卒業論文

NASパラレルベンチマークによる PCクラスタの性能評価（Ｉ）

氏名　　：　石田　真吾
学籍番号　：　2210020032-9
指導教員　：　山崎　勝弘　教授
提出日　　：　2007年2月19日

立命館大学　理工学部　情報学科

削除:

書式変更：両端揃え

内容梗概

本論文では、NASパラレルベンチマークにより、対象のPCクラスタがどのような処理でどれほどの性能を発揮でき、どのような処理に適しているかというマシンの性能を評価する。

本研究では、本研究室で構築した4ノード4プロセッサの対称型マルチプロセッサ（SMP）クラスタDiploの性能評価をNASパラレルベンチマークのParallel Kernel BenchmarkであるEP、MG、CGおよびISの4種のベンチマークによって行った。また、プロセッサに対する処理の割り当て方を変更し、各々のベンチマークの結果の変化を計測した。それらの実験結果より、本研究室のPCクラスタDiploは、通信がほとんど行われないような並列処理に向いた問題に対しては、プロセッサの台数倍の性能を発揮することができ、プロセッサの割り当てを適切に行うことで、よりよい性能向上を得ることが可能であり、また、ソーティングのような通信が頻繁に行われる問題に不向きであることが分かった。実験環境やプログラムに変更を加えることで、より細やかな性能評価を行うことが課題であると考える。

目次

図目次

表目次

1. はじめに

近年、流体力学シミュレーションによる構造解析や、地球規模･天体規模の現象をシミュレーションしたり、高解像度コンピュータ画像を高速処理するなど、より高性能な計算資源の必要性が高まっている。それらの需要に応える一つの手段として並列コンピューティングを挙げることができる。従来から地球シミュレータなどの大規模なスーパーコンピュータがその分野のほとんどを占めてきたが、近年のパーソナルコンピュータの普及による劇的なマイクロプロセッサの高速化と低コスト化により、複数の PC を高速ネットワークでつないで並列処理システムの構築を行う PC クラスタが広まっており、その性能の向上も著しい。

このように計算機の高性能化が多岐にわたり活発に行われていく中で、自分の所有するシステムがどれほどの処理能力を持っているのか、また高性能計算機のベンダとしても自社のシステムがどのような処理において性能を十分に発揮でき、いったい実行性能値はどれくらいなのかということを知ることが重要となる。そこで性能評価の目安としてベンチマークと呼ばれるテストプログラムが必要とされている。

PC クラスタやスーパーコンピュータなどの並列計算機を評価する、並列処理ベンチマークとして、現在、代表的なものは HPL、姫野ベンチ、NAS パラレルベンチマーク、HPCC などである。その中でも比較的、扱いやすく、また PC クラスタの重要な要素である演算性能と通信性能を総合的に評価できるものとして、NAS パラレルベンチマークを挙げることができる。

NAS パラレルベンチマークは、NASA Ames Research Center で開発された、並列コンピュータのためのベンチマークである。NAS パラレルベンチマークは、5 つの Parallel Kernel Benchmark(EP,MG,CG,FT,IS)と 3 つの Parallel CFD（Computational Fluid Dynamics）Application Benchmark(LU,SP,BT)から構成されており、並列コンピュータの実効性能を知る上で、権威あるベンチマークの１つである。それぞれの問題に対し、問題サイズや反復回数が異なる 5 つのクラス（A,B,C,W(Workstation),S(Sample)）が定義されている。

本研究では、NASパラレルベンチマークを用いて、本研究室のPCクラスタDiploがどのような特性を持っており、どのような処理に向いているのかといった性能評価を行う。NASパラレルベンチマークのParallel Kernel BenchmarkのEP、MG、CGおよびISの4種のベンチマークを用い、クラスA、BおよびCに対しプロセッサ数を1～16まで変化させて実験を行う。また、プロセッサに対する処理の割り当て方を変更し、各々のベンチマークについて実行結果の変化を比較する。2章ではPCクラスタと並列プログラミングの諸要素について説明を示す。3章では、並列処理ベンチマークについて述べ、4章では本研究対象であるNASパラレルベンチマークのEP、MG、CGおよびISについて説明を示す。5章ではEP、MG、CGおよびISの実行結果を示し、またそれぞれについてプロセッサに対する処理の割り当て方を変更した実行結果と比較し、PCクラスタDiploの性能を評価する。

2. PCクラスタと並列プログラミング

書式変更: フォント : 10.5 pt

2.1 SCore型クラスタ[1]

削除:

SCoreとは、新情報処理開発機構(RWCP)にて開発されたクラスタ計算機用超並列プログラム実行環境である。SCore型PCクラスタは、複数種類のネットワークをサポートした高い通信性能を実現しているだけでなく、スーパーコンピュータに匹敵する運用管理機構を備えている。SCoreは、PMv2通信機構、SCore-Dグローバルオペレーティングシステム、MPIライブラリ（MPICH-SCore）、バッチシステム（PBS/SCore）などから構成される。SCoreの構成を図1に示す。

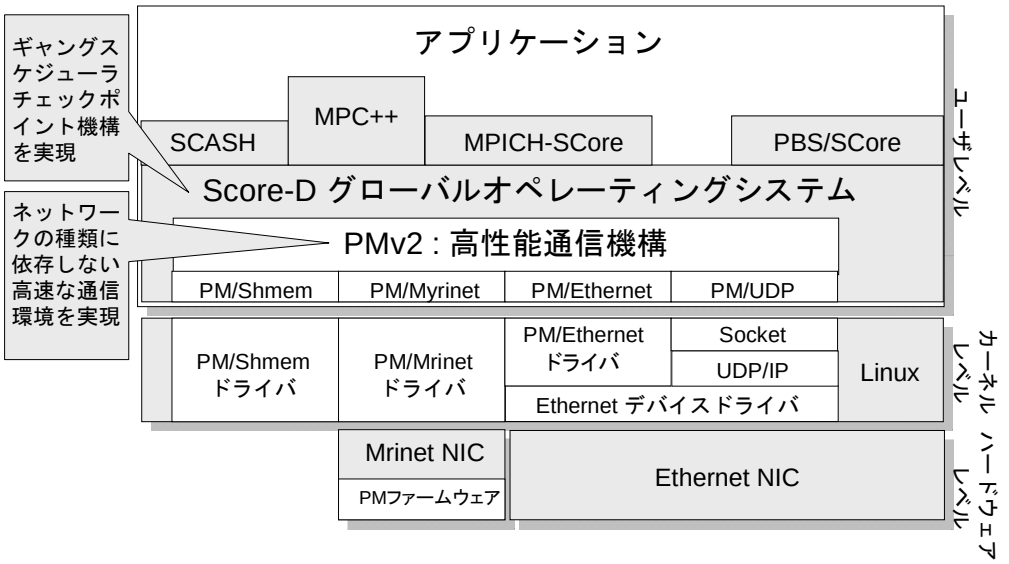


図 1：SCore の構成

PMv2は複数種類のネットワークハードウェアをサポートし、ネットワークの種類に依存しない共通のAPIを上位のSCore-DやMPICH-SCoreに対して提供している。このため、上位のアプリケーションはネットワークの違いを意識することなく実行することができる。現在、PMv2がサポートするネットワークは、Myrinet、Ethernet、UDP/IP、共有メモリである。この中でEthernetについては複数のEthernetのカードを用いてバンド幅の向上を実現するNetwork Trunking機構が実現されている。

SCore-DはPCクラスタ上でのランタイムシステムやジョブスケジューリングの機能のほか、ギャングスケジューラによる複数ジョブでのTSS実行機能を実現している。また、信頼性を高める機構としてチェックポイント機構をも実現している。SCore-Dの提供する

チェックポイント機構ではユーザプログラムへの変更は一切不要で、プログラム起動時にコマンドオプションでチェックポイントを採取する時間間隔を指定する。さらに、1台のノードPCのディスクが故障するとチェックポイントからのリスタートができなくなるため、複数のノードPCにデータを冗長に分割して格納している。

MPICH-SCoreはMPI通信ライブラリの実装の一つであるMPICHをもとにしたライブラリである。MPIについては2章3節で詳しく説明する。

PBS（Potable Batch System）とは、ネットワーク接続された複数のPCをひとつのリソース群として、TSS（タイムシェアリングシステム）のような使い方で有効利用するために開発されたシステムである。

また、SCASHはPMv2を用い、ユーザレベルで実現したソフトウェア分散共有メモリシステムである。

2.2 SMPクラスタ

並列計算機は、図2のような共有メモリ型並列計算機と図3のような分散メモリ型並列計算機の2つに分類される。SMPクラスタは、複数のプロセッサを搭載したSMP構成のPCをさらに複数台ネットワークで接続した並列計算機である。

削除:

削除: <sp>

削除: <sp>

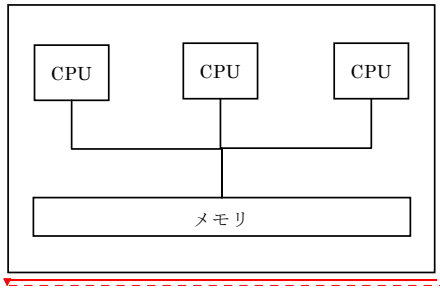


図 2：共有メモリ型並列計算機

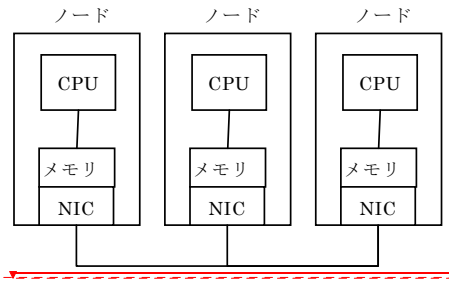
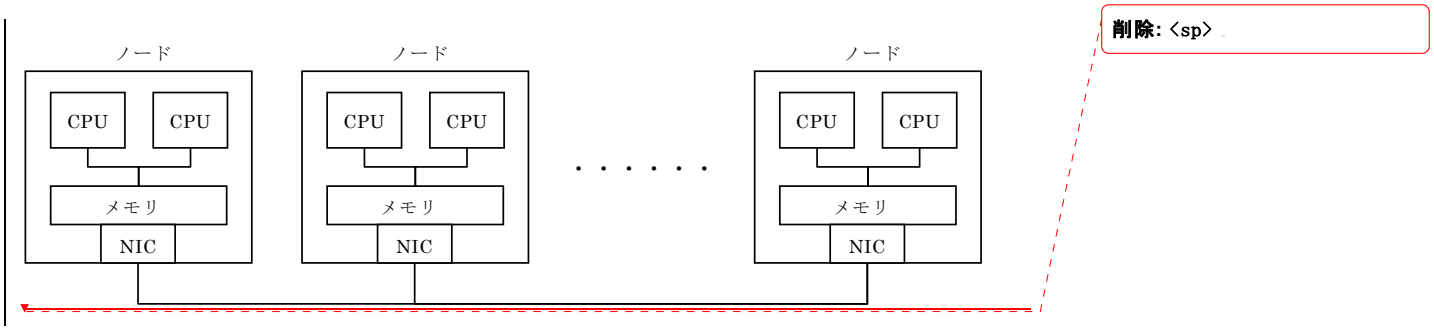


図 3：分散メモリ型並列計算機

SMPクラスタは、図4のように外観は分散メモリモデルと同じであるが，各ノード内は共有メモリモデルと同じ構成になっており，共有メモリモデルと分散メモリモデルとが混在したアーキテクチャである．一般的にPC クラスタのボトルネックはネットワークと言われており，プロセッサ数が同じであれば共有メモリモデルの方が性能面で優れている．SMPクラスタは，分散メモリモデルのように容易に演算ユニットを増やすことができ，ネットワークトラフィックを減らすことができるため，より高速な性能が見込める。



削除: <sp>

図 4：SMP クラスタ

2.3 並列プログラミング

2.3.1 並列計算機のメモリモデル[2]

並列プログラミングを行う際、プログラマが念頭に置いておくべき概念としてメモリモデルが挙げられる。主なモデルとしては、以下の3種類のものがある。

(1) 共有メモリモデル
(2) 分散共有メモリモデル
(3) 分散メモリモデル

以下、それぞれについて述べる。

(1) 共有メモリモデル

複数のプロセッサがメモリバス/スイッチ経由でメモリを共有する形態である（図5）。このアーキテクチュアを有するシステムのことをSMP（Symmetrical Multi Processor）と呼ぶ。この形態の利点はメモリモデルが最も汎用でプログラムが組みやすいことと、並列処理だけでなく、スループットを重視するサーバマシンとしても適しているということである。また、異なるプロセッサが同じメモリ空間にアクセス可能であるため排他制御をする必要とする場合がある。

(2) 分散共有メモリモデル

プロセッサとメモリから構成されるシステムが、互いに接続された形態である（図6）。各プロセッサが全てのメモリとアクセス可能であるため、自由度が高く、大規模なシステムの構築が可能である。逆に1個でもバリア同期の場所を間違えると、タイミング依存で非常にわかりにくいバグを作りこむことになるという欠点も持つ。

(3) 分散メモリモデル

プロセッサとメモリから構成されるシステムが複数個互いに接続された形態で、プロセッサは他のプロセッサの主記憶の読み書きを行うことができず、必ず相手のプロセッサに介

在してもらう必要がある（図7）。利点として、大規模なシステム構築が可能であることと、デッドロックさえ気をつければ、タイミングに依存する嫌なバグは発生することが少ないことが挙げられる。

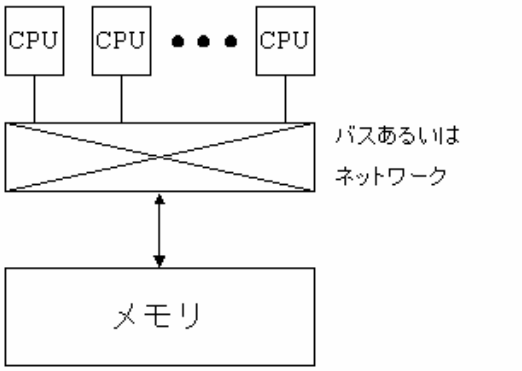


図 5：共有メモリモデル

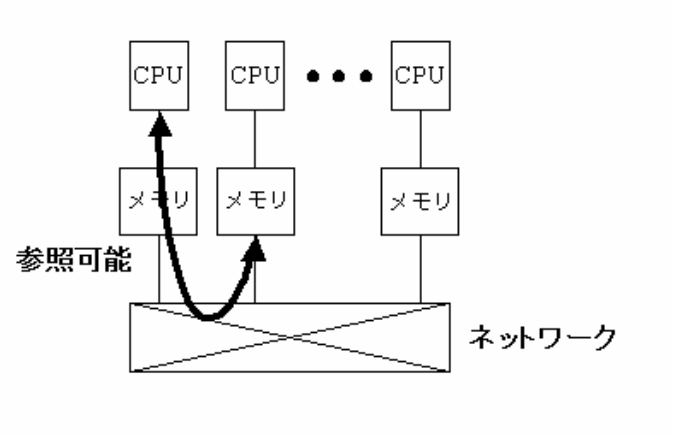


図 6：共有分散メモリモデル

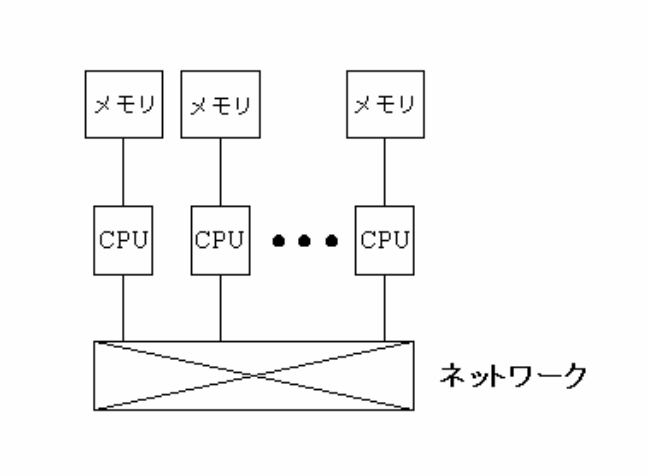


図 7：分散メモリモデル

2.3.2 OpenMP

OpenMPは共有メモリ型並列計算機の並列プログラミングのためのプログラミングモデルであり、ベース言語（Fortran、C、C++）をコンパイラ指示文（directives/pragma）、ライブラリ、環境変数によって拡張したものである。並列実行や同期をプログラマが明示することによって並列化を行なう。また指示文を無視することによって逐次実行が可能なので、逐次版と並列版を同じソースで管理でき、段階的な並列化が可能である。

2.3.3 MPI

MPI（Message Passing Interface）は分散メモリ型並列処理の基本であるメッセージパッシングのライブラリの規格である。ノード間の通信をメッセージの送受信によって実現するライブラリレベルでの並列化実装であるため、C やFortrun など言語を問わず使用できる。

2.4 ハイブリッド並列プログラミング

ハイブリッド並列プログラミングとは、分散メモリ環境と共有メモリ環境が混在した環境において、分散メモリプログラミング環境と共有メモリプログラミング環境の両方の利点を生かしたプログラミングである。MPIは、ノード間での通信を行う並列化実装であるため、ノード内のプロセッサ間の通信を行う場合はオーバーヘッドが大きい。また、OpenMPは、ノード内のプロセッサでメモリ共有を行うためのプログラミングモデルであるため、分散メモリ環境で実現するにはノード間のメモリをソフトウェアで共有する必要があるためボトルネックとなる。また、全てのメモリを1つの共有メモリ空間として扱うため、メモリを有効に活用できない。よって、SMPクラスタ上ではノード間はMPIによっ

削除:

て並列化し、ノード内はOpenMPによってメモリを共有するハイブリッド並列プログラミングを行うことが理想的である。図8にハイブリッド並列プログラミングにおけるMPIとOpenMP の担当範囲を示す。

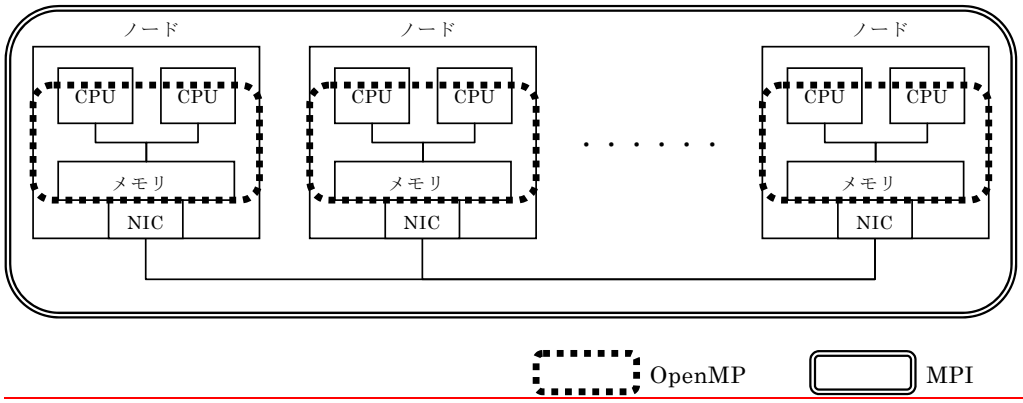


図 8 : MPI と OpenMP の担当範囲

書式変更: 図表番号、中央揃え, 1 行の文字数を指定時に右のインデント幅を自動調整する, 日本語と英字の間隔を自動調整する, 日本語と数字の間隔を自動調整する

書式変更: フォント : 10 pt, 太字（なし）

SMPクラスタ上での実際の実装は、MPIのみの並列化では、各ノードに対してMPIプロセスを2つ割り当て、全てのプロセッサに同じ処理を実行させている。それに対して、ハイブリッドによる並列化では、各ノードに対しMPIプロセスを1つずつ割り当て、ノード内ではOpenMPスレッドを2つ生成して、各プロセッサに割り当て、並列動作を行っている。MPIによる並列実行イメージを図9に、ハイブリッド並列プログラミングによる実行イメージを図10に示す。

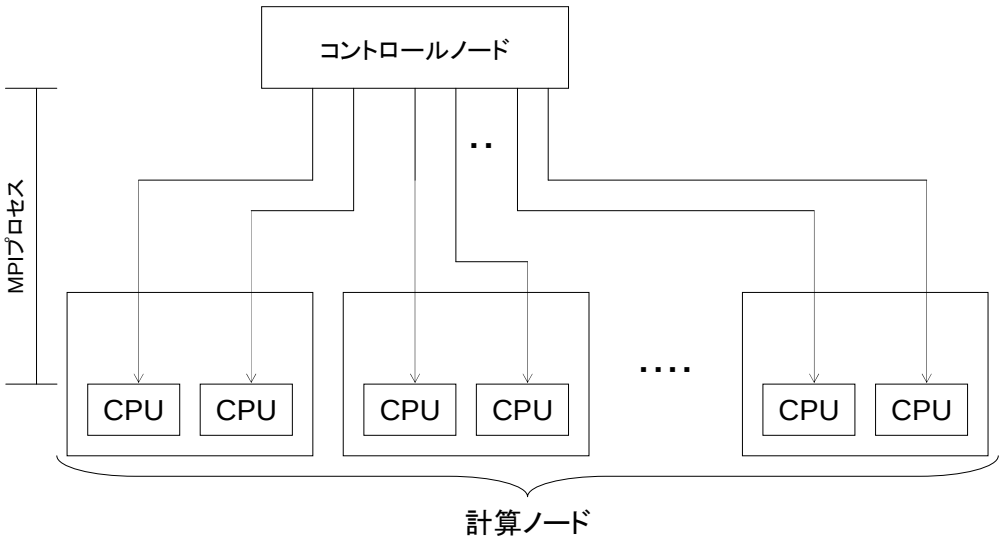


図 9 : MPI による並列実行イメージ

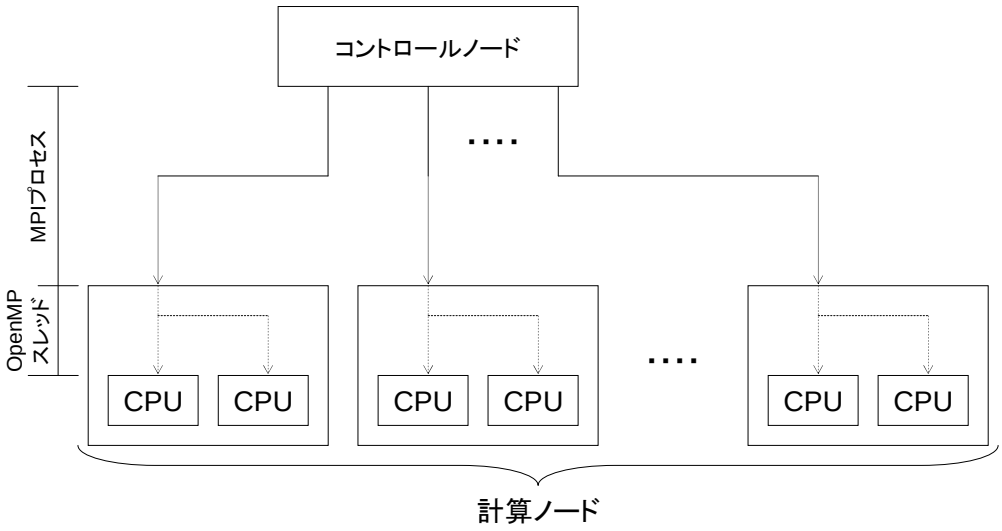


図 10 : ハイブリッド並列プログラミングによる実行イメージ

書式変更: フォント : 10 pt, 太字（なし）

削除: SMPクラスタ上での実際の実装は、MPIのみの並列化では、各ノードに対してMPIプロセスを2つ割り当て、全てのプロセッサに同じ処理を実行させている。それに対して、ハイブリッドによる並列化では、各ノードに対しMPIプロセスを1つずつ割り当て、ノード内ではOpenMPスレッドを2つ生成して、各プロセッサに割り当て，並列動作を行っている．MPIによる並列実行イメージを図5に、ハイブリッド並列プログラミングによる実行イメージを図6に示す。

書式変更: フォント : 10 pt, 太字（なし）

書式変更: フォント : 10 pt, 太字（なし）

書式変更: 図表番号、中央揃え, 1 行の文字数を指定時に右のインデント幅を自動調整する，日本語と英字の間隔を自動調整する，日本語と数字の間隔を自動調整する

削除:

3. 並列処理ベンチマーク

3.1 ベンチマークとは

ベンチマーク（benchmark）という語には、一般的に「基準となるもの」という意味がある。しかし、特にコンピュータ分野においてベンチマークとは、コンピュータのハードウェアやソフトウェアの処理速度を計測する試験問題、およびそれによる性能評価のことを指す。コンピュータは、プログラムによってさまざまな挙動を示す上、その用途も様々であるため、比較の指標を一意に決めることは困難である。そこで、各用途に応じた性能評価の指標を決定し、その尺度に基づいて性能を評価するためのベンチマークが必要となる。ベンチマークの対象となる機能を分類すると次のようになる。

- 演算性能ベンチマーク
- I/O性能ベンチマーク
- グラフィックス関連性能ベンチマーク
- ネットワーク関連性能ベンチマーク
- データベース性能ベンチマーク
- アプリケーション指向ベンチマーク
- 並列/大規模計算ベンチマーク

以下において、PCクラスタの性能を評価するための、並列処理ベンチマークについて述べる。

3.2 NASパラレルベンチマーク

NAS パラレルベンチマーク（NPB）は、NASA Ames Research Centerで開発された、航空関連の流体シミュレーションの並列コンピュータ上での実行性能を評価するためのベンチマークである。NASパラレルベンチマークは、表１に示すような5つの Parallel Kernel Benchmarksと3つのParallel CFD (Computational Fluid Dynamics) Application Benchmarks から構成されている。それぞれの問題に対し、問題サイズや反復回数が異なる５つのクラス（A、B、C、W(Workstation)、S(Sample)）が定義されている。

書式変更: フォント：（英）Century

書式変更: フォント：（英）Century

書式変更: フォント：（英）Century, 10.5 pt

書式変更: 両端揃え

削除: マンデルブロ集合とは、複素関数、*CZZZfnn*+=+21:)(00=Zで表される数列Znがnが無限大まで大きくなったときに発散しないような複素数Cの集合である。マンデルブロ集合を図７に示す。この複素平面上の集合の点の数をX×Yのメッシュに区切り、複素関数を反復計算して計測するプログラムを設計する。[13])(*Zf*

図７：マンデルブロ集合

このとき、数値計算を無限大で扱うことはできない。しかし、条件を付け加えることによって近似することで、計測すること ... [1]

書式変更: 両端揃え

書式変更: フォント：（英）Century, 10.5 pt

削除: PCクラスタgooseで、複素平面上のｘ座標の範囲をブロッ ... [2]

書式変更: フォント：（英）Century, 10.5 pt

削除: スレッド数と実行時の解像度（2500×3700）を設定し ... [3]

書式変更: フォント：（英）Century, 10.5 pt

削除: 手動分割することによりブロック分割とサイクリック分 ... [4]

書式変更: フォント：（英）Century

削除: 3.4 Netperf

書式変更: フォント：（英）Century, 10.5 pt

書式変更: 両端揃え

表 1：NAS パラレルベンチマークの対象問題

Parallel Kernel Benchmarks	
EP	乗算合同法による一様乱数、正規乱数の生成
MG	簡略化されたマルチグリッド法による3次元ポアソン方程式の解法
CG	共役勾配法による正値対称疎行列の最小固有値問題の解法
FT	高速フーリエ変換による3次元偏微分方程式の解法
IS	大規模整数ソート

Parallel CFD Application Benchmarks	
LU	Symmetric SOR によるLU分解
SP	非優位対角なスカラ5重対角方程式の解法
BT	非優位対角な5×5ブロックサイズの三重対角方程式の解法

3.3 HPL（High-Performance Linpack）

Linpack Benchmarkとは、米テネシー大学Jack Dongarra博士が開発した密行列解法ベンチマークテストプログラムで、汎用線形行列計算ライブラリを用いて大規模な連立一次方程式を解くプログラムでる。

HPLは、そのLinpackの実装の一つである。HPLは分散メモリ型並列計算機用のベンチマークソフトウェアであり、ガウス消去法を用いた密行列連立一次方程式の求解における実行時間により性能を評価する。全17種類もの様々なパラメータを計算機の特性に合わせて設定することが可能で、高度に最適化された線形演算ライブラリを組み込むことにより高い性能を得ることができる。

また世界中の高性能コンピュータの上位500位をリストアップしているTOP500 Supercomputer Sitesでの評価基準に採用されており、事実上の世界標準となっている。

3.4 姫野ベンチ

科学技術計算において利用されることが多い処理として、計算流体力学や熱解析、電磁場解析などがあるが、このような処理における非圧縮流体解析コードの性能評価を行うために理化学研究所の情報基盤センター長である姫野龍太郎によって開発されたベンチマークである。このベンチマークは、物理や工学の分野など広い範囲で現れるポアソン方程式を三次元の一般座標系による差分法により離散化し、ヤコビ反復法で近似解を求める場合の主要なループ処理の速度を計るものである。コードは非常に短く簡単にコンパイル・実行できるので、即座に実行性能値を求めることが特徴である。

3.5 HPCC（HPC Challenge Benchmark）

Linpackと同じく、米国Tennessee大学のJack Dongarra博士を中心に開発されたベンチマークである。複数のベンチマークを組み合わせることでシステム全体の性能を評価する目的で7つのテストから構成されている。演算処理の性能のみではなく，I/O性能やネット

削除: 手動分割することによりブロック分割とサイクリック分割のスレッド数が1のときの実行時間に差が見られなくなり、比較が容易にできた。ブロック分割とサイクリック分割とでは、スレッド数が増えるにつれ、サイクリック分割のほうが処理時間が短くなり、速度向上も得られた。これはブロック分割よりもサイクリック分割のほうが負荷均衡が取れているということである。

セクションの最後

4. NASパラレルベンチマーク

4.1 EP

書式変更: 両端揃え

書式変更: フォント : 10.5 pt

ワークの性能なども計測し、システムの総合的なパフォーマンスを計測するため、米国防総省高等研究計画局 （The Defense Advanced Research Projects Agency：DARPA）が資金提供し、米政府後援のスーパーコンピュータ用テストとして開発された。

4. NASパラレルベンチマーク

本研究では、より細やかな性能評価を行うためにParallel Kernel Benchmarksを対象とした。なお、FTについてはFortran90の記述により、本実験環境では動作しなかったため除外する。

4.1 EP

EP（The Embarrassingly Parallel）ベンチマークは、指定されたスキームに従い生成された、多数のガウス分布に従う擬似乱数を用いて、2次元の統計情報を蓄積する。並列計算に適しており、モンテカルロ法を用いた応用プログラムによくみられる。この問題を解く際に転送をほとんど必要としないため、ある意味において本ベンチマークは、システムが持つ浮動小数点演算の最大実効性能を示すものと言える。Fortran77＋MPIで実装されている。

表 2：EP のクラス定義

クラス	乱数生成の個数	反復回数
A	536870912	0
B	2147483648	0
C	8589934592	0

4.2 MG

MG（MultiGrid）ベンチマークは、3次元ポアソン方程式を簡略化したマルチグリッド法で解いている。このベンチマークはプロセス数が増加するにつれ、プロセス間の通信のメッセージサイズは小さくなる一方、通信回数が増えるという特性を持つ。Fortran77＋MPIで実装されている。

表 3：MG のクラス定義

クラス	グリッドのサイズ	反復回数
A	256×256×256	4
B	256×256×256	20
C	512×512×512	20

書式変更: 両端揃え，インデント： 最初の行： 1 字

削除: この章は前章で実験したマンデルブロ集合の計算の条件を変え、PCクラスタraptorで並列化を行い、どのような変化があるのかを調べる。

マンデルブロ集合の複素関数、*CZZZfnn*+=+21:)(00=*Z*であらわされる数列を(-2.0実部<0.5)、(-2.2≤虚部<1.5)の範囲で複素平面をX×Yのメッシュに区切り、X×Y 個の点について上の複素関数の反復計算を行う。|Zn|>2で発散し、n>50のとき収束するとして、収束時の座標点がマンデルブロ集合の要素となる。並列化の手法は、複素平面上のX座標の範囲をブロック分割とサイクリック分割に分けて実行し、マンデルブロ集合に属する座標点の合計数をcountに格納する。≤

今回は以上の条件を以下のように変更する。

<#>(1) 解像度（2500×3700→25000×37000、250×370）

<#>(2) 分割方法（範囲内の実数 ... [5]

削除: 今回の並列化は(2)以外は前回と同じ方法で実行できるが、(2)は前回の方法とは異なる。(2)以外は複素平面上の x 座標の範囲をブロック分割とサイクリック分割で並列化し実行するが ... [6]

削除: 1 EP

書式変更: フォント：（英）Century, 10.5 pt

書式変更: インデント： 最初の行： 1 字

4.3 CG

CG（Conjugate Gradient）ベンチマークは、大規模で正値対称な疎行列の、最小固有値の近似値を共役勾配法を用いて解く。このカーネルは非構造格子を用いたアプリケーションでよく見られる。このベンチマークでは一定の通信が絶えず行われるため、ネットワーク性能がそのまま結果に反映されるという特徴を持つ。Fortran77＋MPIで実装されている。

表 4：CG のクラス定義

クラス	行列のサイズ	反復回数
A	14000	15
B	75000	75
C	150000	75

4.4 IS

IS（Integer Sort）ベンチマークは、パーティクル法を使用したアプリケーションにおいて重要なソートを評価するものである。この種類のアプリケーションは、物理においてパーティクルをあるセルに割り当てて､パーティクルがセルから流れ出るかどうかを見る、particle-in-cell法のアプリケーションに 似ており、ソートは、パーティクルを再び適切なセルに割り当てる際に使用される。C＋MPIで実装されている。

表 5：IS のクラス定義

クラス	乱数生成の個数	反復回数
A	8388608	10
B	33554432	10
C	134217728	10

削除: (1)~(5)のブロック分割とサイクリック分割のそれぞれの速度向上比と前回の実験結果の比較を行った。
(1) 解像度変更
スレッド数と実行時の解像度（2500×3700）､解像度（25000×37000）、（250×370）を設定し、それぞれのブロック分割とサイクリック分割の速度向上比の比較を図１１に示す。スレッド数8でブロック分割は4倍、2.4倍、サイクリック分割では7.3倍、4.6倍の速度向上が得られた。この結果では前回の結果と比較すると、解像度（25000×37000）はブロック分割、サイクリック分割と同様にあまり差異は見られなかったが、解像度（250×370）はスレッド数4 ... [7]

削除: 01234567802468 10スレッド数速度向上比横分割ブロック横分割サイクリック縦分割ブロック縦分割サイクリック ... [8]

削除:
(4)､(5)の実験では条件の変更による速度向上の変化はほぼ見られなかったのに対し、(1)はブロ ... [9]

書式変更: 両端揃え，インデント： 最初の行： 1 字

書式変更: フォント：（英）Century, 10.5 pt

削除: これまでの実験結果から並列化の負荷分散を少なくするためには、より細かく分割する方法が良いと考えられる。その ... [10]

書式変更: フォント：（英）Century, 10.5 pt

書式変更: 両端揃え

5. NASパラレルベンチマークによる性能評価

5.1 実験環境

実験環境には、Dual-Core Intel Xeon3GHz を2台搭載したSMPノード4台をGigabit Etherで接続した、4×4(4ノード4プロセッサ)の計16プロセッサのSMPクラスタ「Diplo」を用いる。図11にその構成を示す。

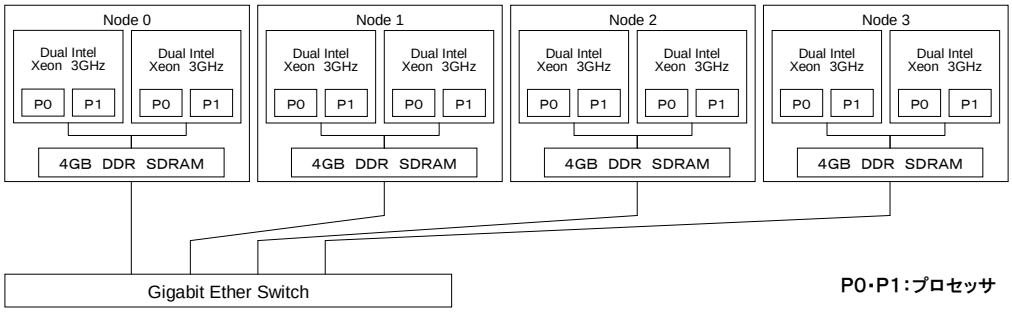


図 11：SMP クラスタ Diplo の構成

オペレーティングシステムはCentOS 4.4、MPIコンパイラとしてMPICH 1.2.7、CおよびFortran77コンパイラとしてGCC 3.4.5を用いる。

5.2 実験内容

NASパラレルベンチマークのEP、MG、CGおよびISの4種類のベンチマークを用いて、PCクラスタDiploの性能を計測する。

EP、MG、CGおよびISそれぞれのベンチマークにおいて、クラスA、BおよびCに対しプロセッサ数を1～16まで変化させて実験を行う。ここで、実験環境であるPCクラスタDiploは4×4のSMP構成のクラスタであるので、プロセッサ数2、4および8のときに処理をノード内とノード間に割り当てる二通りの割り当て方を考えることができる。例えば4プロセッサに処理を割り当てる場合には、ノード内プロセッサ割り当てでは図12のように、ノード間プロセッサ割り当てでは図13のように処理を割り当てる。

実験では、まずノード内プロセッサ割り当ての実験を行い、その後にノード間プロセッサ割り当てに処理の割り当て方を変更して実験を行い、割り当ての変更前後での実験結果を比較する。

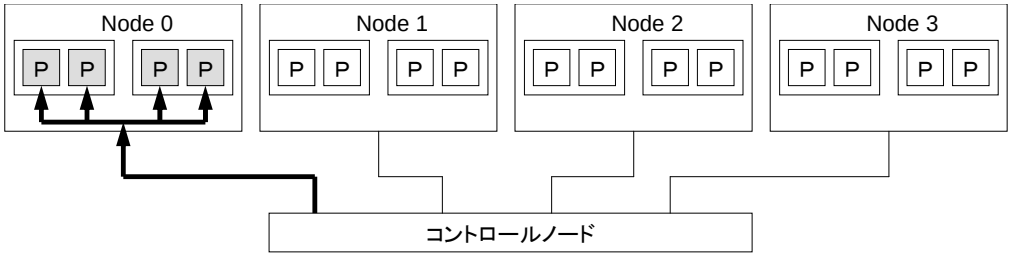


図 12：ノード内プロセッサ割り当て

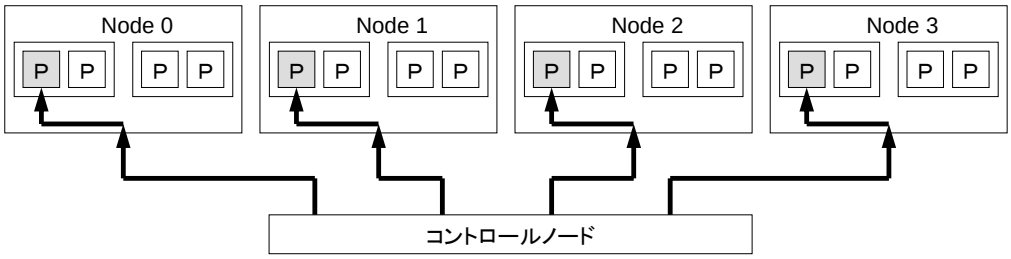


図 13：ノード間プロセッサ割り当て

5.3 実験結果

5.3.1 ノード内プロセッサ割り当て

(1) EP

プロセッサ数16のとき、クラスAとクラスBでは15.7倍、クラスCでは15.8倍とすべてのクラスにおいて、きわめて理想に近い速度向上が得られた。

表 6：EP の実行時間

単位：秒

プロセッサ数	1	2	4	8	16
クラス A	62.39	31.41	15.75	7.97	4.04
クラス B	251.33	125.59	62.94	31.63	15.94
クラス C	997.35	502.28	251.77	125.82	63.11

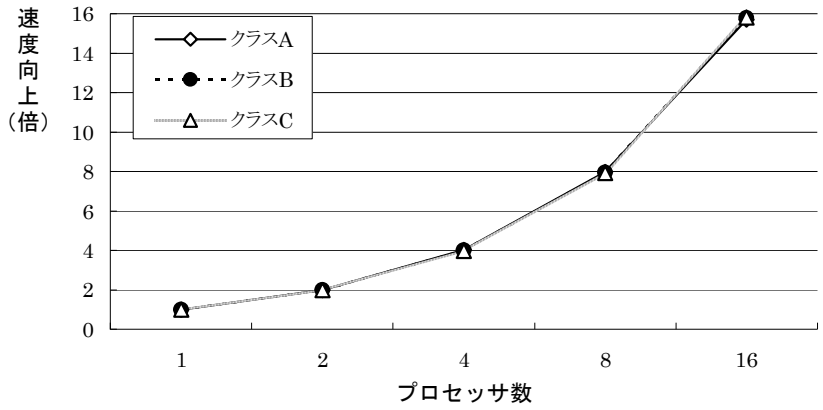


図 14：EP の速度向上

(2)　MG

プロセッサ数16のとき、クラスAとクラスＢでは5.5倍、クラスCでは6.2倍の速度向上が得られた。またクラスCではプロセッサ数2と4で結果が変化せず、プロセッサ数１では実行できなかった。

表 7：MG の実行時間

単位：秒

プロセッサ数	1	2	4	8	16
クラス A	6.46	4.08	3.64	2.01	1.17
クラス B	30.23	19.54	15.77	9.36	5.51
クラス C		138.29	138.53	78.40	44.87

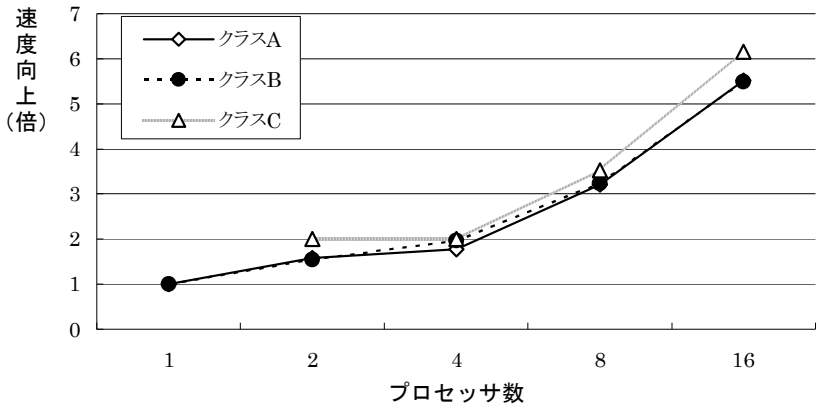


図 15：MG の速度向上

(3)　　CG

プロセッサ数16のとき、クラスAでは3.4倍、クラスＢでは3.7倍、クラスCでは5.3倍とクラスのランクを上げるにつれて、よりよい速度向上が得られた。

表 8：CG の実行時間

単位：秒

プロセッサ数	1	2	4	8	16
クラス A	5.36	3.24	2.55	1.89	1.38
クラス B	209.16	119.43	97.01	70.84	57.04
クラス C	696.66	317.67	242.40	172.69	126.09

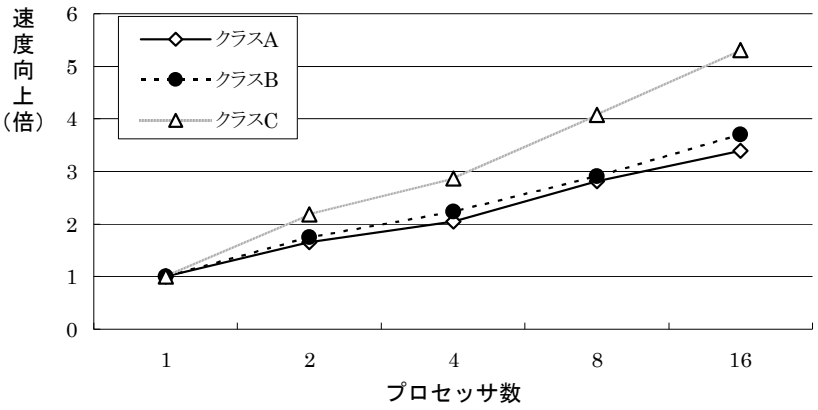


図 16：CG の速度向上

(4)　　IS

すべてのクラスでプロセッサ数2のときに性能が低下し、プロセッサ数16のときにおいても、クラスAでは1.4倍、クラスＢとクラスCでは1.5倍の速度向上に止まった。

表 9：IS の実行時間

単位：秒

プロセッサ数	1	2	4	8	16
クラス A	2.44	3.04	2.36	2.02	1.70
クラス B	11.19	12.62	9.46	8.38	7.13
クラス C	46.86	52.02	38.67	33.27	30.44

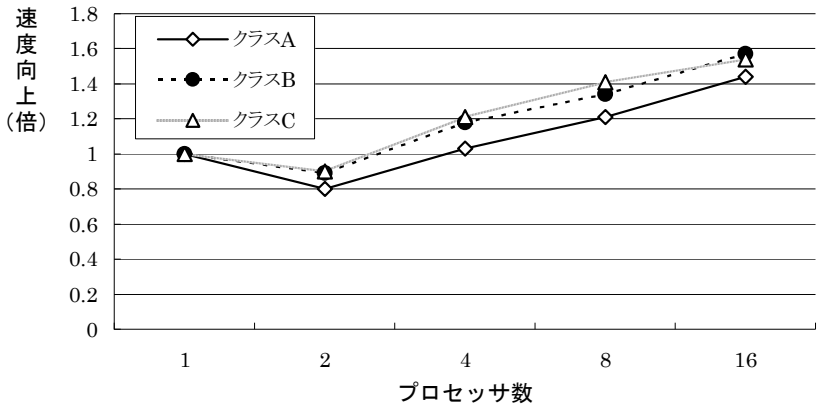


図 17：IS の速度向上

5.3.2 ノード間プロセッサ割り当て

プロセッサに対する処理の割り当て方をノード内からノード間に変更した実験結果には、クラス名の右側に＊印を付加して表した。

(5) EP

すべてのクラスにおいて変化は誤差の範囲内に収まり、プロセッサの割り当て方を変更した後でも、きわめて理想に近い速度向上が得られた。

表 10：プロセッサ割り当て変更後の EP の実行時間

単位：秒

プロセッサ数	1	2	4	8	16
クラス A*	62.15	31.27	15.72	7.87	4.15
クラス B*	249.64	125.69	62.85	31.52	15.82
クラス C*	998.82	501.93	251.66	125.76	63.10

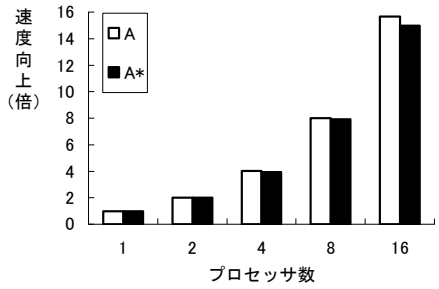


図 18：EP の速度向上の比較（クラス A）

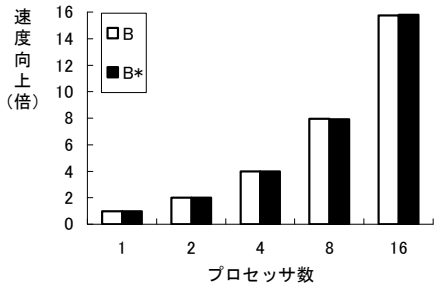


図 19：EP の速度向上の比較（クラス B）

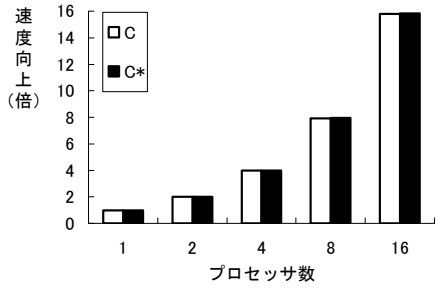


図 20：EP の速度向上の比較（クラス C）

(6)　MG

プロセッサ数4と8のとき、すべてのクラスにおいてかなりの性能向上が得られた。また、実験(2)のクラスCで速度向上が思うように得られなかったプロセッサ数4においては、78%の性能向上が得られた。

表 11：プロセッサ割り当て変更後の MG の実行時間

単位：秒

プロセッサ数	1	2	4	8	16
クラス A*	6.41	4.08	2.29	1.41	1.17
クラス B*	30.15	19.03	10.84	7.32	5.40
クラス C*		139.71	77.89	59.93	44.87

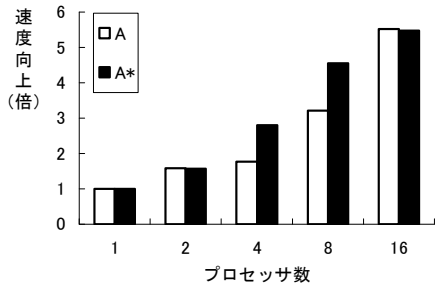


図 21：MG の速度向上の比較（クラス A）

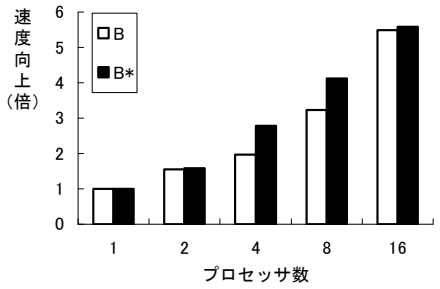


図 22：MG の速度向上の比較（クラス B）

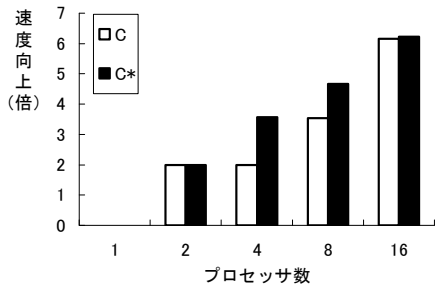


図 23：MG の速度向上の比較（クラス C）

(7)　CG

実験(6)と同様にプロセッサ数4と8のとき、すべてのクラスにおいてかなりの性能向上が得られた。特に、プロセッサ数8のときクラスBでは22%、クラスCでは37%の性能向上が得られ、プロセッサ数16相当の性能を達成している。

表 12：プロセッサ割り当て変更後の CG の実行時間

単位：秒

プロセッサ数	1	2	4	8	16
クラス A*	5.36	3.23	2.23	1.68	1.38
クラス B*	209.16	119.43	76.91	58.16	57.04
クラス C*	696.66	317.61	194.73	126.47	126.09

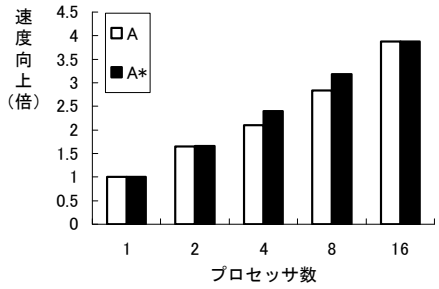


図 24：CG の速度向上の比較（クラス A）

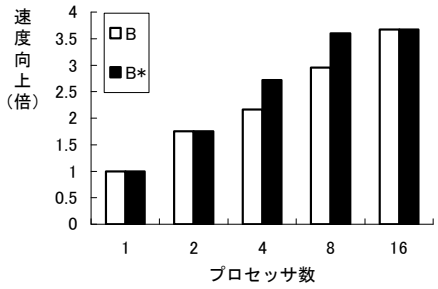


図 25：CG の速度向上の比較（クラス B）

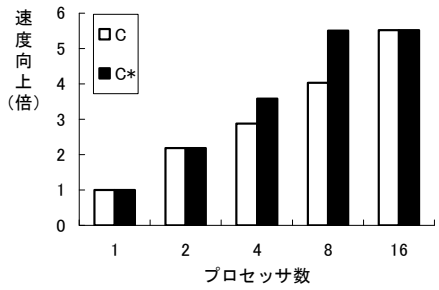


図 26：CG の速度向上の比較（クラス C）

(8)　IS

プロセッサ数4と8のとき、すべてのクラスにおいて10%程度の性能低下が見られる。また、クラスのランクが上がるほど性能低下が著しくなる傾向が見られる。

表 13：プロセッサ割り当て変更後の IS の実行時間

単位：秒

プロセッサ数	1	2	4	8	16
クラス A*	2.39	3.05	2.62	2.11	1.71
クラス B*	11.02	12.62	10.83	9.17	7.36
クラス C*	47.34	52.01	46.23	36.21	30.45

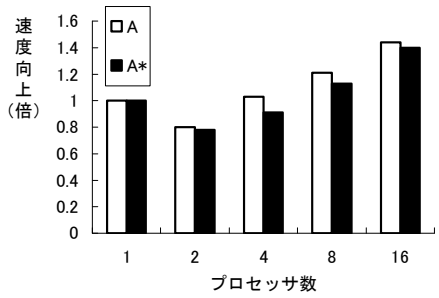


図 27：IS の速度向上の比較（クラス A）

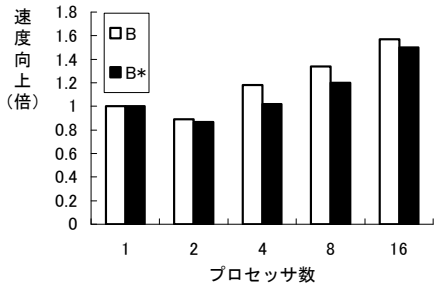


図 28：IS の速度向上の比較（クラス B）

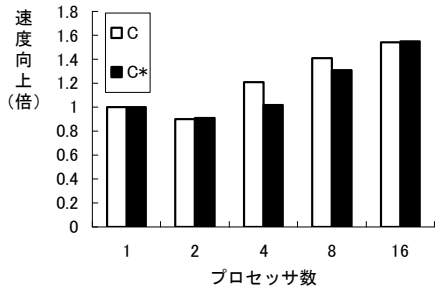


図 29：IS の速度向上の比較（クラス C）

5.4　評価

(1)～(8)の実験結果から以下の3点についてPCクラスタDiploの性能評価を示す。

（ｉ）　台数効果

実験(1)と(5)から、通信がほとんど行われないような並列処理に向いた問題に対しては、プロセッサの台数倍の速度向上を得ることが可能である。

（ii）　プロセッサ割り当て

実験(2)(3)と(6)(7)から、プロセッサに対する処理の割り当て方を適切に行うことにより、よりよい性能を得ることが可能である。

この理由として第一にメモリ領域の差が考えられる。DiploはSMPクラスタであるため、

削除: ブロック分割、サイクリック分割の並列化は実部の範囲を1/2500に分割し並列化し実行した。格子状サイクリック分割の並列化は実部の範囲を1/2500に、虚部の範囲を1/3700に分割し、2500×3700個の格子にして並列化を実行した。図１６に格子状サイクリック分割のモデルを示す。

図１６：格子状サイクリック分割のモデル

書式変更: フォント：（英）Century

各ノード内では複数のプロセッサが一つのメモリに接続されており、また、CG、MGはMPIで実装されているため、ノード内でメモリ領域は処理を割り当てられたプロセッサ台数で等分割して確保される。このため、ノード間に対して処理を割り当てることでより多くのメモリ領域を確保できる。

第二の理由としてシステムコール呼び出しの待ち時間の差が考えられる。これもまた、DiploはSMPクラスタであるためであり、各ノード内では処理を割り当てられた各々のプロセッサがノードのOSに対してシステムコール呼び出しを行うため、ノード内で複数のプロセッサに処理が割り当てられた場合に待ち時間が生じる。このため、ノード間に対して処理を割り当てることでシステムコール呼び出しの待ち時間を減少させることができる。

（iii）　並列効果に対する考察

実験(4)から、プロセッサ間で通信が頻繁に行われるような問題に対しては、思うように性能向上を得られないばかりか、通信の遅延により逆に性能低下を招いてしまうことさえあることが分かる。また、実験(8)から、(ii)のようにプロセッサに対する処理の割り当てを変更しても、ノード内とノード間の通信速度の差によって、かえって性能が低下してしまうことが分かる。

削除:
スレッド数と実行時の解像度（2500×3700）を設定し、ブロック分割とサイクリック分割、格子状サイクリック分割でのそれぞれの速度向上の比較を図１７に示す。スレッド数16でブロック分割では8.3倍、サイクリック分割では15.9倍、格子状サイクリック分割では15.8倍の速度向上が得られた。

024681012141618024681012141618 20スレッド数速度向上比ブロックサイクリック格子状

図１７:格子状サイクリック分割による速度向上の比較

6. おわりに

本研究では、NASパラレルベンチマークのParallel Kernel Benchmarksの中からEP、MG、CGおよびISを用いて、4×4の16プロセッサSMPクラスタDiploの性能評価を行った。最初にノード内プロセッサ割り当ての実験を行い、EPでは15.8倍、MGでは6.2倍、CGでは5.5倍、ISでは1.6倍の速度向上を得ることができた。その後にノード間プロセッサ割り当てに処理の割り当て方を変更して実験を行い、割り当ての変更前後で、MGでは最大78%、CGでは最大37%の性能向上を得ることができた。

実験結果より、Diploは通信がほとんど行われないような並列処理に向いた問題に対しては、プロセッサの台数倍の性能を発揮することができ、プロセッサに対する処理の割り当てを適切に行うことで、よりよい性能向上を得ることが可能であることが分かる。また、ソーティングのようなプロセッサ間で通信が頻繁に行われる問題には不向きであることが分かった。

今後の課題として、実験環境にプログラム実行環境であるSCoreを加えた上での性能評価、性能比較を行うことが挙げられる。また、性能をうまく発揮できなかったISをハイブリッド並列プログラミングに書き換えることにより、性能向上を図ることなどが考えられる。

削除: SCore型PCクラスタでのOpenMPによる並列プログラミングを行ってきた。本研究で行ってきたマンデルブロ集合の様々な角度からの並列化計算を実行することにより、SCoreがPCクラスタの仮想共有メモリシステムを構築し、OpenMPを利用可能にしたことや、OpenMPが幅広いプログラミング言語の種類を網羅していることから、SCoreの仕組みやOpenMPの利用法、移植性の高さを学ぶことができた。また、様々な条件から計算したことで並列化による負荷やオーバーヘッドも実験内容から知ることができた。

削除: 今現在、OpenMPという移植性の高い言語ライブラリができたことで容易に並列化が可能になったが、先ほどの章で述べたデータ毎の最適なプログラミングについてはプログラマが工夫しなくてはならない。これからの課題として、プログラムが最適な速度向上が得られるOpenMPに変わる言語ライブラリの開発が必要だと思っている。これから宇宙や遺伝子などの研究が進み、天文学的な数値の計算を扱ったりする際にこのようなライブラリを作ることが、それらの研究の発展に繋がるのではないかと考える。

謝辞

本研究の機会を与えて下さり、数々の貴重な助言、ご指導を頂きました山崎勝弘教授に深く感謝致します。

また､本研究にあたり､貴重な意見やご指導を頂きました本研究室のマスターの中谷氏、Duy氏に心より感謝致します。そして、励ましの言葉や貴重な意見を頂きました本研究室の皆様、先輩方に心より感謝いたします。

最後に，本研究の共同研究者である松田氏に心より感謝致します。

参考文献

[1]石川裕、住元真司：Linuxで並列処理をしようーSCoreを用いたクラスタ構築ー、共立出版、2002.
[2]湯浅太一、安村通晃、中田登志之：はじめての並列プログラミング、共立出版、1999.
[3]NAS Parellel Benchmarks：http://www.nas.nasa.gov/Resources/Software/npb.html.
[4]高畑 泰祐、廣安 知之、三木 光範：ベンチマークとは、2005、
http://mikilab.doshisha.ac.jp/dia/research/report/2005/0811/003/report20050811003.html.
[5]超並列計算研究会：PCクラスタ超入門、2000、
http://mikilab.doshisha.ac.jp/dia/smpp/cluster2000/.
[6]多田修司：PCクラスタ上でのOpenMPによるマンデルブロ集合の並列化、立命館大学理工学部情報学科卒業論文、2005.
[7]黒川耕平：OpenMPによる並列プログラミング２、立命館大学理工学部情報学科卒業論文、2000.
[8]池上広済：ハイブリッド並列プログラミングによるMPEG2エンコーダの高速化、立命館大学理工学部情報学科修士論文、2006.

ページ 9: [1] 削除　1C　2007/01/29 6:58:00

マンデルブロ集合とは、複素関数、$CZZZf_{nn}+=_{+21}:)(0_0=Z$で表される数列Znがnが無限大まで大きくなったときに発散しないような複素数Cの集合である。マンデルブロ集合を図７に示す。この複素平面上の集合の点の数をX×Yのメッシュに区切り、複素関数を反復計算して計測するプログラムを設計する。[13])(*Zf*

図７：マンデルブロ集合

このとき、数値計算を無限大で扱うことはできない。しかし、条件を付け加えることによって近似することで、計測することができる。ここであるｋに対して|Zk|>2ならばZn→∞となることが証明されているので、これを判定条件にする。これは各Cにつき、有限のnに対して0ｋ≤≤nの範囲で順次計算し、あるｋにおいて|Zk|≤2となった場合Cに属すると判定する。また|Zk|>2ならば発散するのでCに属さないと判定する。また、有限のnで計算を打ち切るため、n>50のとき収束するとして設定する。このnはこの集合の点の数を決定するものでnの限界値が大きくなるほど得られるマンデルブロ集合は近似的なものになる。

この条件で得られた図形の計算を複素平面上(-2.0≤実部<0.5)、(-2.2≤虚部<1.5)の範囲で行う。並列化の手法は、複素平面上のX座標の範囲を以下の方法でブロック分割とサイクリック分割に分けて実行する。

Ｚをｘ、ｙで表すと、

$yixZ+=$

同じくCをA、Bで表すと、

$BiAC+=$

これを複素関数に代入し、複素関数を実部、虚部に分けて表す。

$AyxZ_x+-=_{22}$

$BxyZ_y+=2$

実部と虚部の始点と終点との距離を計算し、それを定めた解像度によって分割する。

そしてそのメッシュ部分のｘ、ｙの値をループで上の式に代入する。

上の式の解のうち、|Zn|≤2の条件を満たしたものだけをカウントする。

3.2 NASパラレルベンチマーク

ページ 9: [2] 削除　1C　2007/01/29 6:58:00

PCクラスタgooseで、複素平面上のｘ座標の範囲をブロック分割とサイクリック分割で並列化し実行する。

ブロック分割ではスレッド数と同じだけ分割して処理するのに対し、サイクリック分割では1/2500に分割し、それを各スレッドに1つずつ順番に分配して処理する方法をとっている。ブロック分割とサイクリック分割のモデルを図８に示す。

ブロック分割　サイクリック分割

図８：ブロック、サイクリックの分割方法

3.3 ScaLAPACK

ページ 9: [3] 削除	1C	2007/01/29 6:58:00

スレッド数と実行時の解像度（2500×3700）を設定し、ブロック分割とサイクリック分割のそれぞれの速度向上の比較を図９に示す。スレッド数16でブロック分割では8.3倍、サイクリック分割では15.8倍の速度向上が得られた。

024681012141618024681012141618 20スレッド数速度向上比ブロック分割サイクリック分割

図９：速度向上の比較

3.4 姫野ベンチ

ページ 9: [4] 削除	1C	2007/01/29 6:58:00

手動分割することによりブロック分割とサイクリック分割のスレッド数が1のときの実行時間に差が見られなくなり、比較が容易にできた。ブロック分割とサイクリック分割とでは、スレッド数が増えるにつれ、サイクリック分割のほうが処理時間が短くなり、速度向上も得られた。これはブロック分割よりもサイクリック分割のほうが負荷均衡が取れているということである。

ページ 12: [5] 削除	1C	2007/01/29 6:58:00

この章は前章で実験したマンデルブロ集合の計算の条件を変え、PCクラスタraptorで並列化を行い、どのような変化があるのかを調べる。

マンデルブロ集合の複素関数、$CZZZf_{nn}+=_{+21}:)(0_0=Z$であらわされる数列を(-2.0実部<0.5)、(-2.2≤虚部<1.5)の範囲で複素平面をX×Yのメッシュに区切り、X×Y 個の点について上の複素関数の反復計算を行う。|Zn|>2で発散し、n>50のとき収束するとして、収束時の座標点がマンデルブロ集合の要素となる。並列化の手法は、複素平面上のX座標の範囲をブロック分割とサイクリック分割に分けて実行し、マンデルブロ集合に属する座標点の合計数をcountに格納する。 ≤

今回は以上の条件を以下のように変更する。

(1) 解像度（2500×3700→25000×37000、250×370）

(2) 分割方法（範囲内の実数による分割を虚数による分割へ）

(3) 実部、虚部の範囲（-2.0≤実部<0.5、-2.2≤虚部<1.5→-5.0≤実部<5.5、-5.2≤虚部<5.5）

(4) 発散条件（|Zn|>2→|Zn|>3）

(5) 収束条件（n>50→n>100）

4.2 MG

ページ 12: [6] 削除	1C	2007/01/29 6:58:00

今回の並列化は(2)以外は前回と同じ方法で実行できるが、(2)は前回の方法とは異なる。(2)以外は複素平面上のｘ座標の範囲をブロック分割とサイクリック分割で並列化し実行するが、(2)はｙ座標の範囲をそれぞれの方法で分割し並列化する。
ブロック分割はスレッド数と同じだけ分割して処理するのに対し、サイクリック分割は実数（虚数）の距離の解像度によって決定した数だけ分割しそれを各スレッドに1つずつ順番に分配して処理する方法をとっている。スレッドがN個のときにブロック分割とサイクリック分割をｙ軸方向で行った方法を図１０に示す。
ブロック分割 サイクリック分割
図１０：y軸方向での分割方法

ページ 13: [7] 削除 1C 2007/01/29 6:58:00

(1)~(5)のブロック分割とサイクリック分割のそれぞれの速度向上比と前回の実験結果の比較を行った。
(1) 解像度変更
スレッド数と実行時の解像度（2500×3700）、解像度（25000×37000）、（250×370）を設定し、それぞれのブロック分割とサイクリック分割の速度向上比の比較を図１１に示す。スレッド数8でブロック分割は4倍、2.4倍、サイクリック分割では7.3倍、4.6倍の速度向上が得られた。この結果では前回の結果と比較すると、解像度（25000×37000）はブロック分割、サイクリック分割と同様にあまり差異は見られなかったが、解像度（250×370）はスレッド数４、８でブロック分割、サイクリック分割両方に差異が見られた。この理由として、解像度（250×370）の情報量が少なかったことによるオーバーヘッドからくる速度の低下だと考えられる。

0123456780246810スレッド数速度向上比25k×37kブロック25k×37kサイクリック2.5k×3.7kブロック2.5k×3.7kサイクリック250×370ブロック250×370サイクリック
図１１：解像度変更による速度向上の比較
(2) 分割方法の変更
この実験では図２で示すとおり、虚数方向からブロック分割、サイクリック分割したものである。スレッド数と実行時の解像度（2500×3700）を設定し、それぞれのブロック分割とサイクリック分割の速度向上比の比較を図１２に示す。スレッド数8でサイクリック分割では7.1倍、ブロック分割では2.6倍の速度向上が得られた。この結果は前回の結果と比較するとサイクリック分割に差異は見られなかったが、スレッド数４、８ではブロック分割に差異が見られた。ブロック分割とサイクリック分割の結果に違いが見られるのは、各プロセッサに仕

事を与えたときの情報量の大小によって、各プロセッサが処理を終える時間に違いが出るためである。差が出たのはマンデルブロ集合の形が横長だからだと考えられる。

ページ 13: [8] 削除　1C　2007/01/29 6:58:00

0123456780246810スレッド数速度向上比横分割ブロック横分割サイクリック縦分割ブロック縦分割サイクリック

図１２：分割方法による速度向上の比較

(3) 実部、虚部の範囲の変更

ここでは、前回の実験よりも実部と虚部の範囲を拡大して比較した。スレッド数と実行時の解像度（2500×3700）を設定し、それぞれのブロック分割とサイクリック分割の速度向上比の比較を図１３に示す。スレッド数8でサイクリック分割では7.0倍、ブロック分割では2.3倍の速度向上が得られた。この結果では前回の結果と比較するとサイクリック分割に差異は見られなかったが、スレッド数４、８では②の結果と同様にブロック分割に差異が見られた。理由は広範囲になったため、-2.0<r<0.5、-1.0<i<1.0に分布するマンデルブロ集合から考えて、処理が多くなるプロセッサと少なくなるプロセッサに分かれたからと考えられる。

0123456780246810スレッド数速度向上比広範囲ブロック広範囲サイクリック狭範囲ブロック狭範囲サイクリック

図１３：範囲変更による速度向上の比較

(4) 発散条件の変更

ここでは、複素関数の発散条件を|Zn|>2から|Zn|>3に変更して実行した。スレッド数と実行時の解像度（2500×3700）を設定し、それぞれのブロック分割とサイクリック分割の速度向上比の比較を図１４に示す。スレッド数8でブロック分割は4.１倍、サイクリック分割では7.2倍の速度向上が得られた。この結果は前回の結果と比較しても差異は見られなかった。)(*Zf* 0123456780246810スレッド数速度向上比|Zn|>3ブロック|Zn|>3サイクリック|Zn|>2ブロック|Zn|>2サイクリック

図１４：発散条件変更による速度向上の比較

(5) 収束条件の変更

ここでは、複素関数 の発散条件をn>50からn>100に変更して実行した。スレッド数と実行時の解像度（2500×3700）を設定し、それぞれのブロック分割とサイクリック分割の速度向上比の比較を図１５に示す。スレッド数8でブロック分割は4.１倍、サイクリック分割では7.2倍の速度向上が得られた。この結果は前回の結果と比較しても差異は見られなかった。

0123456780246810スレッド数速度向上比n>100ブロックn>100サイクリックn>50ブロックn>50サイクリック

図１３：収束条件変更による速度向上の比較

ページ 13: [9] 削除　　1C　　2007/01/29 6:58:00

(4)、(5)の実験では条件の変更による速度向上の変化はほぼ見られなかったのに対し、(1)はブロック分割、サイクリック分割両方に、また(2)、(3)ではブロック分割にだけ差異が見られた。(1)は計算量が少ないのでプロセッサ1台で十分処理できる計算を無駄に分割し、並列化してことで計算量を増やし、オーバーヘッドが発生したと考えられる。サイクリック分割が(2)、(3)の場合も変化がないのは分割を多くして各プロセッサに割り振るため、情報量の差が少なくなり負荷が小さくなるためである。よって、サイクリック分割はブロック分割よりも理想的な速度向上が見られるのである。

セクション区切り (次のページから新しいセクション)

5. NASパラレルベンチマークによる性能評価

5.1 実験環境

ページ 13: [10] 削除	1C	2007/01/29 6:58:00

これまでの実験結果から並列化の負荷分散を少なくするためには、より細かく分割する方法が良いと考えられる。そのために３章で行った実験と同条件のマンデルブロ集合を実軸と虚軸の両方面から細かく分割し、各スレッドに割り当てる方法で実行した。以下の問題をブロック分割、サイクリック分割、格子状サイクリック分割で並列化し、結果を比較、考察した。

5.2 実験結果