JAIST Repository
https://dspace.jaist.ac.jp/

Title	異機種並列計算での負荷分散に関する研究
Author(s)	大岩，博史
Citation	
Issue Date	2003-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/1658
Rights	
Description	Supervisor:松澤 照男，情報科学研究科，修士

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

修 士 論 文

異機種並列計算での負荷分散に関する研究

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

大岩 博史

2002 年 3 月

修 士 論 文

異機種並列計算での負荷分散に関する研究

指導教官　松澤 照男 教授

審査委員主査　松澤 照男　　教授
審査委員　敷田 幹文 助教授
審査委員　堀口 進　　教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

110020 大岩 博史

提出年月: 2002 年 2 月

Copyright © 2003 by Ooiwa Hiroshi

概 要

計算機システム全般の性能向上によって，計算機科学分野は大きく発展した．科学技術計算分野では，大規模計算の実行のために並列計算が良く用いられる．また，高性能ネットワークは，リモートコンピューティングを一般化した．ネットワークの高性能化に伴い，広い地域に任意に配置された計算資源を用いた，並列/分散計算に関する研究が注目されている．

任意の場所に配置された様々な種類の並列計算機を用いる異機種並列計算は，一般的に並列計算機の内部通信機構よりも低速な通信路を利用して計算機間通信を行うため，計算機間のデータ転送時間の発生が問題となる．また，各計算機の性能の違いにおける最適な負荷分散の制御が必要になる．

現在報告されている例は，同機種で PC クラスタを構築し，そのうち CPU 部分を変更した不均質な PC クラスタを用いた研究が報告されているだけである．この研究では，通信負荷に関しては触れていない．

本研究において，並列計算のベンチマークによる，等分割における異機種並列計算をさせたところ，各計算機の処理能力の差異が生じ，処理能力の低いマシンに合わせて全体の計算の処理能力が遅くなってしまい，異機種並列計算を行う利点が見られなかった．

しかし，本研究において負荷分散の最適化を行うことにより，異機種並列計算全体のスピードアップ比が向上した．

行った負荷分散の最適化は，以下の通りである．

1. 異機種並列計算を行う前に，並列計算に用いる各計算機の処理能力の測定．

   各計算機の処理能力に応じた負荷分散を行うため，あらかじめ各計算機の処理能力を知っている必要がある．よって，各計算機 1 台ずつプログラムを実行させて，実行時間を処理能力とする．

2. 各計算機の処理能力に応じた負荷分散を行う．

   データの依存関係の無い計算を行うため，静的負荷分散を行う．静的負荷分散を行うことで，通信負荷を軽減させる目的もある．

3. ネットワーク負荷を考慮して，各計算機に割り当てるタスクの増減を行う．

   ネットワーク負荷を考慮して，ネットワーク負荷の高い計算機に割り当てるタスクを少なくする．

本稿では，負荷分散の最適化の方法，効率を検討する．

目 次

第 1 章　序論 **1**
1.1　背景 1
1.2　目的 2
1.3　本論文の構成 2

第 2 章　並列処理 **3**
2.1　ネットワーク結合コンピュータによる並列処理 3
2.2　速度向上度 4
2.2.1　最大速度向上度 4
2.3　オーバーヘッド 4

第 3 章　異機種並列計算 **6**
3.1　負荷分散 6
3.1.1　動的負荷分散 7
3.1.2　静的負荷分散 7
3.1.3　計算機間の処理時間の差について 7
3.2　並列処理における通信時間 8
3.3　MPI 9
3.3.1　MPICH 9
3.3.2　MPICH のインストール 9
3.4　Nas Parallel Benchmark 10
3.4.1　NPB のインストール 11
3.4.2　結果 11
3.4.3　考察 15

第 4 章　行列ベクトル積の並列化 **16**
4.1　行列ベクトル積 16
4.2　評価方法 16
4.3　ノード 0 の計算機決定方法 18
4.4　等分割における異機種並列計算実行時間 18
4.5　考察 18

第 5 章　負荷分散の最適化 **20**
5.1　各計算機の性能測定 20
5.2　負荷分散の最適化 20
5.2.1　各計算機の処理能力に応じた負荷分散 20
5.2.2　通信負荷を考慮に入れた負荷分散の最適化 22

第 6 章　まとめ **28**
6.1　異機種並列計算 28
6.2　負荷分散の最適化 28
6.3　今後の課題 29

謝辞 **30**

図 目 次

2.1 逐次部分の並列化-アムダールの法則 5

3.1 論理的通信時間 8
3.2 NPB-LU 実行結果 (実行時間) 12
3.3 NPB-LU 実行結果 (Mop/s) 12
3.4 NPB-SP 実行結果 (実行時間) 13
3.5 NPB-SP 実行結果 (Mop/s) 13
3.6 NPB-BT 実行結果 (実行時間) 14
3.7 NPB-BT 実行結果 (Mop/s) 14

4.1 行列ベクトル積並列化 17
4.2 スピードアップ比 18
4.3 等分割におけるスピードアップ比 19

5.1 各計算機の処理能力に応じた負荷分散の最適化後のスピードアップ比 . . . 21
5.2 4 ノード使用した場合のスピードアップ比の比較 23
5.3 8 ノード使用した場合のスピードアップ比の比較 24
5.4 スピードアップ比の比較 25
5.5 負荷予測した結果 27

表 目 次

3.1 NPB で用いた各計算機の仕様 11

4.1 各計算機の仕様 17
4.2 各計算機の処理能力 17
4.3 4 ノード，8 ノード，20 ノードを用いた場合の計算処理時間 19

5.1 各計算機の処理能力 20
5.2 最適化を行った場合の 4 ノード，8 ノード，20 ノードを用いた計算処理時間各計算機の処理能力 21
5.3 最適化を行った場合の 4 ノードを用いた計算処理時間各計算機の処理能力 . 22
5.4 8 ノード使用した場合のスピードアップ比の比較 23
5.5 通信負荷を考慮した計算結果 25
5.6 通信負荷と ‘ping’ コマンドを用いた応答時間との比較 26
5.7 新たに追加した計算機の処理能力 27

第1章　序論

1.1　背景

近年，コンピュータの低価格化と目覚しい性能向上により，あらゆる分野でコンピュータが利用されてきている．テレビや映画などで使用されているコンピュータグラフィックスや，医用画像診断のための3次元画像データ処理，新製品を開発するためのシミュレーション実験など，我々はより身近にコンピュータの恩恵を受けるようになった．

科学技術計算分野では，処理に必要なデータや情報が日増しに複雑かつ膨大な量になっており，大規模計算の実行のために並列計算が良く用いられる．このような複雑で膨大な問題を解決するためには，より高性能なコンピュータを必要とするが，そのような高性能のコンピュータを購入することは，物理的にも経済的にも容易ではない．

一方，高性能ネットワークは，リモートコンピューティングを一般化した．ネットワークの高性能化に伴い，広い地域に任意に配置された計算資源を用いた，並列/分散計算に関する研究が注目されている．また，多くの企業や研究室などの組織内に置いても複数のパーソナルコンピュータ (Personal Computer : PC) やワークステーション (Work-Station : WS) などを相互に接続した，ローカルエリアネットワーク (Local Area Network : LAN) が広く一般に構築されている．このようなコンピュータネットワークの普及によって，コンピュータ間での負荷の分散と資源の共有が可能にり，ネットワークに接続された複数台のコンピュータを一代の仮想的な並列計算機としてみなす「並列処理」の研究が盛んに行われている．利点として，既存のコンピュータネットワークをそのまま並列計算機の一部として利用することが可能であるため，専用の並列計算機を導入する場合と比較すると経済的な問題が大幅に緩和される点が上げられる．

代表的なものの一つに Grid コンピューティングがある．Grid コンピューティングは，広域ネットワーク上に分散配置された計算資源を使う計算システムである．並列計算機およびネットワーク技術の高性能化を背景として，これまで不可能であった大規模な科学技術計算を行うことが可能となってきている．

任意の場所に配置された様々な種類の並列計算機を用いる異機種並列計算は，一般的に並列計算機の内部通信機構よりも低速な通信路を利用して計算機間通信を行うため，計算機間のデータ転送時間の発生が問題となる．また，各計算機の性能の違いにおける最適な負荷分散の制御が必要になる [1].

1.2 目的

本論文では，異機種並列計算における各計算機での計算処理フェーズの差異を軽減させるため，各計算機の処理性能に応じた計算負荷量を設定し，異機種並列計算での全体の計算時間の短縮が可能となる最適化方法を提案し，効果を検討する．

1.3 本論文の構成

本論文は次章以降，以下のような内容で構成されている．

第 2 章 並列処理

一般的な並列処理について説明する．

第 3 章 異機種並列計算

異機種並列計算について説明し，異機種並列計算における特徴を示す．

第 4 章 行列ベクトル積の並列化

行列ベクトル積の並列化の方針と計算結果について説明する．

第 5 章 負荷分散の最適化

第 4 章で行った計算の負荷分散の最適化を行い，効率の検討を行う．

第 6 章 まとめ

第2章　並列処理

2.1　ネットワーク結合コンピュータによる並列処理

コンピュータの開発当初から研究の対象であった並列処理コンピュータは，1980 年代になると商用機も開発され，実用されるようになった．21 世紀においてはその役割が増加することは明らかである．しかし，専用の並列コンピュータはまだ高価であり，気軽に利用できる状況ではない．

一方，1990 年代のパーソナルコンピュータと高速ネットワークの発展によって，市販の低価格 PC やワークステーションをネットワークで結合して並列マシンとして使うことが可能になった [2]．現在では，このようなワークステーションクラスタあるいはワークステーションネットワークが，スーパーコンピュータや並列コンピュータシステムの代替として非常に魅力的なことが広く認知されている．クラスタ型並列計算機の利用には特殊設計のマルチプロセッサに比べていくつもの利点があるが，重要なものは次ぎの通りである．

- 非常に高性能なワークステーションや PC が低コストで容易に入手できる．
- 最新のプロセッサをすぐにシステムに組み込める．
- 既存のソフトウェアを使用したり，変更して使える．

ワークステーションの並列計算への利用は，最初は，汎用計算機のワークステーションネットワークが既に存在していたので興味がもたれた．ワークステーションはその名前が示唆するように，各種のプログラミングやコンピュータ関連の活動に前から使用されていたが，1980 年代後半にそれらのワークステーションに並列プログラミング用ソフトウェアツールを提供するためのプロジェクトが実施された．最も有名なのは，並列仮想マシン (Parallel Virtual Machine ; PVM) で，これは後に広く普及した．その後メッセージ通信インターフェース (Message Passing Interface ; MPI) と呼ばれる標準メッセージ通信ライブラリも定義された (MPI については，3.3 節で詳しく説明する．)．強力で安いワークステーションや PC の登場によってクラスタ型並列計算機は，ますます重要な概念となっている [3].

2.2 速度向上度

マルチプロセッサシステムの単一プロセッサシステムに対する相対性能の尺度が速度向上度 $S(n)$ で，次のように定義される．

$$S(n) = \frac{\text{1 プロセッサを用いたときの実行時間}}{\text{n プロセッサのマルチプロセッサでの実行時間}} = \frac{t_s}{t_p} \tag{2.1}$$

ここで，t_s は単一プロセッサでの実行時間，t_p はマルチプロセッサでの実行時間で，$S(n)$ はマルチプロセッサによる速度向上度を与える．

並列解を逐次解と比較するには，単一プロセッサ上で走るアルゴリズムのうちの知られている中で最も高速なものを使用する．したがって，並列プログラムのアルゴリズムとは (通常) 異なることもある．

2.2.1 最大速度向上度

1 プロセッサでしか実行できない部分が存在すると仮定すると，理想的な状況は，それ以外の時間は全てのプロセッサが同時実行するというものである．並列タスクに分解できない処理の割合を f とし，計算を並列実行する部分に分解するオーバヘッドがないとすると，n プロセッサで計算を行う時間は，図 2.1 に示すように $ft_s + (1-f)t_s/n$ で表される．図に示したのは逐次部分が計算の最初にある場合であるが，計算全体にどのように分布していてもよい．したがって，速度向上度は以下の式で表される．

$$S(n) = \frac{t_s}{ft_s + (1-f)t_s/n} = \frac{n}{1 + (n-1)f} \tag{2.2}$$

この式はアムダールの法則 (Amdahl, 1967) として知られている．相当の速度向上を達成するには，並列プロセスによって実行される計算の割合が，計算全体の大部分であることが必要である．無限個のプロセッサがあっても最大向上度は $1/f$ に制限される．すなわち，以下の式のようになる．

$$\lim_{n\to\infty} S(n) = \frac{1}{f} \tag{2.3}$$

たとえば，わずか 5%の処理が逐次であっても，最大速度向上度はプロセッサ数によらず 20 である．アムダールは 1960 年代に単一プロセッサシステムがよいことを示すためにこの議論を用いた．もちろん，最大速度向上度 20 でも十分大きいと反論できるし，逐次処理の割合が非常に小さい処理対象では，何桁もの改善が可能である [4].

2.3 オーバーヘッド

オーバーヘッドとは，目的とする処理に直接波関係野内処理，あるいはその処理に要する時間のことを指す．並列処理においてオーバーヘッドとして現れ，速度向上を制限する要因背屈もあるが，重要なのは次のものである．

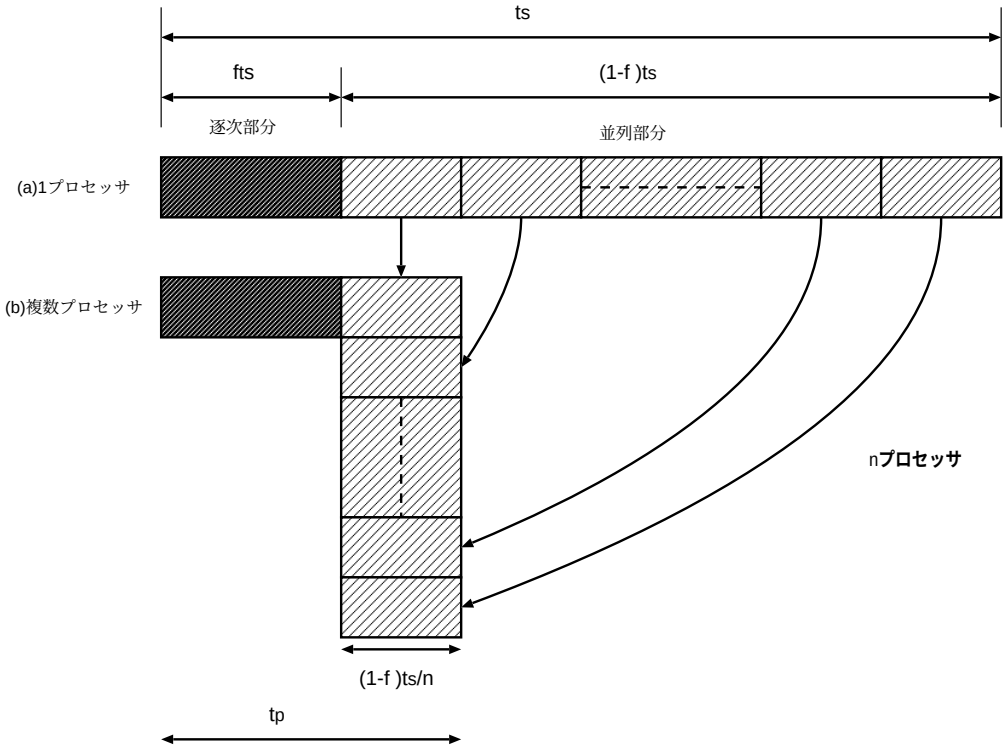

図 2.1: 逐次部分の並列化-アムダールの法則

1. 一部のプロセッサしか有用な仕事をしておらず，残りのプロセッサはアイドルである時間 (この中には 1 台のプロセッサのみが計算の本質的に逐次の部分を実行している時間を含む).

2. メッセージ通信のための通信時間.

3. 定数を局所的に再計算するなど，逐次版には現れない並列版における余分な計算時間.

第3章　異機種並列計算

近年，超並列計算機に代わってPCやワークステーションをネットワークで繋いだクラスタ型並列計算機が注目されている点は，前章までに述べた．クラスタ型並列計算機はコモディティなハードウェアによって構築されていることから，本質的に多様性を持っている．その多様性はCPU性能やネットワークハードウェアの種類，メモリ容量の差異，場合によってはOSやCPUアーキテクチャまで及ぶことがあり，今後クラスタの普及に伴い，このような，各計算機の性能が不均質なクラスタ型並列計算機 (異機種並列計算機) の増加は必至である．

異機種並列計算においてはノード間の性能差を考慮して負荷分散をすることが異機種並列計算全体の性能向上に必要である．また，超並列計算機とは違い，超並列計算機内部の通信機構より低速なネットワークを使用しなくてはならないため，データ転送における通信負荷を考慮しなければならない．

しかし，現在報告されている例は，同機種でPCクラスタを構築し，そのうちCPU部分を変更した不均質なPCクラスタを用いた研究 [5] が報告されているだけである．通信負荷を考慮した場合や，異機種における並列化がなされていない．

本章では，異機種並列計算の特徴を把握し，ベンチマークを用いて，今までの等分割による並列計算方法で異機種並列計算を行い，異機種並列計算の基本性能，特徴を確認する．

3.1　負荷分散

負荷分散とは，マルチプロセッサを構成する各プロセッサの能力に応じて負荷が均衡化するようにタスクを割り付けることをいう．つまり，負荷分散はマルチプロセッサにおけるそれぞれのプロセッサ利用率を高めることにより，全体としてスループットを向上させることを目的としたものである．また，負荷分散は，目的に応じてタスクをいつどのような順序で実行させるかを決めることと，どのプロセッサに割り当てるかを決めることからなるスケジューリングの一種と考えることができる．この制御がうまくいかなければ，通信遅延の増加や，プロセッサのアイドル時間が長くなるなどの理由により，処理の高速化が思うように図れなくなる．

以下に，代表的な負荷分散の手法について述べる．

3.1.1 動的負荷分散

動的負荷分散では，あらかじめもとの仕事を多くの細かいタスクに分割しておく．タスクの分割数は，計算機数よりも十分に多くしておく．各サーバはタスクを一つずつ処理し，処理の終わった計算機はすぐに次のタスクを処理する．したがって，高速な計算機はその分多くのタスクを処理し，低速な計算機は少ないタスクだけを処理することになる．

動的負荷分散では，静的負荷分散に比べると計算機間の通信回数が多くなるため，それに伴って通信に必要な処理時間が大きくなる．したがって低速なネットワークを用いる可能性のある異機種並列計算には向かない．また，負荷分散のアルゴリズムが複雑になるため，スケジューリングに必要な処理時間も大きくなる可能性がある．

3.1.2 静的負荷分散

静的負荷分散では，各サーバに対するタスクの割り当て量はユーザによって決定される．タスクの割り当て量は，プログラム作成時，あるいはプログラム実行開始時に決定される．各サーバの仕事量は，処理の開始前に完全に決められてしまうため，処理中に各サーバのタスクの割り当て量を変更することはできない．

静的負荷分散では，計算機間の通信の回数が少ないため，計算機間の通信にともなうオーバーヘッドを抑えることができる．この利点は，異機種並列計算においては，ネットワーク上に分散している計算機を用い計算を行う点と，ネットワークに低速な回線を用いる可能性があり，ネットワーク負荷が増大する点を考慮する必要があるので重要である．また，負荷分散に複雑なアルゴリズムを必要としないため，スケジューリングに必要な処理時間も小さく抑えることができる．通信時間も加味した，処理時間に対する制度の高い予測ができれば，静的負荷分散では良好な結果が得られるので，実行前にタスクの実行時間が予測しやすい定型的な処理に向いている．

3.1.3 計算機間の処理時間の差について

あるタスクを負荷分散によって処理するために，タスクを均一に分割して各計算機で処理したとする．このとき各計算機の計算処理速度に差異があった場合，計算処理能力の高い計算機が先に仕事を終わらせてしまって，その後その計算機がなにもしないという状態が発生する．これを計算機間の処理終了時間の差とよぶ．計算機間の終了時間の差が発生すると，計算機の利用効率が低下するため，全体の計算時間の性能低下につながる．

特に異機種並列計算では，各計算機の処理能力の差異が生じるので，負荷分散を行う場合には，できるだけ計算機間の処理終了時間の差が発生しないようにタスクを割り当てることが重要になる．

3.2 並列処理における通信時間

並列処理では，計算ステップ数の決定だけでなく，通信オーバヘッドの推定が必要である．メッセージ通信システムでは，問題の全実行時間中にメッセージ通信時間を考慮する必要があって，並列実行時間 t_p は，計算部分 t_{comp} と通信部分 t_{comm} との和によって構成される．

$$t_p = t_{comp} + t_{comm} \tag{3.1}$$

クラスタ型並列計算機での通信時間 t_{comm} は，ネットワーク結合網の構造や通信時の利用率などの多くの要因に依存し，簡単な式は無いので，第一近似として次の式を用いる．

$$t_{comm} = t_{startup} + nt_{data} \tag{3.2}$$

ここで，$t_{startup}$ は起動時間，メッセージレイテンシとよばれるもので，実質的にデータの無いメッセージを送信する時間である．実際に，そのようにして簡単に測定できる．この中には，発信元でメッセージをパックする時間と，宛先でアンパックする時間が含まれる．レイテンシという用語は全通信遅延を表すのにも用いられるので，ここでは，起動時間という用語を使う．起動時間は一定と仮定する．項 t_{data} は 1 データ語の転送時間で，これも一定で n データ語あるものとする．転送速度は普通，ビット/秒で表され，データ語が b ビットとすると，b/t_{data} ビット/秒である．

この式を図 3.1 に示す．実際のシステムではこのような完全に線形の関係は得られない．通信媒体上での競合など多くの要因が通信時間に影響するし，実システムでは発信元と相手先が直接接続されていないために，メッセージが中間ノードを通過しなければならないこともあることをこの式は無視している．また，パケットにデータ以外の情報を入れるためのオーバヘッドは一定で，$t_{startup}$ の一部として含められるものと仮定している．

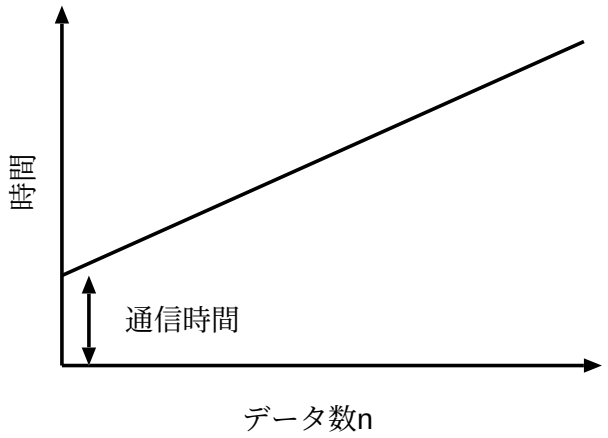

図 3.1: 論理的通信時間

3.3 MPI

Message Passing Interface (MPI) フォーラムは並列システムのプログラミングの規格の開発に，新しい言語の仕様を決めるのではなく，C または Fortran プログラムから呼び出せる関数群からなるライブラリの仕様を決めたのである．このライブラリの基本はメッセージパッシング (message passing) によって並列処理を実現するのに用いられる関数群からなる小さなグループである．メッセージパッシング関数は単にあるプロセスから他のプロセスへデータを明示的に送る関数である．メッセージパッシングは並列処理を実現する強力で非常に一般的な手法である．メッセージパッシングプログラムは極めて効率の良い並列プログラムを書くのに用いられ，メッセージパッシングは現在，数多くの種類の並列計算機上のプログラミングに最も広く使われている．

本論文では，MPI の実装系である MPICH をインストールして用いる．

3.3.1 MPICH

MPICH は，アメリカのゴードン国立研究所が模範実装として開発し，無償でソースコードを配布したライブラリである．移植しやすさを重視した作りになっているため盛んに移植が行われ，世界中のほとんどのベンダの並列マシン上で利用することができる．特に，MPICH では UNIX 系に限らず Windows 系へのサポートも充実している．さらに，SMP，Myrinet などハード面にも対応している上，DQS，Globus といった様々なツールを使用できることも大きな特徴の一つである．

3.3.2 MPICH のインストール

本論文で，異機種並列計算を行う場合，MPI による異機種通信を試みた．異機種計算に用いたのは，研究室内ワークステーション 3 台，研究室内 PC1 台，学内 PC クラスタ 16 台である．この中で，PC クラスタは MPI を実装していたが，他のマシンは未実装であった．よって，MPI の実装を MPICH を用いて行う．以下に，実装手順を示す．

(1) MPICH の入手

ANL の MPI のページ[1]から，“mpich.tar.gz” ファイルをダウンロードしてくる．

(2) MPICH のコンパイル

ダウンロードしてきた，ファイルを展開する．展開してできたディレクトリに移動して，‘.configure’ を実行する．オプション ‘-prefix=ディレクトリ名’ でインストールするディレクトリを指定する．

[1] http://www-unix.mcs.anl.gov/mpi/mpich/

(3) インストール

コンパイルできたら，‘make’,‘make install’ でインストールをする．

(4) マシンファイルの設定

実行用の ‘mpirun’ が読む，ホスト名一覧がインストールしたディレクトリの ‘/share/machines.LINUX’ にある．このマシンファイルに，自分の使いたいホスト名を列記する．

3.4 Nas Parallel Benchmark

本節では，異機種並列計算の基本性能を把握するため，ベンチマークを実施した．今回用いたベンチマークは，Nas Parallel Benchmark (NPB) である．NPB は，NASA Ames Research Center で作成された，並列コンピュータの性能評価のためのマクロベンチマークである．主に大規模な CFD Application における計算とデータの送受信性能を評価することができる．ソースコードは Fortran77 と MPI を用いて記述された並列版と Fortran77 のみで記述された遂次版が存在し，両方ともインターネット上で公開されている[2]．また，様々なメーカの計算機システムで実行された結果についても同様に公開されている．NPB は 5 つのカーネルプログラムと 3 つの CFD アプリケーションプログラムから構成されている．

本稿では，その中から CFD アプリケーションである以下のプログラムを用いてベンチマークを計測した．

- LU: 3 次元圧縮性 Navier-Stokes 方程式を 5×5 のブロック上下三角行列方程式を簡易化 SSOR 法で解いている．この解法は，小さなデータを多く交換する．MPI を用いた場合，使用するノード数は 2 の累乗である必要がある．
  - gridsize = 33×33×33
- SP: 3 次元圧縮性 Navier-Stokes 方程式を類似した構造の非優位対角なスカラ 5 重方程式で解いている．この解法は，大きなデータを少ない回数で交換する．MPI を用いた場合，ノード数は平方数にする必要がある．
  - gridsize = 33×33×33
- BT: SP と同様の方程式を非優位的な 5×5 のブロックサイズのブロック 3 重対角方程式で解いている．MPI を用いた場合，ノード数は平方数にする必要がある．
  - gridsize = 33×33×33

[2]http://science.nas.nasa.gov/Software/NPB

3.4.1 NPB のインストール

(1) NPB の入手

NPB のサイトから，プログラムのソースをダウンロードしてくる．

(2) 設定ファイルの変更

ダウンロードしてきた，ファイルを展開する．make する前に，環境に応じて設定ファイルを変更する必要がある．NPB2.3/config/ ディレクトリにに make.def.template ファイルがあるので，これをコピーして，make.def ファイルを作成し，‘mpicc’，‘mpirun’，MPI ライブラリのパスなどを設定する．

(3) make

NPB2.3 ディレクトリで次のように make します．NPB では，さまざまな種類のベンチマークが可能で，それぞれについて実行するプロセス数，問題のクラスを指定することができる．それらはすべて別々の実行ファイルになるため，make の際に対象問題，プロセス数，クラスを指定して make しる必要がある．例えは，S クラスの LU ベンチマークを 2 プロセスで実行したい場合には，以下のように make する．

make LU NPROCS=2 CLASS=S

(4) ベンチマークの実行

make を行うと，NPB2.3/bin/ディレクトリに実行ファイルが生成される．‘mpirun’ で実行できる．

3.4.2 結果

ベンチマークに用いた各計算機の仕様を表 3.1 に示す．

表 3.1: NPB で用いた各計算機の仕様

	PC1	PC2	PC クラスタ (1 node)
CPU	Pentium4-2.8GHz	Pentium2-300MHz	Pentium3-1GHz
MEMORY	1GB	128MB	512MB
OS	Vine-Linux	Vine-Linux	RedHat-Linux

結果として，NPB から得られる，

- 計算時間
- Mop/s (Million operation per second) 浮動小数点演算数を計算時間で割ったもの．

の結果を図 3.2 と，図 3.3 と，図 3.4 と，図 3.5 と，図 3.6 と，図 3.7 に示す．

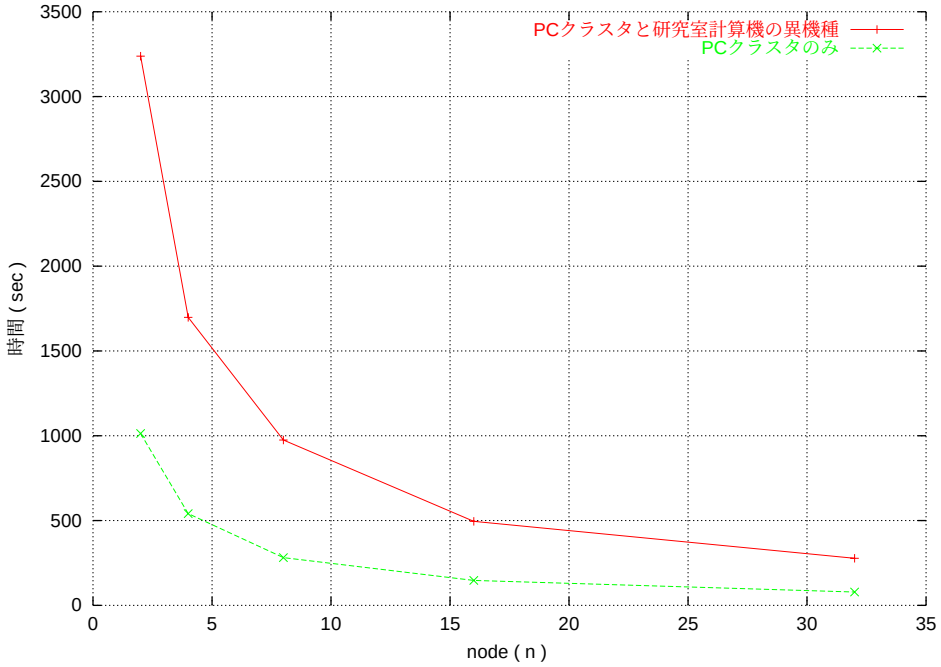

図 3.2: NPB-LU 実行結果 (実行時間)

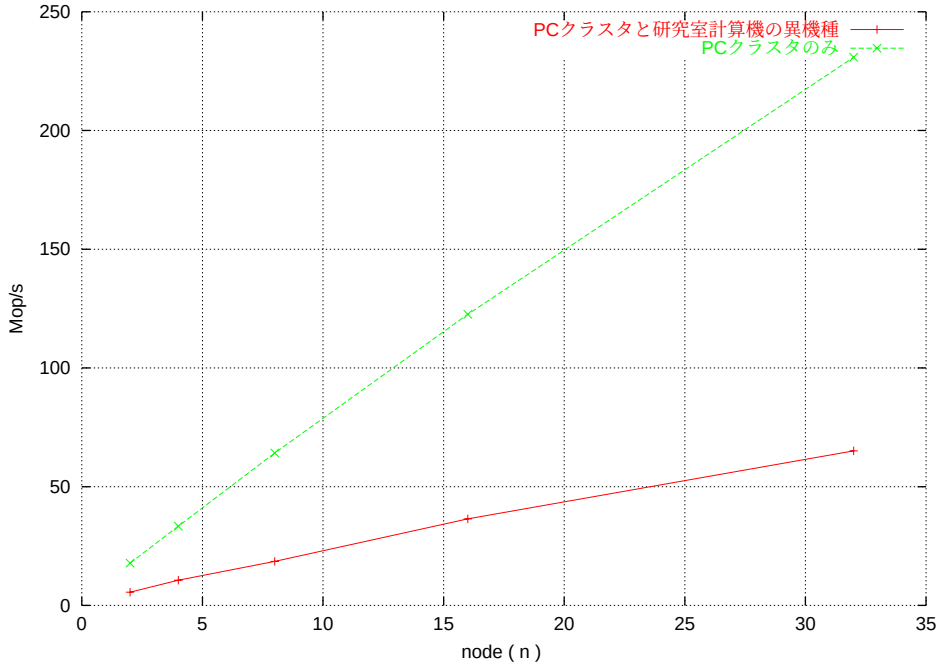

図 3.3: NPB-LU 実行結果 (Mop/s)

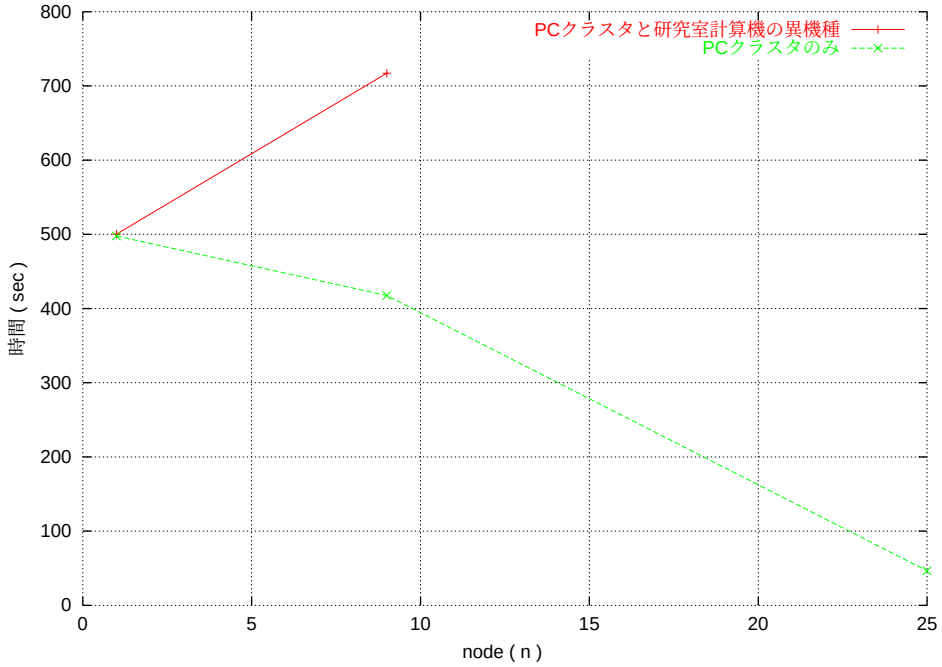

図 3.4: NPB-SP 実行結果 (実行時間)

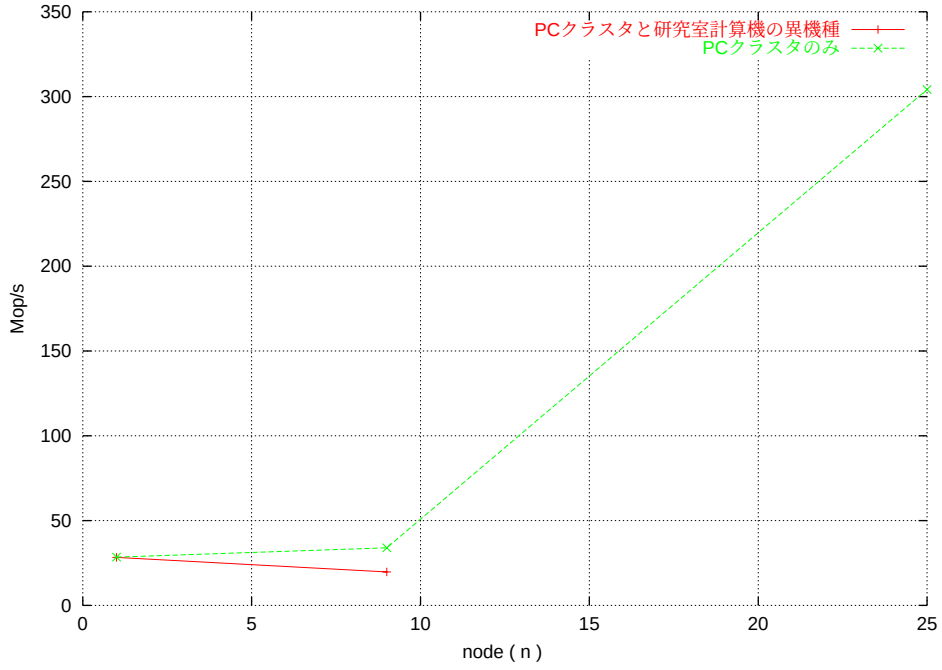

図 3.5: NPB-SP 実行結果 (Mop/s)

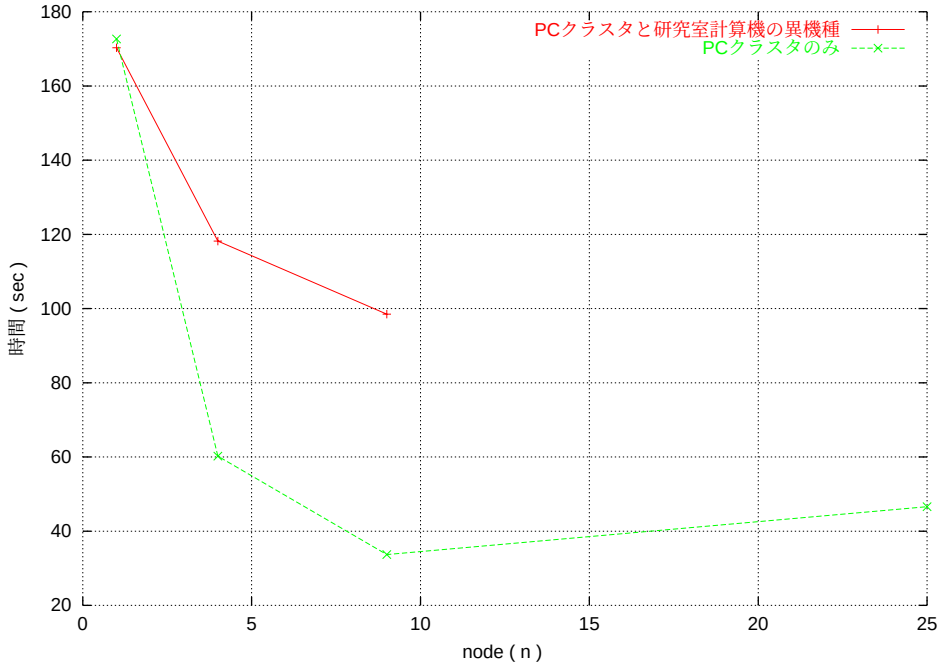

図 3.6: NPB-BT 実行結果 (実行時間)

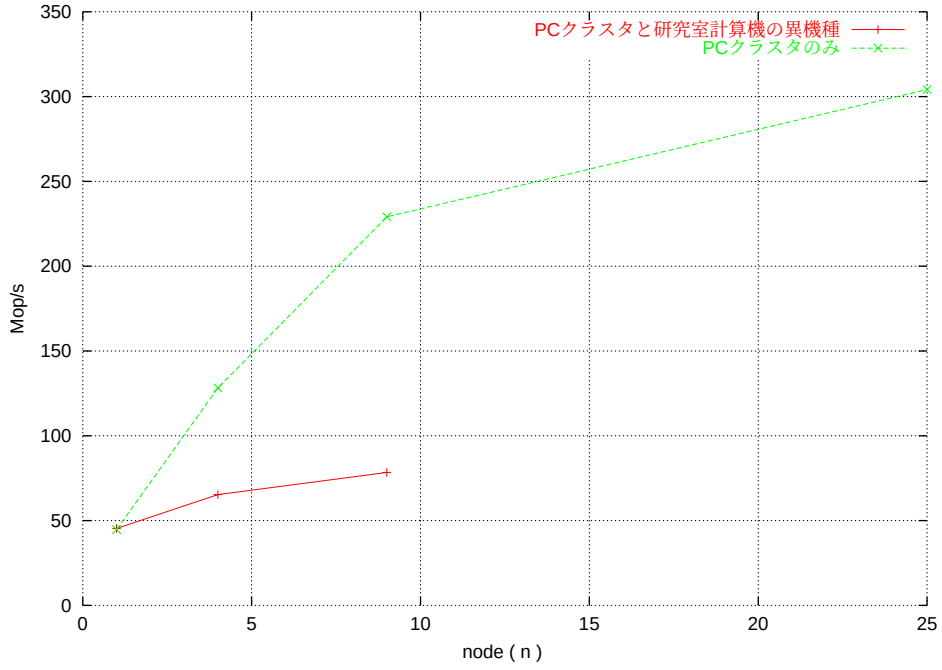

図 3.7: NPB-BT 実行結果 (Mop/s)

3.4.3 考察

図 3.2 から図 3.7 まで，赤色ラインが異機種並列計算，緑色ラインが PC クラスタのみ用いた計算結果となっている．

図 3.2 より，実行時間は，2 ノード用いた場合の実行時間差が最大で約 3 倍の差がある．ノード数が増えていくにつれて実行時間差が縮小している．全体的には，異機種並列計算を行った場合が実行時間が大きくなっている．

図 3.3 より，Mop/s 値は逆にノード数が増加していくと差が拡大していき．最大で，165.75 (Mop/s) の差がある．全体的には，異機種並列計算を行った場合が Mop/s 値が小さい．

図 3.4 から図 3.6 より，異機種のベンチマークでは，9 ノードまでしか計測できなかった．それ以上のノードでは，エラーを出してしまう．この結果も，異機種並列計算結果より，PC クラスタの結果の方が実行時間，Mop/s 値ともに良い．

以上のことから，異機種並列計算では，PC クラスタ 1 ノードより計算処理性能の高い計算機 1 台と，計算処理性能の遅い計算機 1 台とを含めて計算を行ったが，処理性能の低い計算機に合わせるように，全体の計算処理能力が落ちていることがわかった．処理能力の高い計算機が有効利用されていないことが確認できる．

第4章　行列ベクトル積の並列化

NPB ベンチマークは，異機種並列計算用に作られていないので，異機種並列計算の最適化を検討するために別のプログラムを用いる必要がある．

今回採用した計算は，行列ベクトル積の計算である．これは，数値流体問題を扱う上で，大規模な連立方程式を解く際の基本の計算となる．

今回は，700×700 の行列ベクトル積の計算を 100000 回繰り返すプログラムを異機種並列計算で行うことにする．

本論文では，この行列ベクトル積の計算を並列化して，異機種並列計算における負荷分散の最適化を行う．

4.1　行列ベクトル積

行列ベクトル積の並列化の方法は，行列をノード 0 で，各計算機に分配する分に分割し，各計算機に分配する．各ノードは，与えられた行列で計算を行い，計算結果をまたノード 0 に送信する．という方法で並列化を行う．概念図を図 4.1 に示す．ノード 0 から各ノードへ行列を送る場合と，各ノードの計算が終わり計算データをノード 0 へ送る場合とに，全体の同期を取っている．

4.2　評価方法

評価方法を以下に示す.

- プログラム全体の実行時間の測定
- ノード 0 の計算時間測定
- 通信時間 (待ち時間を含む) = 全体の実行時間 - ノード 0 の計算時間
- 速度向上比 = PC1 実行時間 / 全体の実行時間

本論文で用いた，各計算機の仕様を表 4.1 に示す.
各計算機 1 台での計算時間を，表 4.2 に示す.

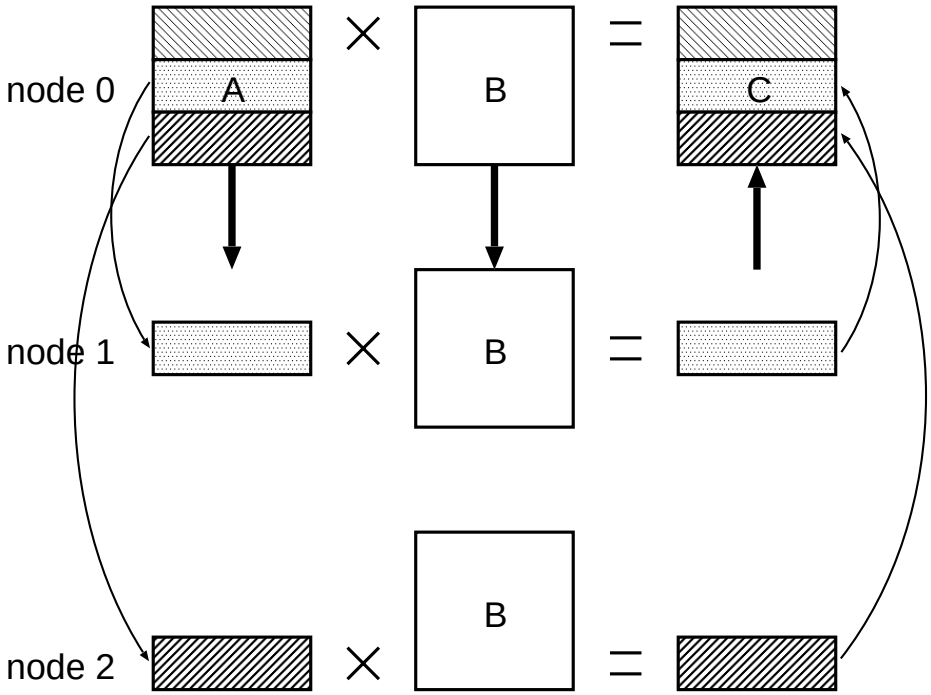

図 4.1: 行列ベクトル積並列化

表 4.1: 各計算機の仕様

	PC1	WS1, WS2	WS3	PC クラスタ (1 node)
CPU	Pentium4-2.8GHz	Sparc 550MHz	Sparc 330MHz	Pentium3-1GHz
MEMORY	1GB	640MB	128MB	512MB
OS	Vine-Linux	Solaris-9	Solaris-7	RedHat-Linux

表 4.2: 各計算機の処理能力

	処理時間 (sec)
PC1	508
WS1, WS2	2308
WS3	3214
PC クラスタ (1 node)	1061

4.3 ノード0の計算機決定方法

ここで，2ノード用いた場合の異機種並列計算実行時間を図4.2に示す.

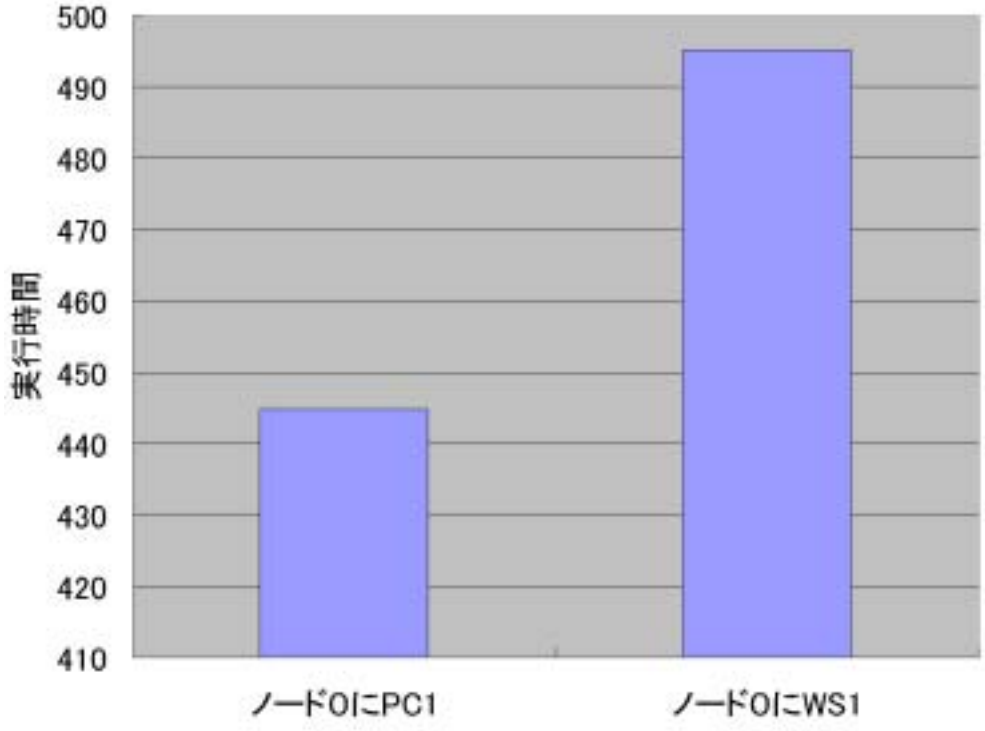

図 4.2: スピードアップ比

図4.2と表4.2より，ノード0に計算実行時間の短い計算機を用いた場合が，異機種並列計算全体の実行時間が短縮されていることがわかる.

よって，今後は，ノード0にPC1を用いることにする.

4.4 等分割における異機種並列計算実行時間

2ノード，4ノード，8ノード，20ノードの計算機を用いた場合の計算処理時間の結果を表4.3に示す.

4.5 考察

表4.3より，図4.3を示す.

図4.3より，等分割の負荷分散では，1ノードから2ノードに増やした場合，極端にスピードアップ比が落ち込んでいるのがわかる．これは，PC1とWS1との処理能力の性能

表 4.3: 4 ノード，8 ノード，20 ノードを用いた場合の計算処理時間

利用した計算機の種類	処理時間 (sec)	PC1 とのスピードアップ比 (倍)
PC1,WS1	836.17	0.61
PC1,WS1,WS2,WS3	426.59	1.20
PC1,WS1,WS2,WS3,PC クラスタ× 4 台	627.82	0.81
PC1,WS1,WS2,WS3,PC クラスタ× 16 台	426.89	1.19

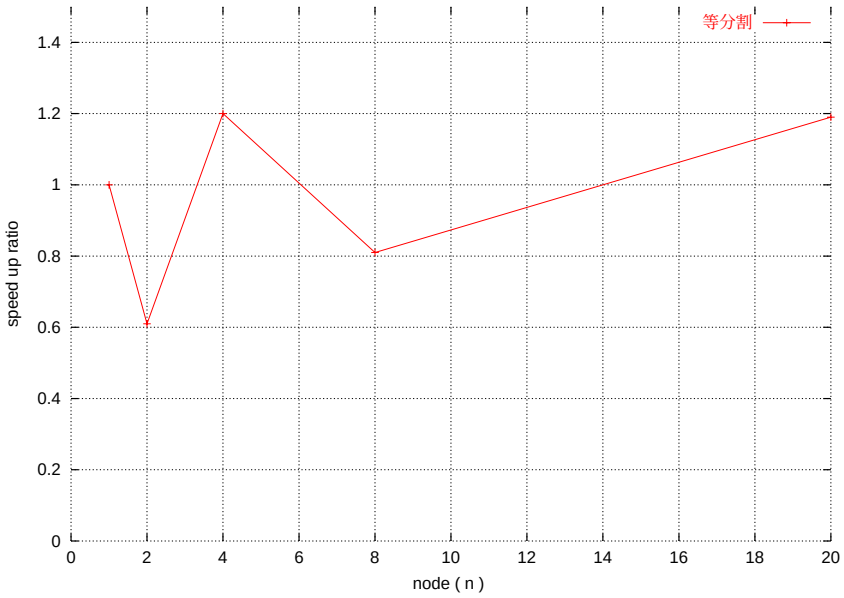

図 4.3: 等分割におけるスピードアップ比

差が大きく開いているため，WS1 の計算時間が遅く，全体の計算時間が遅くなってしまったと考えられる．

また，全体的に図 4.3 を見ると並列化におけるスピードアップ比が向上されていないことがわかる．

よって，各計算機の処理能力に応じた負荷分散の最適化を行い，スピードアップ比の改善を試みる必要がある．

第5章　負荷分散の最適化

5.1　各計算機の性能測定

異機種並列計算では，前章で述べた通り各計算機の処理能力に応じて負荷分散を行わないと，処理能力の低い計算機に合わせて並列化による全体の計算性能が低下してしまう．よって，各計算機の処理能力を調べ，処理能力に応じた負荷分散を行う必要がある．今回用いる方法は，各計算機単体で同じ計算を行い，計算時間をそのマシンの性能として，各計算機間で性能比を求めこの性能比に応じて，負荷分散を行った．各計算機の処理能力は，プログラムを5回実行させ平均をとり，各計算機の処理能力を測定する．

各計算機の処理能力を表5.1に示す．

表 5.1: 各計算機の処理能力

	処理時間 (sec)	性能比
PC1	508	6.33
WS1, WS2	2308	1.39
WS3	3214	1
PC クラスタ (1 node)	1061	3.03

5.2　負荷分散の最適化

5.2.1　各計算機の処理能力に応じた負荷分散

表5.1の結果より，各計算機の処理能力に応じて，分割する行列数を変更する．なお，計算前に分割する行列数を確定し，その後の分割数の変更は行わない，静的負荷分散を行う．

測定結果を以下に示す．

結果

4 ノードの場合，8 ノードの場合，20 ノードの場合の負荷分散の最適化の結果を，表 5.2 に示す.

表 5.2: 最適化を行った場合の 4 ノード，8 ノード，20 ノードを用いた計算処理時間各計算機の処理能力

利用した計算機の種類	処理時間 (sec)	PC1 とのスピードアップ比 (倍)
PC1,WS1	444.64	1.14
PC1,WS1,WS2,WS3	349.07	1.46
PC1,WS1,WS2,WS3,PC クラスタ× 4 台	661.68	0.77
PC1,WS1,WS2,WS3,PC クラスタ× 16 台	413.57	1.23

表 4.3 と表 5.2 との結果を比較したのを，図 5.1 に示す.

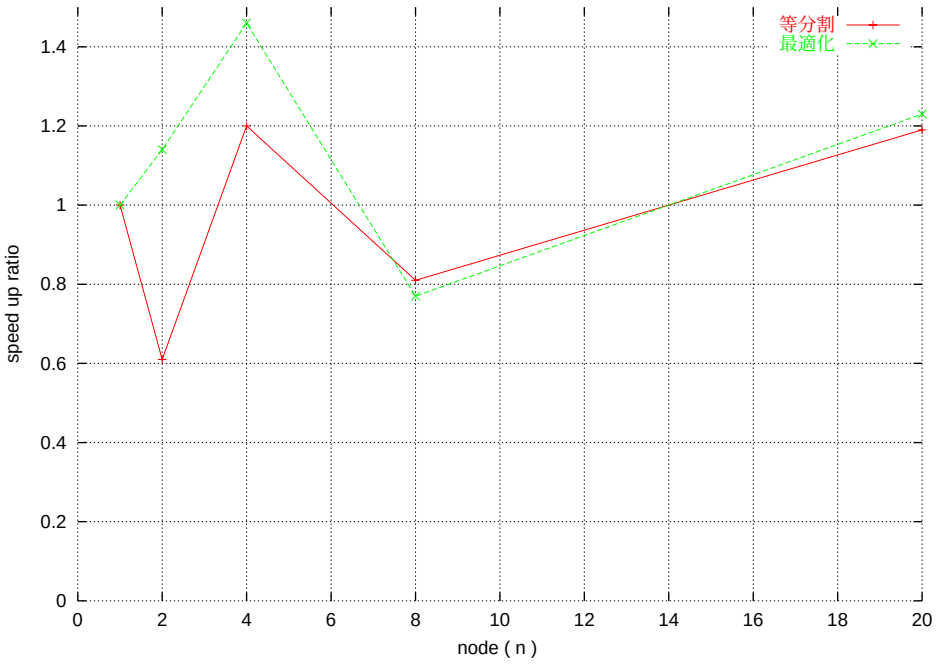

図 5.1: 各計算機の処理能力に応じた負荷分散の最適化後のスピードアップ比

考察

図5.1より，赤色のラインは等分割によって並列化した場合のスピードアップ比，緑色のラインは，各計算機の処理能力に応じて負荷分散をした場合のスピードアップ比となる．

4ノードまではスピードアップ比が上昇しており，計算機の処理能力に応じた負荷分散の最適化がうまく行われていることがわかる．

しかし，8ノード以上では，ノード数の増加による，スピードアップ比の向上が見られない．

このスピードアップ比が見られない問題に関して，今回の計算ではデータの依存関係が無い計算なので，データの依存関係による計算時間の遅れは考えられない．しかし，計算方法が左行列をノード数分で分割し，分割分を各ノードに送信しているため，ノード数が増加するにつれて，通信の負荷が増加し，全体の処理時間がかかることが考えられる．また，4ノードの計算までは，研究室内ネットワーク上にあるPCとワークステーションを用いていたが，8ノード以降では，研究室外のPCクラスタを用いている．よって，研究室内の計算機と研究室外の計算機との通信負荷が増大したため，8ノード以降のスピードアップ比が向上しなかったと考えられる．

検証のため4ノードでの計算を，PC1，WS1，PCクラスタ×2台を用いた場合を，表5.3に示す．

表 5.3: 最適化を行った場合の4ノードを用いた計算処理時間各計算機の処理能力

利用した計算機の種類	処理時間 (sec)	PC1とのスピードアップ比 (倍)
PC1,WS1,PCクラスタ2×台	593.94	0.86

表5.2の4ノードの結果と，表5.3との結果を比較したものを図5.2に示す．

同じ4ノードでの計算を行う場合，理論では計算機性能の高い研究室外のPCクラスタを用いた場合の方が，スピードアップ比が高いが，図5.2より，研究室内の計算機だけを用いた場合の方がスピードアップ比が高い．これにより，通信の負荷により，スピードアップ比が向上しない点がわかる．

以上の結果より，次節で，通信負荷を考慮に入れた負荷分散の最適化を行う．

5.2.2 通信負荷を考慮に入れた負荷分散の最適化

通信負荷を考慮に入れる方法として，ネットワーク負荷が大きくなる研究室外のPCクラスタに渡すデータを，各計算機の処理能力に応じて負荷分散したデータ量よりも，少ないデータを渡す方法をとる．

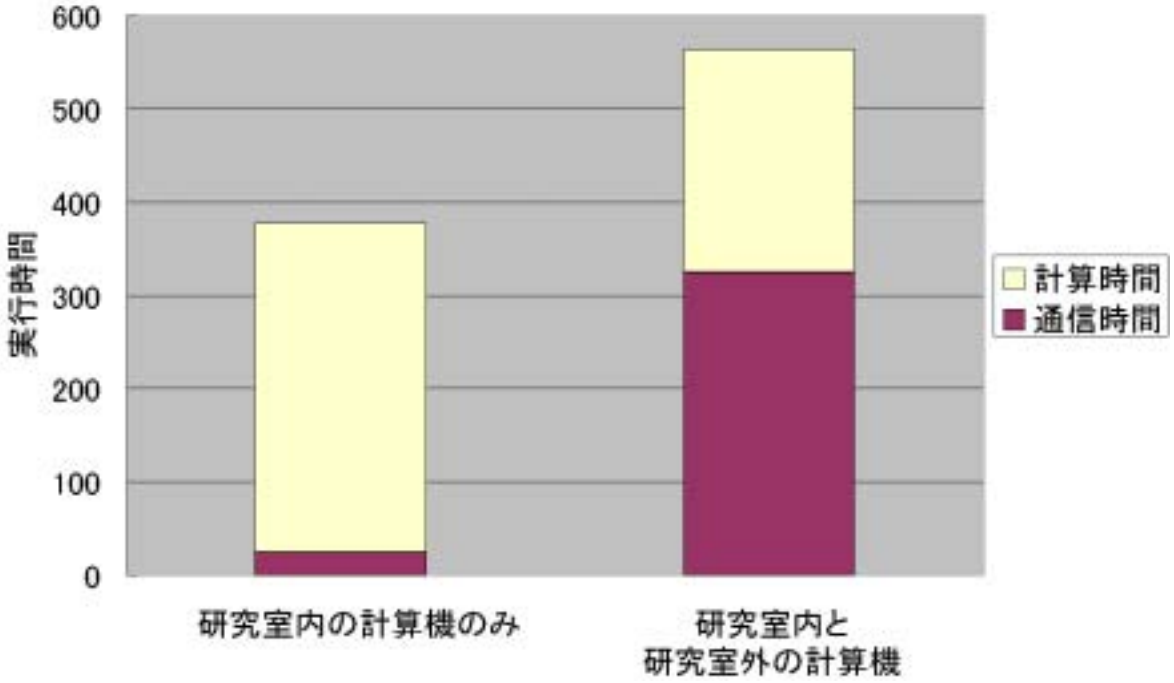

図 5.2: 4 ノード使用した場合のスピードアップ比の比較

結果

以下，表 5.4 に，通信負荷を考慮した負荷分散の最適化結果を示す.

表 5.4: 8 ノード使用した場合のスピードアップ比の比較

負荷を軽減した割合	処理時間 (sec)	PC1 との処理能力比 (倍)
PC クラスタの負荷 80%	595.35	0.85
PC クラスタの負荷 65%	436.25	1.16
PC クラスタの負荷 55%	385.92	1.32
PC クラスタの負荷 45%	322.45	1.58
PC クラスタの負荷 40%	356.63	1.42

考察

表 5.2 と，表 5.4 との比較を図 5.3 に示す.

図 5.3 より，PC クラスタへの行列の割り当てを，各計算機の処理能力に応じて分割した後さらに 65%以下に割り当てを減らし場合，全体のスピードアップ比は，徐々に改善さ

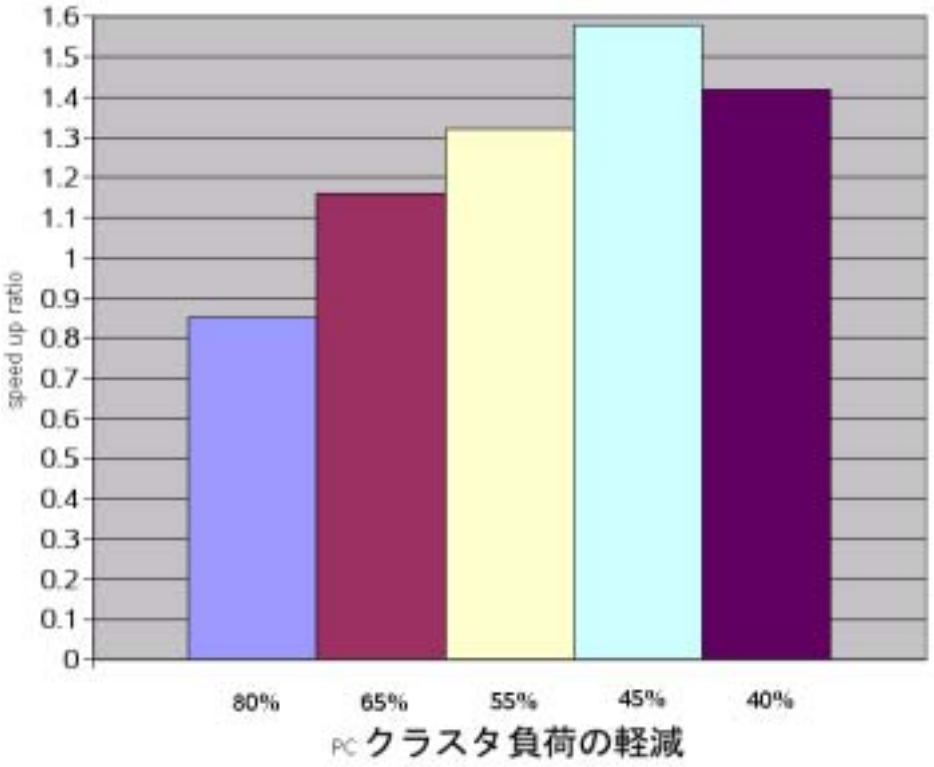

図 5.3: 8 ノード使用した場合のスピードアップ比の比較

れ，PC クラスタへの割り当てを 45%にまで減らした場合が最高値となり，スピードアップ比は 1.58 倍となった．その後は，逆にスピードアップ比が減少していった．

20 ノード用いた計算結果も，40%の負荷では 359.70 秒，55%の負荷では 374.53 秒となり，45%の場合が一番良い結果となった．45%削減をノード数 20 の場合にも当てはめ，2 ノード，4 ノード，8 ノード，20 ノード全体の計算させた結果を表 5.5 にまとめて示す．

表 4.3，表 5.2，表 5.5 の結果をまとめ比較したものを図 5.4 に示す．

図 5.4 より，赤色ラインは等分割した場合のスピードアップ比，緑色ラインは各計算機の処理能力に応じて負荷分散を行った場合のスピードアップ比，青色ラインは処理能力に応じて負荷分散を行った後，通信負荷を考慮に入れて各計算機に分配するデータの増減を調節した場合のスピードアップ比となる．

通信負荷を考慮に入れた場合，8 ノードまではスピードアップ比が向上した．8 ノードから 20 ノードではスピードアップ比が横ばいだが，これは PC クラスタ側の外部のネットワークへの接続口が一つなので，スピードアップ比が向上しなかったと考えられる．

表 5.5: 通信負荷を考慮した計算結果

使用した計算機	処理時間 (sec)	PC1 との処理能力比 (倍)
PC1,WS1	444.64	1.14
PC1,WS1,WS2,WS3	349.07	1.46
PC1,WS1,WS2,WS3,PC クラスタ ×4 台	322.45	1.58
PC1,WS1,WS2,WS3,PC クラスタ ×16 台	339.38	1.50

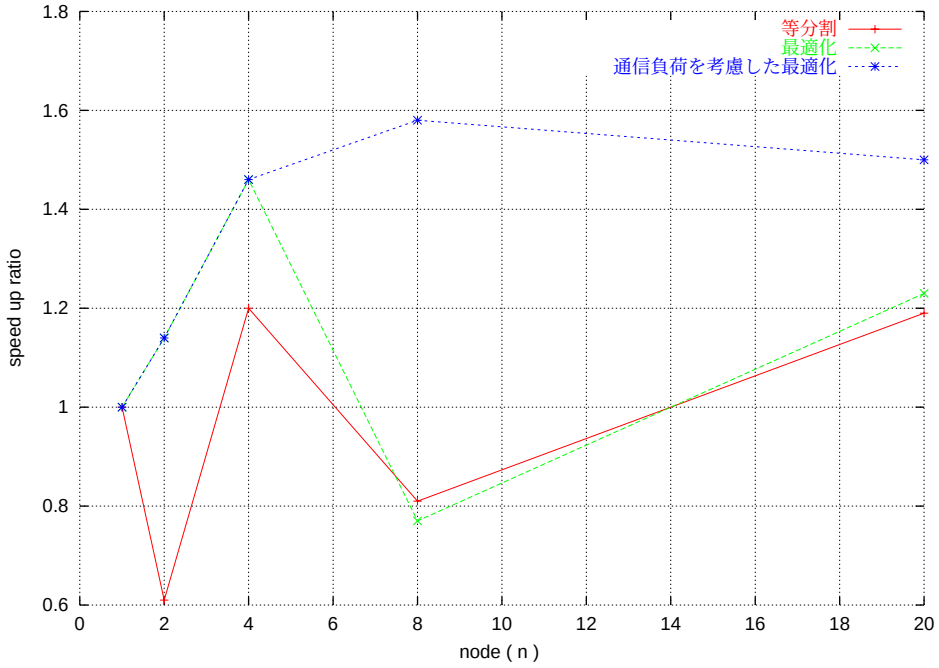

図 5.4: スピードアップ比の比較

負荷分散の予測

ここで，ネットワーク負荷を考慮に入れる場合のネットワークの負荷を予測して，計算実行前に負荷分散の最適化が行えないかを検討してみる．ネットワークの負荷を予測する方法として，‘ping’ コマンドを用いた．

表 5.4 の結果と，‘ping’ コマンドを用いて，各計算機の応答時間を比較してたのを表 5.6 に示す．

表 5.6: 通信負荷と ‘ping’ コマンドを用いた応答時間との比較

使用した計算機	最適化したときの負荷の軽減率 (%)	‘ping’ コマンド応答時間 (msec)
研究室内の各計算機		0.50
PC クラスタ	45	1.02

表 5.6 より，‘ping’ コマンド応答時間が 2 倍になると，負荷の軽減率も約 50%になっている．

この結果より，異機種並列計算で使いたい各計算機を ‘，ping’ コマンドで応答時間を計測し，応答時間から，各計算機の負荷をネットワーク負荷を考慮して軽減できる．

負荷分散の予測の検証

実際に新たに計算機を 1 台追加して，負荷分散の予測をする．今回追加した計算機の処理能力を表 5.7 に示す．

表 5.7: 新たに追加した計算機の処理能力

計算機の種類	処理時間 (sec)	性能比	'ping' コマンド応答時間 (msec)
CS1	1813	1.77	1.21

PC クラスタの計算機を 1 台を表 5.7 の計算機に置き換えて，計 20 台の計算機で異機種並列計算を行う．通信負荷の考慮は，'ping' コマンドの応答時間を参考にし，負荷を 41%に軽減させる．

上記の負荷分散の予測より，異機種並列計算を行った結果を図 5.5 に示す．

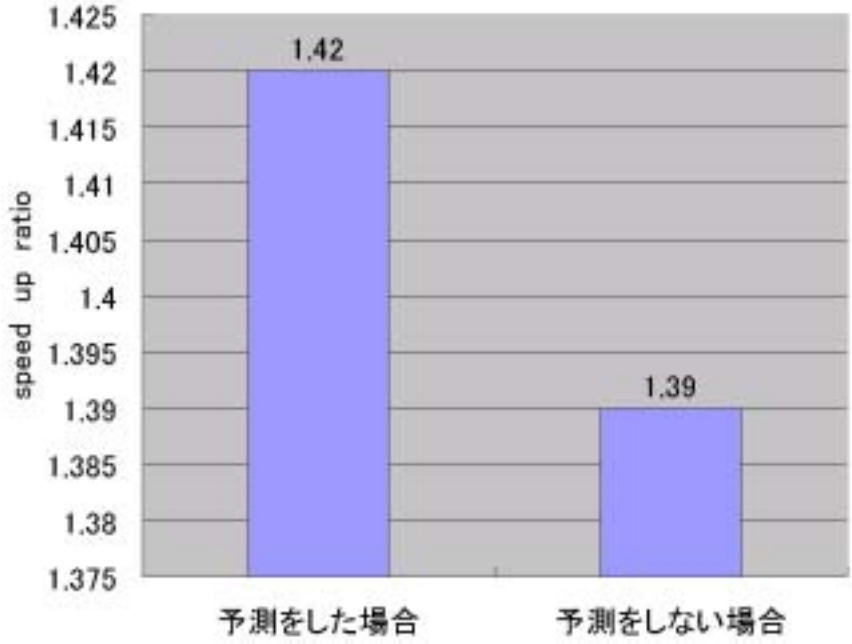

図 5.5: 負荷予測した結果

図 5.5 の予測した場合とは，表 5.7 を参考に計算機の処理能力に応じた負荷分散をし，通信負荷を考慮して計算機の負荷を軽減させてから，異機種並列計算を行った結果である．

予測しない場合とは，計算機の処理能力に応じた負荷分散をし，異機種並列計算を行った結果である．また，それ以外の分割方法も，予測した場合の結果よりスピードアップ比が低かった．図 5.5 より，負荷分散の予測をした場合のスピードアップ比が高いということがわかる．

第6章　まとめ

6.1　異機種並列計算

本研究では，今までの並列計算機で行っていた等分割による並列計算を NPB を用いて異機種の計算機で行った．この結果より，ネットワーク上の資源を利用できるという利点があるが，今までの等分割による異機種並列計算では，各計算機の処理能力の差異によって， 効率良く計算処理時間が短縮できないことが明らかになった．

6.2　負荷分散の最適化

各計算機の処理能力の差異を軽減させるため，あらかじめ各計算機の処理能力を計測しておき，その計測結果をもとに，負荷分散の最適化を行った．この方法では，研究室内の計算機を用いて異機種並列計算をする場合には全体計算のスピードアップ比が向上し有効である．

しかし，研究室外の計算機を用いるときにネットワーク負荷が増加してしまい，全体計算のスピードアップ比が向上しない．

研究室外の計算機も有効利用し，全体の計算処理時間を少しでも短縮したいため，ネットワーク負荷を考慮して研究室外の計算機に分配するタスクを軽減させた結果，異機種並列計算をする場合には全体計算のスピードアップ比が向上した．

あらかじめ，各計算機の処理能力を計測し，また ‘ping’ コマンドの応答時間からネットワーク負荷の予測をして，負荷分散の最適化を行った結果，予測した分割方法で，効率の良い異機種並列計算スピードアップ比となった．

よって，今回の行列ベクトル積の計算では，学内の計算機を用いる場合は，計算前に異機種並列計算に適した負荷分散の最適化ができる．これによって，異機種並列計算の性能を十分に引き出した計算が可能となる．

異機種並列計算は，研究室内の計算機のみを用いた場合には，各計算機の処理能力のみを考慮して，簡単な負荷分散を行うことで比較的良好なスピードアップ比が得られる．研究室外の計算機と併用するときは，ネットワーク負荷を考慮しなければならなず，複雑な負荷分散の最適化を行わなければならない．

6.3 今後の課題

現時点では，負荷分散の簡略化という点から，データの依存関係の無い計算，通信の少ない計算，静的負荷分散を行い，ネットワーク資源の有効利用という観点から，良い結果が得られた．今後，異機種並列計算を応用させるために，

- データの依存関係のある場合の負荷分散の検討．
- ネットワーク上の資源であるため，他のユーザの割り込みによる，計算機負荷の変化，ネットワーク負荷の変化に対する適用．
- 上の問題を解決するために，動的負荷分散があるが，動的負荷分散によるデータ通信量の増加に対する検討．

を検討する必要がある．

謝辞

本論文の作成に際し，有益な助言，御指導，御鞭撻を賜わりました，松澤 照男教授に深く感謝致します.

また，黒川 原佳さん，渡邉 正宏さん，松澤研究室の皆さんに深く感謝いたします。

2003 年 2 月

大岩 博史

参考文献

[1] 今村 俊幸，小出 洋，徳田 伸二，武宮 博 : “トカマクプラズマ計算 (NEXT) における異機種並列計算を結合したハイブリッドコンピューティング”，二本計算機工学会計算工学講演会論文集 Vol.4，p316 (1999)

[2] 石川 裕，佐藤 三久，堀 敦史，住元 真司，原田 浩，高橋 俊行 :“Linux で並列処理をしよう-SCore で作るスーパーコンピュータ-”，P8，共立出版 (2002)

[3] Barry Wilkinson，Michael Allen (飯塚 肇，緑川博子 訳) :“並列プログラミング入門”，pp.22-23，丸善 (2000)

[4] Barry Wilkinson，Michael Allen (飯塚 肇，緑川博子 訳) :“並列プログラミング入門”，pp.28-29，丸善 (2000)

[5] 笹生 健，松岡 聡，建部 修見 :“ヘテロなクラスタ環境における並列 LINPACK の最適化”，情報処理学会論文誌ハイパフォーマンスコンピューティングシステム，pp.49-54，No. 86-9，(2001)