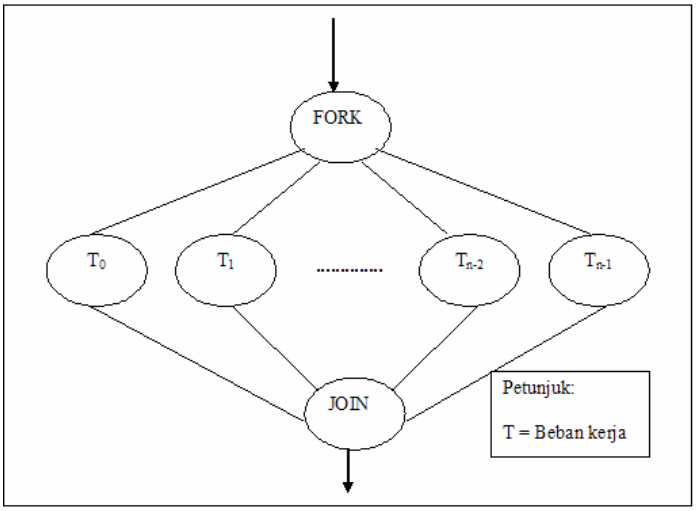
Seksyen 2.3 telah menekankan tentang kriteria-kriteria yang ada pada JOPI dan MPIJAVA. Salah satunya adalah pembahagian beban kerja yang jelas kepada proses-proses yang terlibat oleh pengguna. Oleh kerana ia merupakan satu kriteria yang penting, maka seksyen ini akan membincangkan penjadualan data selari yang berfungsi untuk mengagihkan beban kerja pada program pendaraban matriks.

Di dalam persekitaran selari, sesuatu program komputer boleh digambarkan sebagai sekumpulan kerja atau tugas yang boleh dilarikan secara selari. Matlamat penjadualan adalah untuk menentukan skop bagi sesuatu tugas itu bagi mengoptimumkan prestasi sesuatu sistem selari (El-Rewini *et al*. 1994). Terdapat banyak kaedah penjadualan data selari. Walaubagaimanapun, di sini hanya satu kaedah yang akan diketengahkan iaitu *Fork-and-Join*.

2.4.5.1.1.1 Fork-and Join

Kaedah ini juga dikenali sebagai penjadualan *Piece-Wise* (El-Rewini *et al*. 1994). Melalui kaedah ini kerja dibahagikan kepada pengiraan-pengiraan kecil. Kaedah ini sesuai untuk jenis masalah dengan struktur penyelesaian yang berulang melalui fasa komunikasi dan fasa pengiraan. Struktur bagi penjadualan ini boleh dilihat pada Rajah 2.3. Rajah ini menunjukkan satu fasa pada *fork-and-join*. Operasi *fork* membahagikan pengiraan kepada n bilangan kerja iaitu T_0,

T_1, ..., T_{n-1}. Setelah pengiraan selesai, kumpulan kerja ini akan dikumpulkan melalui operasi *join* (Dandamudi, 2003).



Rajah 2.3 : Struktur kerja *fork-and-join* (Dandamudi, 2003).

3.0 Hasil Eksperimen

Bab ini akan menekankan tentang hasil yang diperolehi daripada eksperimen-eksperimen yang telah dijalankan untuk menguji prestasi JOPI dan MPIJAVA. Terdapat empat ekperimen yang telah diuji dengan menggunakan komputer *cluster* 13 nod yang setiap satunya menggunakan pemproses Intel Pentium III 1400MHz. Dua daripada eksperimen tersebut iaitu pengujian ping dan ping pong dibuat bertujuan untuk mengukur preastasi komunikasi JOPI dan MPIJAVA. Untuk menguji *overhead* perisian pula, eksperimen yang menggunakan Petua Trapezium telah dijalankan. Eksperimen terakhir iaitu aplikasi pendaraban matriks telah dibuat untuk menguji masa pemprosesan bagi JOPI dan MPIJAVA.

3.1 Prestasi Komunikasi

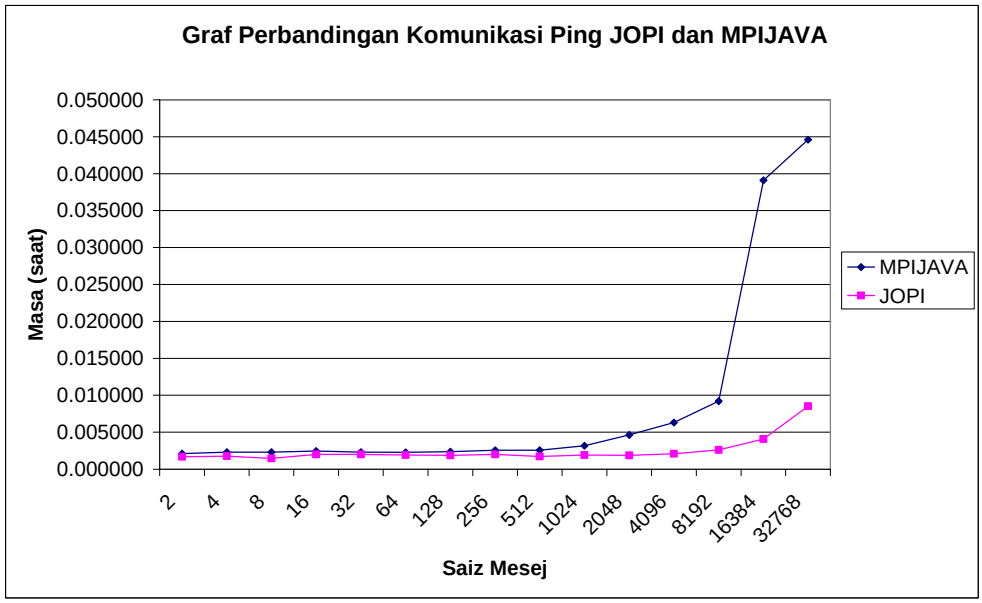
Seperti yang telah ditekankan di dalam bab 2, komunikasi amat penting dalam mempengaruhi masa keseluruhan program selari. Oleh yang demikian, untuk menilai JOPI dan MPIJAVA, maka pengujian terhadap prestasi komunikasi perlu dibuat. Untuk menguji prestasi komunikasi ini, dua pengujian telah digunakan iaitu ping dan ping pong.

3.1.1 Hasil Eksperimen Ping

Pengujian bagi ping telah dibuat melibatkan satu nod penghantar dan satu nod penerima. Pergerakan data adalah berlaku dalam satu arah di mana nod penghantar hanya menghantar data manakala nod penerima hanya menerima data daripada nod penghantar. Saiz data yang dihantar dipertingkatkan untuk setiap pengujian. Ini kerana menurut Nupairoj dan Lional (1995), saiz mesej memberikan kesan major kepada *communication latency*. Rajah 3.1 menunjukkan hasil daripada eksperimen bagi ping yang telah dibuat ke atas JOPI dan MPIJAVA.

Jadual 3.1 : Hasil pengujian ping JOPI dan MPIJAVA

Saiz Mesej (bait)	**MPIJAVA (Saat)**	**JOPI (Saat)**
2	0.002114	0.001682
4	0.002298	0.001773
8	0.002303	0.001455
16	0.002463	0.002000
32	0.002296	0.002000
64	0.002284	0.001909
128	0.002377	0.001870
256	0.002562	0.002000
512	0.002566	0.001727
1024	0.003182	0.001909
2048	0.004626	0.001864
4096	0.006309	0.002087
8192	0.009212	0.002609
16384	0.039103	0.004087
32768	0.044600	0.008522



Rajah 3.1 Graf perbandingan komunikasi ping JOPI dan MPIJAVA.

Hasil daripada eksperimen ini dipaparkan di dalam bentuk graf supaya prestasi dan perbandingan di antara JOPI dan MPIJAVA dapat dilihat dengan jelas. Daripada graf pada Rajah 3.1, dapat diperhatikan masa yang diperlukan bagi JOPI dan MPIJAVA tidak banyak berbaza jika data yang dihantar bersaiz 2 hingga 512 bait. Sebagai contoh prestasi MPIJAVA kurang daripada JOPI sebanyak 49 peratus bagi data bersaiz 512. Namun begitu prestasi MPIJAVA mula menurun apabila saiz data dipertingkatkan dan kemuncak kepada penurunan prestasi ini adalah ketika data bersaiz 16384 bait digunakan iaitu sebanyak 857 peratus lebih lambat berbanding JOPI. Berlainan dengan hasil pengujian menggunakan JOPI, prestasi JOPI terus stabil sehingga saiz data bersaiz 8192. Penurunan prestasi tidak begitu ketara antara setiap saiz data yang dipertingkatkan, sebagai contoh di antara data bersaiz 16384 bait dan 32868 bait, penurunan prestasi adalah sebanyak 109 kali ganda sahaja.

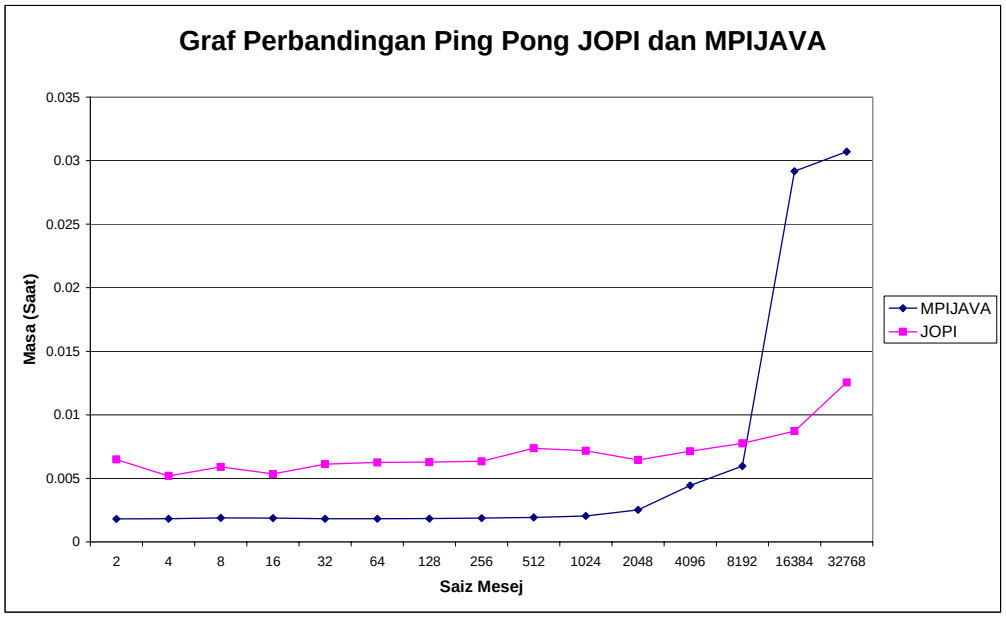
Secara keseluruhannya, daripada hasil yang diperolehi didapati bahawa prestasi JOPI dari segi *communication latency* adalah lebih baik berbanding MPIJAVA.

3.1.2 Hasil Eksperimen Ping pong

Pengujian bagi ping pong juga telah dibuat melibatkan satu nod penghantar dan satu nod penerima. Namun begitu pergerakan data adalah berlaku dalam dua arah di mana nod A bukan hanya menghantar data kepada nod B, tetapi turut menerima data daripada nod tersebut, manakala nod B bukan hanya menerima data daripada nod A tetapi turut bertindak sebagai penghantar data kepada nod A. Saiz data yang dihantar dipertingkatkan untuk setiap pengujian. Seperti yang telah diterangkan di dalam bab 2, eksperimen ping pong di dalam kajian ini lebih memfokuskan kepada pengujian untuk mengukur kadar *bandwidth* bagi JOPI dan MPIJAVA. Rajah 3.2 menunjukkan graf hasil daripada pengujian ping pong yang telah dibuat ke atas JOPI dan MPIJAVA.

Jadual 3.2 : Hasil pengujian ping pong JOPI dan MPIJAVA

Saiz Mesej (bait)	**MPIJAVA (Saat)**	**JOPI (Saat)**	***Bandwidth* MPIJAVA (Mb/s)**	***Bandwidth* JOPI (Mb/s)**
2	0.001818	0.006493	0.016789	0.0047
4	0.00182	0.005189	0.033531	0.011762
8	0.00189	0.005908	0.064591	0.020661
16	0.001882	0.005352	0.129718	0.045613
32	0.001827	0.006119	0.267201	0.0798
64	0.001831	0.00626	0.533318	0.156
128	0.001838	0.00628	1.062617	0.311007
256	0.001875	0.00634	2.083816	0.616128
512	0.001929	0.00737	4.050442	1.060041
1024	0.002048	0.00717	7.629965	2.179219
2048	0.00253	0.00645	12.35209	4.844961
4096	0.00444	0.00714	14.07561	8.753501
8192	0.005963	0.00775	20.96158	16.12903
16384	0.029164	0.00872	8.57212	28.66972
32768	0.030702	0.01255	16.28553	39.84064

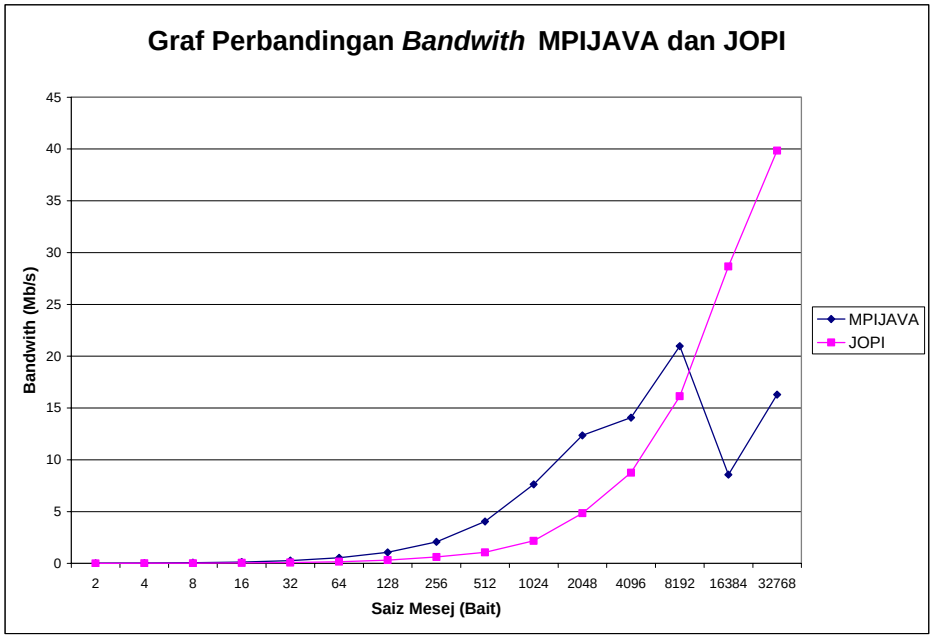


Rajah 3.2 Graf perbandingan ping pong JOPI dan MPIJAVA

Daripada graf yang diperolehi, dapat dilihat prestasi bagi JOPI adalah stabil walaupun pada awalnya prestasi JOPI lebih rendah jika dibandingkan dengan MPIJAVA. Ini dapat dilihat dengan penggunaan data bermula dengan saiz 2 bait sehingga ke 8192 bait prestasi MPIJAVA adalah lebih baik daripada JOPI iaitu kira-kira 250 peratus lebih rendah. Namun begitu apabila saiz data semakin meningkat, prestasi MPIJAVA semakin menurun. Sebagai contoh apabila saiz data yang digunakan bersaiz 16384, prestasi MPIJAVA menurun dan menjadi rendah sebanyak 234 peratus berbanding JOPI. Jurang perbezaan yang ketara ini berkekalan selari dengan peningkatan saiz data.

Secara keseluruhannya, prestasi MPIJAVA dari sudut *end-to-end delay* adalah lebih baik berbanding JOPI jika data yang digunakan kurang atau sama dengan 8192 bait atau dengan kata lain saiz data yang digunakan adalah kecil, tetapi prestasi JOPI adalah lebih baik jika saiz data yang digunakan adalah besar iaitu lebih daripada 8192 bait. Perkara ini amat berkait rapat dengan pengaruh daripada *overhead* perisian. Ini kerana, dengan saiz data yang kecil, *overhead* perisian bagi JOPI membuatkan masa keseluruhan proses menjadi tinggi apabila saiz data yang digunakan adalah kecil. Namun begitu, pengaruh *overhead* JOPI akan menjadi semakin kurang nyata selari dengan peningkatan saiz data.

Melalui pengujian ping pong juga, kadar *bandwidth* bagi JOPI dan MPIJAVA dapat diukur. Seperti yang telah dijelaskan di dalam bab 2, nilai *bandwidth* boleh diperolehi daripada data masa komunikasi ping pong ini dengan menggunakan rumus seperti yang telah dinyatakan di dalam seksyen 2.4.1.1.2. Rajah 3.3 menunjukkan graf perbandingan kadar *bandwidth* bagi JOPI dan MPIJAVA.



Rajah 3.3 Graf perbandingan *bandwidth* JOPI dan MPIJAVA

Berdasarkan graf pada Rajah 3.3, dapat disimpulkan kadar *bandwidth* bagi MPIJAVA adalah lebih baik berbanding JOPI jika data bersaiz kurang atau sama dengan 8192 bait digunakan. Sebagai contoh, apabila data bersaiz 8192 bait digunakan, kadar *bandwidth* bagi MPIJAVA adalah 30 peratus lebih baik berbanding JOPI. Ini bermakna pada tahap tersebut, komunikasi bagi MPIJAVA dalam konteks transaksi data adalah lebih efektif berbanding JOPI. Namun begitu, apabila saiz data semakin meningkat iaitu melebihi 8192 bait, penggunaan JOPI adalah lebih efektif untuk melakukan transaksi data. Sebagai contoh, apabila data bersaiz 32768 digunakan, kadar *bandwidth* bagi JOPI mengatasi MPIJAVA sebanyak 145 peratus.

Seperti yang telah dijelaskan dalam seksyen 2.4.1.1.2, *bandwidth* merupakan kadar sejauh mana rangkaian dapat menghantar data. Ia amat berkait rapat dengan saiz data seperti yang telah dinyatakan oleh Ishfaq, (1997) iaitu saiz data memberikan kesan yang besar terhadap sistem

komunikasi. Oleh itu, daripada graf yang diperolehi, dapat diperhatikan bahawa apabila saiz data adalah kecil, kadar *bandwidth* bagi MPIJAVA adalah tinggi daripada JOPI. Ini menunjukkan bahawa MPIJAVA lebih efektif untuk digunakan berbanding JOPI. Walaubagaimanpun, apabila saiz data semakin dipertingkatkan, kadar *bandwidth* bagi JOPI akan mengatasi *bandwidth* bagi MPIJAVA. Ini membuktikan bahawa MPIJAVA kurang efektif untuk digunakan jika dibandingkan dengan JOPI apabila saiz data adalah besar.

Hasil daripada ujian ini amat penting untuk menentukan pemilihan di antara MPIJAVA dan JOPI berdasarkan kepada saiz data yang hendak dihantar di dalam sesuatu sistem selari.

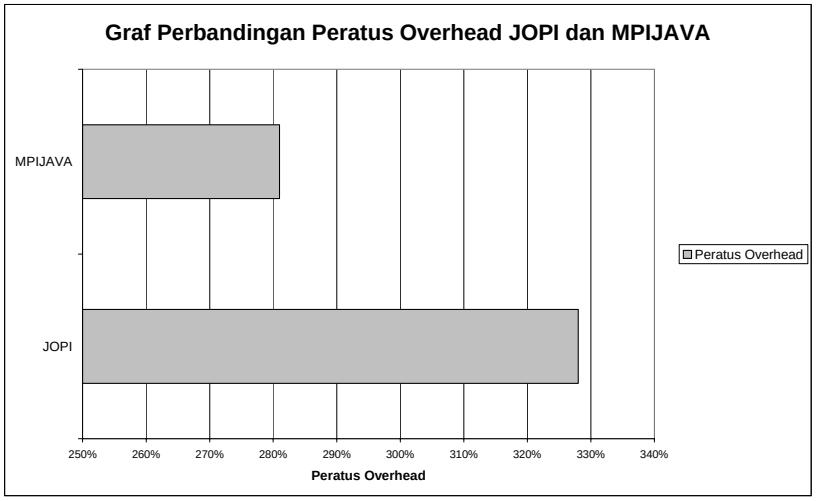
3.2 Hasil Eksperimen *Overhead* Perisian

Overhead perisian merupakan masa yang diperlukan oleh JOPI dan MPIJAVA untuk memulakan sesuatu program selari. Seperti yang telah dijelaskan di dalam bab 2, *overhead* perisian adalah tetap untuk sebarang saiz mesej di dalam transaksi data. Selain daripada itu, turut dinyatakan bahawa *overhead* perisian juga memberikan kesan terhadap *communication latency*. Justeru itu pengujian ini adalah penting untuk membandingkan di antara JOPI dan MPIJAVA, yang manakah yang mempunyai peratus *overhead* yang lebih rendah.

Jadual 3.3 : Hasil pengujian *overhead* JOPI dan MPIJAVA

Platform	**Purata Masa(milisaat)**	**Peratus *Overhead***
Java	13.46	0%
JOPI	57.63	328%
MPIJAVA	51.4	281%

Untuk memperolehi peratus *overhead* perisian, masa larian bagi JOPI atau MPIJAVA ditolakkan dengan masa larian bagi Java. Nilai yang diperolehi kemudiannya dibahagikan dengan masa larian Java dan didarabkan dengan 100 untuk mendapatkan peratus *overhead* perisian.



Rajah 3.4 Graf perbandingan *overhead* perisian JOPI dan MPIJAVA.

Rajah 3.4 menunjukkan graf perbandingan *overhead* perisian antara JOPI dan MPIJAVA. Daripada graf tersebut, dapat dilihat bahawa JOPI mempunyai *overhead* perisian yang tinggi iaitu sebanyak 328 peratus, berbanding MPIJAVA sebanyak 280 peratus. Dari segi teori, peratusan ini menunjukkan bahawa prestasi JOPI sepatutnya lebih rendah daripada MPIJAVA. Namun begitu, daripada segi praktikalnya, penggunaan JOPI lebih praktikal jika saiz data yang digunakan untuk transaksi adalah besar seperti yang telah dibuktikan dan dijelaskan dalam seksyen 3.2.2. Ini kerana peningkatan saiz data akan memperbaiki prestasi kerana kesan daripada *overhead* perisian akan dikurangkan apabila masa transaksi meningkat (Nupairoj dan Lionel, 1995; Al-Jaroodi *et al*. 2001).

3.3 Pengujian Aplikasi Program Selari

Untuk memastikan di antara JOPI dan MPIJAVA, perpustakaan mana yang paling sesuai digunakan untuk aplikasi masalah sebenar, maka pengujian terhadap program pendaraban matriks telah dilakukan. Daripada pengujian ini, nilai bagi masa keseluruhan pemprosesan bermula dari proses penghantaran data, pengiraan dan penerimaan semula telah diperolehi. Data-data ini telah direkodkan seperti yang terdapat pada Jadual 3.4, Jadual 3.5, Jadual 3.6, Jadual 3.7 dan Jadual 3.8.

Jadual 3.4 : Hasil pengujian pendaraban matriks (*128x128*) JOPI dan MPIJAVA

Bil. Pemproses	MPIJAVA	JOPI
2	0.114984	0.109400
4	0.128431	0.110000
6	0.183375	0.121800
8	0.266635	0.136500
10	0.365256	0.153000
12	0.486534	0.168100

Jadual 3.5 : Hasil pengujian pendaraban matriks (256*x256*) JOPI dan MPIJAVA

Bilangan Pemproses	MPIJAVA (Saat)	JOPI (Saat)
2	0.490459	0.488200
4	0.342350	0.293091
6	0.438380	0.301000
8	0.568932	0.334900
10	0.761872	0.372100
12	0.800541	0.411300

Jadual 3.6 : Hasil pengujian pendaraban matriks (*512x512*) JOPI dan MPIJAVA

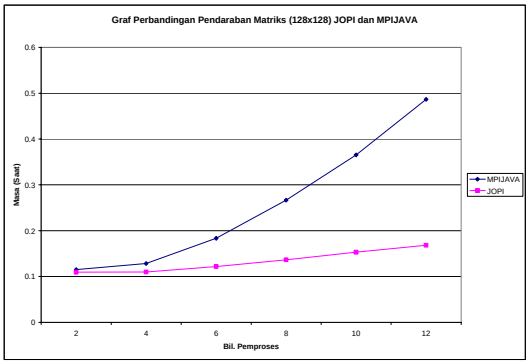
Bilangan Pemproses	MPIJAVA (Saat)	JOPI (Saat)
2	3.350914	2.669700
4	1.524666	1.169000
6	1.551060	1.064950
8	1.908501	1.122571
10	2.262153	1.331200
12	2.746955	1.417800

Jadual 3.7 : Hasil pengujian pendaraban matriks (*1024x1024*) JOPI dan MPIJAVA

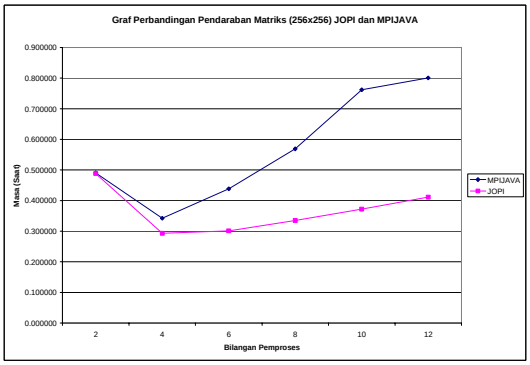
Bilangan Pemproses	MPIJAVA (Saat)	JOPI (Saat)
2	25.440900	11.115000
4	10.031000	6.589000
6	7.629700	6.323000
8	8.008000	7.059000
10	9.238100	8.290000
12	10.334000	9.465000

Jadual 3.8 : Hasil pengujian pendaraban matriks (*1130x1130*) JOPI dan MPIJAVA

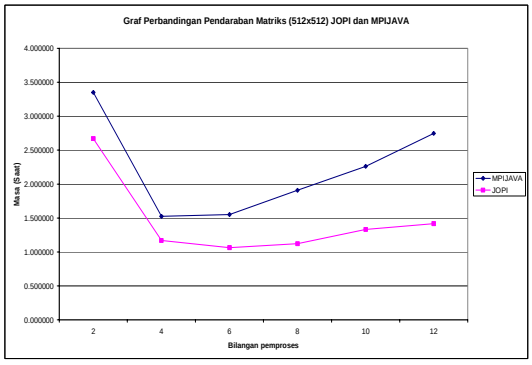
Bilangan Pemproses	MPIJAVA	JOPI
2	33.999000	28.743000
4	13.201980	8.985000
6	9.794660	8.029000
8	9.607440	8.768000
10	11.569000	10.252000
12	12.754990	11.782000



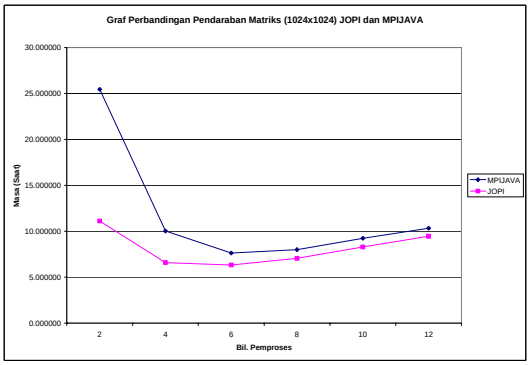
Rajah 3.5 Graf perbandingan masa pemprosesan pendaraban matriks (*128x128*) JOPI dan MPIJAVA.



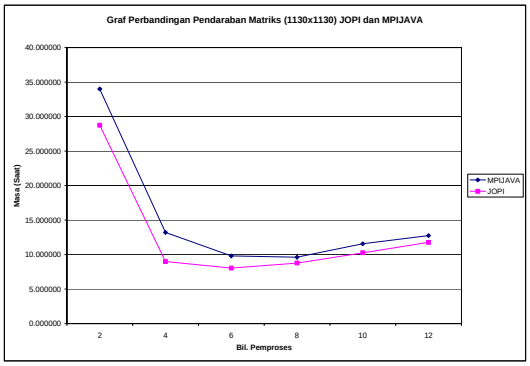
Rajah 3.6 Graf perbandingan masa pemprosesan pendaraban matriks (*256x256*) JOPI dan MPIJAVA.



Rajah 3.7 Graf perbandingan pemprosesan masa pendaraban matriks (*512x512*) JOPI dan MPIJAVA.



Rajah 3.8 Graf perbandingan masa pemprosesan pendaraban matriks (*1024x1024*) JOPI dan MPIJAVA.



Rajah 3.9 Graf perbandingan masa pemprosesan pendaraban matriks (*1130x1130*) JOPI dan MPIJAVA.

Data-data ini telah dipersembahkan di dalam bentuk graf seperti pada Rajah 3.5, Rajah 3.6, Rajah 3.7, Rajah 3.8 dan Rajah 3.9. Daripada corak graf yang terhasil, dapat diperhatikan bahawa apabila saiz matriks yang digunakan dipertingkatkan, prestasi MPIJAVA akan semakin menghampiri JOPI atau dengan kata lain jurang perbezaan prestasi bagi MPIJAVA dan JOPI akan semakin berkurangan. Namun begitu ia tetap tidak dapat mengatasi pretasi bagi JOPI. Sebagai contoh, apabila matriks *128x128* digunakan, peratus prestasi JOPI adalah 189 peratus lebih baik berbanding MPIJAVA untuk proses larian pada 12 buah nod. Apabila saiz matriks dipertingkatkan sehingga *1130x1130*, prestasi JOPI hanya 8 peratus lebih baik berbanding MPIJAVA. Keadaan ini membuktikan bahawa hasil yang diperolehi daripada pengujian amat berkait rapat dengan platform yang digunakan. Komputer *cluster* yang digunakan untuk kajian ini adalah amat berkuasa tinggi. Oleh itu, ia hanya efektif memproses sekiranya data yang sangat besar digunakan. Rajah 3.9 membuktikan bahawa apabila saiz data yang sangat besar digunakan, masa pemprosesan yang diperolehi adalah lebih stabil dengan pertambahan bilangan nod. Walaubagaimanapun, faktor ini hanya banyak mempengaruhi MPIJAVA. Ini dapat dilihat dari segi corak graf yang terhasil daripada pengujian pendaraban matriks bermula dari matriks *128x128* sehingga matriks *1130x1130*. Dapat diperhatikan bahawa prestasi MPIJAVA semakin bertambah baik untuk setiap pertambahan nod hanya jika saiz matriks yang digunakan ditambah untuk setiap pengujian.Perkara ini amat berbeza jika dibandingkan dengan JOPI. Daripada corak graf yang terhasil, dapat diperhatikan bahawa prestasi JOPI kurang dipengaruhi oleh saiz data.

4.0 Kesimpulan

Secara keseluruhannya kajian ini dibangunkan untuk mengkaji penggunaan MPIJAVA dan JOPI pada persekitaran pengkomputeran selari. Secara umumnya MPIJAVA dan JOPI dipilih untuk kajian ini kerana kedua-duanya merupakan pustaka yang menyokong teknik berorientasikan objek yang semakin popular dikalangan pembangun sistem pada masa kini. Namun begitu, untuk membuktikan di antara MPIJAVA dan JOPI, pustaka manakah yang terbaik dari segi prestasi, telah membawa kepada kajian ini dibuat.

Bagi merealisasikan kajian ini, aturcara yang melibatkan pengiraan dan komunikasi telah dibangunkan pada persekitaran pengkomputeran selari. Masalah matematik yang diambil sebagai rujukan hendaklah sesuai dengan persekitaran pemprosesan selari supaya setiap proses yang hendak dipecahkan tidak bergantung antara satu sama lain. Perkara ini adalah amat penting kerana masalah matematik yang tidak menepati ciri-ciri ini tidak akan dapat dilarikan dengan berkesan pada persekitaran pengkomputeran selari.

Daripada hasil pengujian yang telah dibuat, didapati bahawa JOPI mempunyai prestasi komunikasi yang lebih baik jika saiz data yang digunakan adalah besar. Prestasi bagi MPIJAVA pula amat baik sekiranya saiz data yang digunakan sewaktu proses transaksi adalah kecil. Namun begitu, prestasi bagi JOPI adalah lebih stabil merujuk kepada graf yang dihasilkan daripada pengujian ping dan ping pong. Berbeza dengan JOPI, prestasi MPIJAVA akan menurun secara mendadak selari dengan peningkatan saiz data. Perkara ini juga boleh dikaitkan dengan kadar *bandwidth* bagi JOPI dan MPIJAVA. Daripada hasil ujikaji, dapat diperhatikan bahawa JOPI lebih efektif daripada MPIJAVA sekiranya saiz data sewaktu transaksi adalah besar dan ianya stabil apabila saiz data semakin dipertingkatkan. Keadaan ini berbeza dengan MPIJAVA kerana ianya efektif untuk digunakan sekiranya saiz data adalah kecil. Namun begitu MPIJAVA menjadi tidak efektif apabila saiz data dipertingkatkan dan daripada data yang diperolehi didapati bahawa peningkatan ini menjadikan MPIJAVA tidak stabil. Prestasi MPIJAVA yang lebih baik daripada JOPI apabila saiz data sewaktu transaksi adalah kecil berkait rapat dengan *overhead* perisian yang dikaji. Seperti yang boleh diperhatikan pada hasil ujikaji yang telah dibuat, JOPI mempunyai *overhead* perisian yang tinggi berbanding MPIJAVA. Oleh itu, apabila saiz data yang digunakan semasa transaksi adalah kecil, maka ia akan memberikan kesan buruk kepada masa keseluruhan bagi JOPI, tetapi adalah baik untuk MPIJAVA kerana ianya mempunyai *overhead* perisian yang

rendah. Namun begitu, apabila saiz data dipertingkatkan keadaan ini berubah kerana JOPI mempunyai kadar *bandwidth* yang tinggi. Perkara ini membuatkan kesan *overhead* perisian semakin tidak kelihatan selari dengan peningkatan saiz data. Pengujian terakhir iaitu pendaraban matriks membuktikan bahawa masa pemprosesan iaitu masa keseluruhan proses penghantaran data, pengiraan dan penerimaan data bagi JOPI adalah lebih baik berbanding MIJAVA.

Di dalam kajian ini, untuk mengkaji masa pemprosesan, pendaraban matriks dengan saiz maksimum *1130x1130* telah digunakan. Untuk kajian pada masa hadapan, saiz matriks yang lebih besar boleh digunakan. Selain daripada itu, penggunaan teori-teori matematik yang lain juga boleh digunakan supaya hasil yang diperolehi boleh dibandingkan dan lebih jitu.

RUJUKAN

Al Jaroodi and Nader Mohamed (2001). JOPI: Java Object-Passing Interface, User's guide. University of Nebraska-Lincoln

Al Jaroodi, Jameela and Nader Mohamed (2003a). An Object-Passing Model for Parallel Programming. 2003. *Proceedings of the 27th Annual International Computer Software and Application Conference(CoOMPSAC'03).* IEEE

Al Jaroodi, Jameela and Nader Mohamed (2003b). Middleware Infrastructre for Parallel and Distributed Programming Models in Heterogeneous Systems.

Al-Jaroodi, Jameela, Nader Mohamed, Hong Jiang, and Swanson D.(2001). Agent-Based Parallel Computing in Java Proof of Concept. Technical Report TR-UNL-CSE-2001-1004. Department of Computer Science and Engineering, University of Nebraska, Lincoln.

Al-Jaroodi, Jameela, Nader Mohamed, Hong Jiang and S. David. (2002). A Comparative Study of Parallel and Distributed Java Projects for Heterogeneous Systems. *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS.02).* 1530-2075/02 2002 IEEE

Baiardi, F., Guerri, D., Mori, P., Ricci, L. And Vaglini, L. (2004). MPI on a virtual shared memory. Parallel Computing Software technology, algorithm, architectures and applications. 2004 Elsevier B.V.

Baker, M., Carpenter, B., Fox, G., S. H. Ko, and S. Lim. (1999). mpiJava: An Object-Oriented Java interface to MPI, *International Workshop on Java for Parallel and Distributed Computing*, IPPS/SPDP 1999.

Baker, M., Carpenter, B., Fox, G., Sung Hoon Ko, Xinying Li (1998). mpiJava: A Java Interface to MPI. September 14, 1998

Behrens, J., Haan, O. and Kornblueh,L. (2004). OpenMP vs.MPI on a Shared Memory Multiprocessor. Parallel Computing Software technology, algorithm, architectures and applications. 2004 Elsevier B.V.

Bhandarkar, M., Kale, L.V., Eric de Sturler and Hoeflinger, J.(2001). Adaptive Load Balancing for MPI Programs. *Proceedings of the International Conference on Computational Science-Part II* Berlin Heidelberg : Lecture Notes In Computer Science.

Bischofberger W., Pomberger G. (1992). Prototyping-Oriented Software Development—Concepts and Tools; Springer Verlag, 1992.

Buonocore, A., Nobile, A.G., Ricciardi, L.M. (1987). A New Integral Equation for the Evaluation of First-Passage-Time Probability Densities**.** *Advances in Applied Probability*, Vol. 19, No. 4 (Dec., 1987).

Carpenter, Getov, V., Judd, G., Skjellum, A., and Fox, G.,(2000). MPJ: MPIlike message passing for Java. Concurrency: Practice and Experience, vol. 12, no. 11,2000.

Dagum, L. and Menon R. (1998). OpenMP: An Industry-Standard API for Shared Memory Programming. *IEEE Computational Science and Engineering*.

Dandamudi, S.P. (2003). *Hierarchical Scheduling in Parallel and Cluster Systems*. New York: Kluwer Academic/Plenum Publisher.

David, K., Lowenthal, Freeh, V. W. and Mille, D. W.(2002). Efficient Support for Two-Dimensional Data Distributions in Distributed Shared Memory Systems. *Proceedings of International Parallel and Distributed Processing Symposium*. Ft. Lauderdale, FL, April.

Decegama, A.L. (1989*). Parallel Processing Architecture and VLSI Hardware*. Englewood Cliffs, N. J.: Prentice-Hall.

Dhillon, I.S., and Modha, D.S.(1999). A data clustering algorithm on distributed memory machines. USA Workshop on Large-Scale Parallel KDD Systems August 15th, 1999, San Diego, CA, USA in conjunction with ACM SIGKDD International Conference on Knowledge Discovery and Data Mining

El-Rewini, Hesham. (1994). *Task Scheduling in Parallel and Distributed Systems*. Englewood Cliffs, N. J.: Prentice-Hall.

Gennart, B. and Herschi, R.D.(1999).Computer-Aided Synthesis of Parallel Image Processing Applications. *Proceedings Conf. Parallel and Distributed Methods for Image Processing III*, Vol-3817.

Gibbons, P. B. (1996). What good are shared-memory models? In *Proc.* 1996 *ICPP Workshop on Challenges for Parallel Processing*, August 1996.

Govett, M. W. (2002). The Parallel Pre-Processor: a Compiler for Shared and Distributed Memory Computers. Accepted for the 8th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS 2003).

Groove, D.A., and Coddington, P.D.(2004). Communication Benchmarking and Performance Modelling of MPI Programs on Cluster Computers. *Proceedings of the 18th International Parallel and Distributed Processing Symposium(IPDPS'04)*. IEEE.

Gropp, W., Lusk, E., Doss, N. and Skjellum, A.(1996). A high- performance, portable implementation of the MPI message passing interface standard. Parallel Computing, 22(6):789–828, Sept. 1996.

Guster, D., Al-Hamamah, Abdullah, Safonov, P., and Bachman, E (2003). Computing and network performance of a distributed and network performance of a distributed parallel processing environment using MPI and PVM communication method. *The Journal of Computing in Small Colleges*, portal.acm.org

Hennessy, J., L. and Patterson, D., A. (2002). Computer Architecture: A Quantitative Approach, 3rd Edition. Morgan Kaufmann Publishing Co. Menlo Park, CA. ISBN: 1-55860-596-7 http://www.mkp.com/ca3

Ishfaq Ahmad (1997). Express versus PVM: A performance comparison. Elsevier Science.

Muller, H. A., Orgun, M. A., Tilley, S. and Uhl, J. S. (1993). A Reverse Engineering Approach To Subsystem Structure Identication. John Wiley & Sons, Ltd. Reprinted from Software Maintenance: Research and Practice, 5(4):181-204, December 1993.

Nader Mohamed, Al-Jaroodi, Jameela, Hong Jiang, and Swanson D. JOPI: A Java Object-Passing Interface. *Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande.* Seattle, Washington, USA: Association for Computing Machinery.

Nelisse, A., Kielmann, T., Henri E. B. and Maassen, J. (2001). Object-based Collective Communication in Java.JAVA Grande/ISCOPE 01 Palo Alto CA USA Copyright ACM 2001.

Nelisse, A., Maassen, J., Kielmann, T. and Bal, H., E. CCJ: Object-based Message Passing and Collective Communication in Java. Division of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, The Netherlands. http://www.cs.vu.nl/manta

Nupairoj, N., and Lionel, M. N. (1995). Performance Evaluation of Some MPI Implementations on Workstation Clusters. 0-8186-6895495 1995 IEEE

Orlando, S., Perego, R. and Silvestri, F. (2001). Design of a Parallel and Distributed Web Search Engine. *Proceedings of the 2001 Parallel Computing Conference (ParCo 2001)*. 4-7 September. Naples, Italy: Imperial College Press.

Parent, J., Verbeeck, K., Lemeire, J.(2002). Adaptive Load Balancing of Parallel Applications with Reinforcement Learning on Heterogeneous Networks. *Published in Proc. of Int. Symposium DCABES 2002.* Wuxi, China.

Rossi, A. R. (2004). Distributed Memory Programming and MPI. IBM pSeries Workshop. IBM Research Division Yorktown Heights, New York 10598. June 15 - 16, 2004.

Sanchez, J. L., Garciya, J. M., Fernandez, J.(1996). Improving the performance of parallel triangularization of a sparse matrix using a recofigurable multicomputer. Lecture Notes in Computer Science, vol. 1041, Springer, Berlin, 1996, pp. 493-502.

Seinstra, F.J., Koelma, D., Geusebroek, J.M., Verster, F.C., and Smeulders, A.W.M.(2002). Efficient Applications in User Transparent Parallel Image Processing. *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS.02).* 2002 IEEE

Stout, Q., F.(1987). Supporting Divide-and-Conquer Algorithms for Image Processing. Journal of parallel and distributed computing 4.

Taboada, G. L., Tourino, J., and Doallo, R. (2003). Performance Analysis of Java Message-Passing Libraries on Fast Ethernet, Myrinet and SCI Clusters. *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'03).* 2003 IEEE

Westein, P., Pethick, M., Zhiyi Huang.A performance Comparison of DSM, PVM and MPI. 2003. IEEE. *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies, 2003. PDCAT'2003.*

Zafril Rizal (2005). *Penyelesaian Masalah Matematik MenerusiKaedah MPI Dengan Bahasa Pengaturcaraan Java(MPIJAVA)*. Universiti Teknologi Malaysia: Tesis Sarjana Muda.

Ziemer, R., R., Reid, L., M. (1997). What Have We Learned, And What is New in Watershed Science. *Published in Proceedings of the Sixth Biennial Watershed Management Conference.* S. Sommarstrom, editor. Water Resources Center Report No.92. University of California, Davis (1997).

Zomaya, Y., El-Ghazawi, Frieder, O.(1999). Parallel and Distributed Computing for Data Mining. IEEE Concurrency (7) October-December 1999