2004 年度 修士論文

MPIETE2：
MPI プログラム実行時間予測ツール MPIETE の
通信時間予測誤差に関する改良

提出日： 2005 年 2 月 2 日

指導教員：山名 早人

早稲田大学大学院 理工学研究科 情報・ネットワーク専攻

学籍番号：3603U017-1

岩渕 寿寛

概要

本稿では, MPI プログラム実行時間予測ツール MPI Execution Time Estimator (MPIETE) の, 通信予測手法の改善案を提案する. MPIETE は, 本学山名研究室の堀井洋が開発したツールであり, プログラムを計算ブロック, 通信ブロックに分割し, 各ブロックの実行時間情報からプログラム全体の実行時間を予測する. 予測の際, 各ブロックの実行時間情報の取得を高速に行うことで, 実際の実行よりも速く実行時間予測が可能である. しかし, MPIETE は, 通信コンテンション等が発生しない, 理想的な環境での予測を前提としていたため, 通信コンテンションの発生に伴う通信の待ち時間を予測することが不可能であり, 通信時間予測誤差に問題がある. 本稿では, MPIETE の通信予測手法を変更し, 通信の待ち時間を考慮したブロック実行時間情報を導入する. 本稿で示す提案手法は, 通信の待ち時間による通信性能の低下を予測し, 予測対象計算機上で最大の実行性能を示す Processing Unit(PU)数の特定を可能とするものである. また筆者は, 本手法を MPIETE に適用し, MPIETE2 を開発した. 本稿では, MPIETE2 を用いて, NAS Parallel Benchmarks (NPB) ver2.4 の実行時間の予測を行った結果も示す. NPB2.4 の EP, CG, FT, MG の CLASS B の実行時間を予測したところ, 2-128PU の実行で予測誤差は 14%以内であり, 予測に必要な処理時間は, 実際の実行と比較して約 1/4 の時間であった. 特に, すべてのプログラムにおいて, 通信性能の低下を予測でき, 実際の実行よりも速く, 最大の実行性能を示す PU 数を特定できた.

目次

第1章 はじめに

本章では, 研究の目的と概要, 及び本稿の構成を示す.

1.1 研究の目的と概要

PC クラスタや Grid テストベッドの普及により, 近年, 並列計算に対する注目が増してきている. 一般に並列プログラムは, Processing Unit (PU) 数を変化させ実行することが可能で, ユーザは実行 PU 数によって様々な結果を得ることができる. しかし, 並列プログラムは, PU 数を増やして実行しても, 期待する実行性能を得られない場合があり, ある実行 PU 数以降から実行性能は低下していくのが普通である. つまり, 並列プログラムの性能向上には, プログラムの特徴を見極め, 適切な PU 数で実行することが重要となる.

そのような中,「ある並列プログラムを, 予測対象計算機上で実行した際, プログラムの性能を実行する前に予測する」という, 並列プログラムの性能予測に対する要求が高まってきている. 実際の実行に先立ち, 並列プログラムの性能を予測することは, 先に述べたプログラム自身の性能を向上させるだけでなく, 最大の実行 PU 数で実行することで, 余剰 PU を他の計算に利用し, 結果的に並列計算機の稼働率を向上させる有用性もある.

従来, 並列プログラムの性能を予測する手法として, 並列プログラムの実行時間を予測する手法が提案されている. 従来の実行時間予測手法としては, シミュレータを用いて, アセンブラコードのトレースを行う予測手法[4]や, シミュレーションを行わず, ソースコードを静的に解析することで, プログラムの実行時間を予測する手法[5][6]が提案されている. 前者はキャッシュヒット率や, ネットワークレイテンシ, バンド幅の変化に対応した挙動の解析が可能であるが, 予測にかかる時間として, 実際の実行時間に対し数倍から数十倍の時間を要する. また後者はコンパイラ最適化やキャッシュ効果, ネットワーク性能を考慮することが困難であり, 前者に比較して予測精度が悪い.

上記した問題に対して, MPI Execution Time Estimator (MPIETE)という実行時間予測ツールを用いた手法[10]が提案されている. MPIETE を用いた予測手法は, MPI[1]を用いた並列プログラムを, 計算ブロック, 通信ブロックに分割し, 各ブロック 1 回分の実行時間 (ブロック基礎データ) と実行回数からプログラムの実行時間を予測するものである. [10]では, ブロック基礎データと実行回数の取得の際, 不必要な実行文のコメントアウトや, ブロックの繰り返し回数を削減することで, 予測に必要な処理時間を軽減し, 実際の実行よりも速い予測を実現している. しかし, 通信の予測の際, 通信コンテンションの影響が無い理想的な環境を前提としているため, 通信性能の低下を予測できない.

本稿では, MPIETE の通信ブロックの予測手法に変更を加え, 通信性能の低下による並列プログラム全体の性能低下を予測する手法を提案する. 本手法は, 通信コンテンションの発生による, 通信の待ち時間を考慮した通信ブロック基礎データを予測に用いることで, 上記した問題の解決を図るものである. また, 筆者は, MPIETE に本手法を導入し, MPIETE2 を開発した. MPIETE2 は, 予測対象 MPI プログラムから, 計算ブロック基礎データ, 通信ブロック基礎データ, ブロック実行回数を取得するプログラムを自動生成する. ユーザは, 自動生成されたプログラムを予測対象計算機上で実行し, 結果を MPIETE2 に入力することで実行時間予測結果を得ることができる.

1.2 本稿の構成

本節では, 本稿の構成を示す.

第 2 章では, これまでに提案されてきた並列プログラムの性能予測手法を, Instruction-Level 型シミュレーション, Static-Parameter 型シミュレーション, Execution-Driven 型シミュレーションの3つに分類し, それぞれの特徴、問題点をあげる.

第 3 章では, MPIETE[10]の通信部分の予測に改良を行った提案手法について述べる.

第 4 章では, 第 3 章で述べる提案手法を用いて開発した MPIETE2 の機能および, MPIETE2 を用いた予測の流れを示す.

第 5 章では, MPIETE2 を用いて NPB2.4 の実行時間を予測した結果を示す. 予測結果では, 実行時間予測誤差および, 予測に必要な処理時間について示す.

最後に第 6 章では, 本稿のまとめを行う.

第2章 関連研究

本章では, これまでに提案されてきた並列プログラムの実行時間予測手法を示す. 本論文では, これまで提案されてきた手法を Instruction-Level 型シミュレーション, Static-Parameter 型シミュレーション, Execution-Driven 型シミュレーションの3つに分類する.

Instruction-Level 型シミュレーションは, インストラクションレベルで並列プログラムの挙動を解析し, 実行性能を予測する手法である. Static-Parameter 型シミュレーションは, 並列プログラムを静的に解析することで, 並列プログラムを実行した際の性能を予測する手法である. Execution-Driven 型シミュレーションは, プログラムを複数ブロックに分割し, 各ブロックを実際に実行することで得られるブロック実行時間情報から, プログラム全体の実行時間を予測する手法である.

本章ではこれより, Instruction-Level 型シミュレーション, Static-Parameter 型シミュレーション, Execution-Driven 型シミュレーション毎に, 従来提案されてきたシミュレーション手法を示し, それぞれの特徴, 問題点を示す.

2.1 Instraction-Level 型シミュレーション

インストラクションレベルで並列プログラムの挙動を解析し, 実行性能を予測する手法が提案されている. 本論文ではこれを Instruction-Level 型シミュレーションと呼ぶこととする. Instraction-Level 型シミュレーションは, プログラムの実行対象 PU 上での実行インストラクションの詳細な影響をシミュレーションするものとして, 逐次プログラムの予測にも用いられる伝統的な手法である.

しかし, 一般にシミュレーションの精度が高いほど, 予測に必要な時間は比例して大きくなる. また, 並列プログラムの予測には, PU ごとの挙動予測の他に, PU 間における通信のシミュレーションも行う必要があり, 予測に必要な時間はさらに大きくなる. 現在までに, Instruction-Level 型シミュレーションとして, EXCIT[2]と INSPIRE[3]を利用した手法[4]が提案されている.

2.1.1 久保田等の手法[4]

1998 年, Real World Computing Partnership の久保田等は, 計算部分は EXCIT[2]を, 通信部分は INSPIRE[3]を利用して, データ並列プログラムの実行時間を予測する手法を提案した. 尚, 計算部分とは2つの通信関数に挟まれた部分を指し, データ並列プログラムは, 通信の内容によって処理内容が変化しないことを前提としている.

EXCIT は, ソースプログラムをインストラクションレベルで解析し, 計算時間を予測するツールであり, 計算部分の実行に必要なクロック数を, CPU のシミュレータを用いて算出する. INSPIRE は, クロックレベルでネットワークのシミュレーションを行うツールであり, PU 構成をモデル化する言語である NDF(Network Description File)を用いて通信時間を予測する.

図 2.1-1 に, EXCIT と INSPIRE を用いたシミュレーション全体の構成を示す.

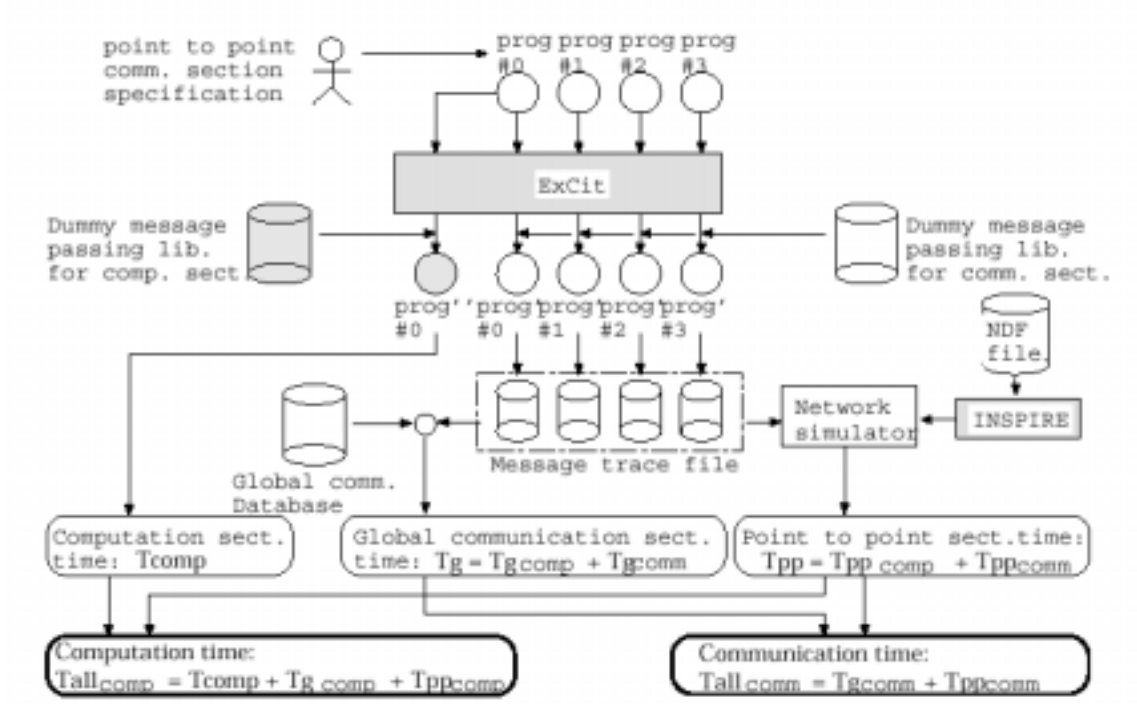

図 2.1-1：EXCIT と INSPIRE によるシミュレーションの流れ（[4]より転載）

EXCIT と INSPIRE を利用したシミュレーションの処理の流れを以下に示す.

（1） オリジナルの並列プログラム(Prog#0～Prog#3)に対し, EXCIT でインスツルメントを施し, 計算時間予測に必要な中間コード Prog”#0 と, 通信時間予測に必要な中間コード Prog’#0 ～Prog’#3 を得る.

（2） Prog”#0 を実行することで並列プログラムの計算時間を求める.

（3） Prog’#0～Prog’#3 を個別に実行し, プログラムごとに通信トレースファイルを生成する. 通信トレースファイルをネットワークシミュレーションすることで並列プログラムの通信時間を得る.

[4]では, Sun Ultra-1 1台を用いて, NAS Parallel Benchmarks の MG プログラム, 問題サイズ CLASS B 実行時間を予測している. 16PU を用いた際の実行時間予測誤差は, 10%以内で収まっている. しかし, 実際の MG プログラム実行時間に比べて, Prog”#0 の実行には数十倍の時間がかかり, MG プログラムの通信トレースファイルの作成には 20 分程度かかる.

2.2 Static-Parameter 型シミュレーション

並列プログラムを静的に解析することで, 並列プログラムを実行した際の性能を予測する手法が提案されている. 本論文ではこれを, Static-Paramete 型のシミュレーションと呼ぶことにする. Static-Parameter 型のシミュレーションは実際に予測にかかる時間は少ない. しかし, 静的な解析のみでは, コンパイラの最適化, キュッシュ効果, ネットワーク性能を考慮し, 様々なアーキテクチャに対応した予測をすることができない. また Instruction-Level 型の予測手法に比べ, 予測精度が悪い. 現在までに[5], [6]の手法が提案されている.

2.2.1 Yarrow 等の手法[5]

1997 年, NASA Ames Research Center の Yarrow 等は, NAS Parallel Benchmarks LU プログラムの持つ, キャッシュヒット率, 通信コストの計算を, 静的に行うことにより, 実行時間を予測している. [5]では, LU プログラムタイムステップイタレーション毎の実行時間[s]を,

$$\left[\frac{n^3}{2^{2k}}\cdot 485f + 4\left(\frac{n^2}{2^k}\cdot 80b + l\right)\right] + \left(1+\frac{2(2^k-1)}{N-2}\right)\left[2(N-2)\left(\left(\frac{N}{2^k}\right)^2\cdot 279f + 2\left(\frac{N}{2^k}\cdot 40b + l\right)\right) + (N-2)\left(\frac{N}{2^k}\right)^2\cdot 953f\right] \quad (p=2k \text{ のとき})$$

$$\left[\frac{n^3}{2^{2k+1}}\cdot 485f + 2\left(\frac{n^2}{2^k}\cdot 80b + l\right) + 2\left(\frac{n^2}{2^{k+1}}\cdot 80b + l\right)\right] + \left(1+\frac{3\cdot 2^k-2}{N-2}\right)\left[2(N-2)\left(\frac{N^2}{2^{2k+1}}\cdot 279f + \frac{N}{2^k}\cdot 40b + \frac{N}{2^{k+1}}\cdot 40b + 2l\right) + (N-2)\frac{N^2}{2^{2k+1}}\cdot 953f\right] \quad (p=2k+1 \text{ のとき})$$

と表している.

ここで, p, N はそれぞれ, プログラムの実行 PU 台数を 2^p, 問題サイズの大きさを N^3 と表したときのパラメータであり, その他 f, l, b は, プログラム中の浮動小数点演算毎にかかる平均コストを f[s], 通信レイテンシを l[s], バイトあたりの転送時間を b[s]としたものである.

[5]では予測対象機として IBM SP2 を用い, b=29[Mbytes/s], l=54[μs]として, LU プログラムの CLASS A 及び B での実行時間予測を行った結果, 予測誤差 30%以内で予測できている.

2.2.2 Rothberg 等の方法[6]

1993 年, Intel Supercomputer Systems Division の Rothberg 等は, キャッシュサイズ, PU の処理能力の影響と, 通信時間を静的に予測している.

予測は LU 分解, CG（Conjugate Gradient）, FFT（Fast Fourier Transform）, Barnes-Hut（段階的木構造を持った N-Body シミュレーション）, レイトレーシング法を利用したボリュームレンダリングの 5 つの並列アルゴリズムに関して行われている. 表 2.2-1 の Data に示すデータサイズの問題が与えられたときに, 総演算数（Opt）, 通信量（Communication）, 問題の並列性（Concurency）のオーダーを以下のように静的に見積もっている. 例えば, LU では入力データとして n 次正方行列が与えられたことを意味し, p は実行 PU 台数, θ はユーザが指定する, $0.5 \leqq \theta \leqq 1.2$ を満たす変数である.

[6]では, 実際に実機との比較をしていないが, すべてのプログラムにおいて, キャッシュサイズが 32KM 以上だと, キャッシュヒット率は 90%得られると予測している.

表 2.2-1：各プログラム計算量の内訳

Application	Data	Opt	Concurency	Communication
LU	n^2	n^3	n^2	$n^2\sqrt{p}$
CG	n^2	n^2	n^2	$n\sqrt{p}$
FFT	n	$n\log n$	n	$n\log p$
Barnes-Hut	n	$\frac{1}{\theta^2} n\log n$	n	$n^{\frac{1}{3}}\theta^3 p^{\frac{2}{3}} \log p^{\frac{4}{3}}$
Volume Rendering	n^3	n^3	n^2	n^3

2.3 Execution-Driven 型シミュレーション

プログラムを複数ブロックに分割し, 各ブロックを実際に実行することで得られるブロック実行時間情報から, プログラム全体の実行時間を予測する手法が提案されている. 本論文ではこれを, Execution-Driven 型シミュレーションとよぶこととする. Execution-Driven 型シミュレーションは, Instruction-Level 型シミュレーションと比べ, 予測にかかるオーバヘッドが小さいという特徴を持つ. しかし, 予測に必要な時間は, 実際の実行時間よりも大きい. 現在までに[8], [9], [10]の手法が提案されている.

2.3.1 Dickens 等の手法 [8]

1996 年, NASA Langley Research Center の Dickens 等は, LAPSE というシミュレータを利用し, 並列プログラムの実行時間予測を行った. LAPSE は, 並列計算機 Intel Paragon でのみ動作するシミュレータである.

LAPSE では, 計算部分は, 実際に想定するシステム上で計算させることにより, 計算時間を予測する. 予測対象計算機の PU 数以下の計算機で予測を行う場合は, 予測を行う計算機 1PU で複数の PU 上で行われる計算部分を測定する必要がある. LAPSE では, 予測を行う計算機 1PU では, 常に予測対象計算機 1PU の計算部分の測定のみを行うようにすることで, 正確な計算時間の測定を行う.

また, 通信部分の予測は, それぞれの PU で別々に予測している予測時間の同期をとり, メッセージサイズに比例した通信時間を想定することで予測を行う. LAPSE では, それぞれの PU の実行時間は, 別々に予測が行われているため, ある PU の受信の実行時間を予測する場合は, 送信する PU の送信が行われる実行時間を求める必要がある. LAPSE では, 複数 PU 間の予測時間の同期をとるため, WHOA[1]という通信同期プロトコルを利用する. WHOA を用い, 全てのプロセスにおいて通信関数が呼ばれる時間を予測し, 通信の同期をとることで通信時間の予測を行う.

[8]では, SOR (Successive OverRelaxation 法), BPS (Bramble, Pasciak, Schatz により提案された領域分割法), APUCS (連続時間マルコフ離散事象シミュレーション), PGL(Parallel Graphic Library の略[2]) という 4 つのアプリケーションを対象に, 実行時間予測を行っているが, 全てのアプリケーションにおいて, 予測の誤差は 10%以内であった. しかし, APUCS アプリケーションにおいて, 8PU 実行時の予測を行うためには, 実際の実行時間と比べて, 2PU を利用すると約 30 倍, 40PU を利用しても約 12 倍の処理時間が必要となる.

[1] WHOA も LAPSE と同様に, Dickens において実装されたプロトコルである.
[2] 科学データの可視化を目的に実装された, 並列グラフィックライブラリであり, [8]では PGL 中のサンプルプログラムを指す.

2.3.2 Prakash 等の手法[9]

2000 年, カリフォルニア大学情報学科の Prakash 等は, MPI-SIM というシミュレータを利用し, 並列プログラムの実行時間予測を行った. MPI-SIM は MPI ライブラリを利用した C 言語のみを対象としたシミュレータであり, 予測対象となる CPU 台数 P に対して, シミュレーションに使用する CPU 台数 n が n≦P であることを前提に予測を行う.

MPI-SIM では, LAPSE と同様に, 計算時間の予測には, 予測対象機と同じ計算機構成の計算機で, 実際に実行することで行い, 通信時間の予測には, 同期プロトコルを利用することで, 異なる PU の予測時間の同期をとることで行われる. また, [9]では, 同期を必要としない通信を, ランタイム時, コンパイル時に解析することで, 同期プロトコルによる通信によるオーバヘッドを, 軽減する手法を提案している.

[9]では, 並列計算機 IBM SP2 を用いて, NPB の MG, LU, SP, BT を用いて実行時間の予測を行っている. これらプログラムの実行時間予測誤差は 5%から 20%であった. しかし SP, BT において, 16PU 実行時の予測を行うためには, 実際に実行する時間と比べて, 2PU で約 16 倍, 16PU を利用しても約 4 倍の処理時間が必要である.

2.3.3 堀井等の手法[10]

2004年, 早稲田大学理工学研究科の堀井等は, MPIETE というツールを用いて, MPI プログラムの実行時間予測を行った. MPIETE は, MPI ライブラリを利用した Fortran を対象とした実行時間予測ツールである.

MPIETE を用いた予測手法は, 並列プログラムを計算ブロック, 通信ブロック, 通信待ちブロックに分割し, 各ブロックの実行時間から, プログラム全体の実行時間を予測するものである.

計算部分の実行時間は, 予測対象計算機上で実際に実行することによって予測する. その際, 計算ブロックの実行回数を削減することで, 計算ブロック実行時間の予測を高速に行う.

通信部分の実行時間は, 通信サイズに比例した実行時間を割り当てることで予測を行う. その際, 集団通信関数は, 2PU で行う通信に変換することで, 予測対象 PU 数に関わらず, 予測は2PU で行うことができる.

[10]では, 128PU までの性能ホモな PC クラスタを用いて, NPB EP, CG, LU の予測を行っている. 16PU までの予測では, 予測誤差は 10%以内であり, 予測に必要な処理時間は実際の実行時間と比較して 1/10 以内である. ただし, 32PU 以上の実行では, 予測誤差は 30%以上あり, 最大の実行性能を示す PU 台数を予測できていない. [10]では, 通信コンテンション等の影響のない, 理想的な環境下での予測を前提としている. そのため, 全実行時間中, 通信時間が支配的になり, かつ通信コンテンションが発生するような PU 数では, 予測誤差が大きくなると述べている. また, 通信部分の予測で, 集団通信関数を 2PU に行う通信に変換するが, その通信方法は MPI 規格の実装依存であるという問題も存在する.

2.4 関連研究のまとめ

これまでに提案されている予測手法を，表 2.4-1 に示す．

表 2.4-1 従来の提案手法の比較

手法の型	提案者	予測対象プログラム	予測誤差	予測 オーバヘッド	欠点
Instruction-Level 型	久保田等[4]	NPB2.3 MG，LU	10%以内	10 倍以上	シミュレータの生成する中間コードおよび，ネットワークシミュレーションに膨大な時間がかかり，予測オーバヘッドが大きい
Static-Parameter 型	Yarrow 等[5]	NPB2.3 LU	30%以内	低	プログラム毎に実行時間を手動で導き出す必要があり，汎用性がない．また，モデル化にあたり，一定の通信時間を想定するため，予測誤差が大きい．
	Rothberg 等[6]	LU，CG，FFT， Burnes-Hut Volume Rendering		低	解析方法が全てのプログラムに適用できず，汎用性がない．なお，実記との比較がなされていないため，予測精度は不明
Execution-Driven 型	Dickens 等[8]	SOR，BPS， APUCS，PGL	10%以内	30 倍	通信予測の際，通信関数の実行タイミングを予測するため，PU 毎のシミュレーションを行わなくてはならず，予測オーバヘッドが大きい．
	Prakash 等[9]	NPB2.3 MG，LU，SP，BT	20%以内	5 倍以上	実行 PU 数分全てのシミュレーションを行うため，予測オーバヘッドが大きい．また，予測対象機の PU 数に予測オーバヘッドが依存する．
	堀井等[10]	NPB2.4 EP，CG，LU	10%以内 （16PU 以内） 30%以上 （32PU 以上）	1/10 以内 （16PU 以内） 10 倍 （32PU 以上）	通信コンテンションの影響を考慮していないため，通信性能の低下を予測できず，最大の実行性能を示す PU 数の特定ができない．予測方法が MPI 実装依存である．

表 2.4-1 に示す従来の予測手法は，予測精度と予測に必要な時間との妥協点を導き出すことで，利用法にあった実行性能予測を実現している．しかし，比較的予測精度の高い Instruction-Level 型と，[10]を除く Execution-Driven 型のシミュレーションでは，実際のプログラム実行時間以上の時間が必要であり，プログラムの実行時間が何日とかかるプログラムを予測

するのには適していない. また, 比較的予測オーバヘッドが小さい[10]では, 予測前提条件に問題があり, 予測 PU 数が増えると通信予測誤差が大きくなり, 最大の実行性能を示す PU 数を予測できない.

第3章 提案手法

第 2 章で示した[10]の手法は, 他の手法に比べ, 実際の実行よりも速く実行時間を予測することが可能であった. しかし, 通信部分の予測は, 通信コンテンションの発生がない, 理想的な環境で行うことを前提としているため, 通信性能の低下を予測することができない.

本章では, [10]の手法のうち, 通信部分の予測手法に変更を加え, 通信性能の低下による並列プログラム全体の実行性能の低下を予測し, 最大の実行性能を示す PU 数の特定を可能にする手法を提案する. 本手法は, 通信コンテンションの発生による, 通信の待ち時間を考慮した通信ブロック基礎データを予測に用いることで, 上記した問題の解決を図るものである.

本手法は, MPI プログラムを計算ブロック, 通信ブロックに分割し, 各ブロックの実行時間を予測することで, プログラム全体の実行時間を予測する. 各ブロックの実行時間の予測は, 各ブロック 1 回分の実行時間 (以降, ブロック基礎データとする) と, 各ブロックの実際の実行回数を取得することで行う. その際, ブロック基礎データとブロック実行回数の取得を短時間で行うことで, 実際の実行よりも速く実行時間を予測することが可能となる. この取得方法を簡易実行と呼ぶ.

本章ではこれより, 3.1 でブロックの定義及びプログラムのブロック分割方法を, 3.2 及び 3.3 で計算ブロック基礎データ及び通信ブロック基礎データの取得方法を, 3.4 でブロック実行回数の取得方法を述べ, 最後に 3.5 でプログラム全体の実行時間予測方法を述べる.

3.1 プログラムのブロック分割方法

本節では, ブロックの定義及び, 予測対象プログラムを計算ブロック･通信ブロックへ分割する方法を述べる.

まず, ブロックの定義を以下に示す.

- 計算ブロック ：
  MPI 通信関数を除く, 繰り返し文の最内ループボディ及び,
  繰り返し文で区切られる連続した実行文
- 通信ブロック ：
  メッセージの送受信を行う MPI 通信関数を含む実行文

次に, プログラムのブロック分割方法を以下に示す.

(1) 繰り返し, 条件分岐を行う文や, GOTO･STOP･RETURN･END 文を含まない, 連続した実行文の集合を計算ブロックとする.
(2) MPI 通信関数を含む文を通信ブロックとする.
(3) (1)で得られた計算ブロック内で, MPI 通信関数が宣言されている場合は, MPI 通信関数の前後で 2 つの計算ブロックに分割し, また, MPI 通信関数を含む文を通信ブロックとする.

図 3.1-1 に, プログラムのブロック分割例を挙げる. 図 3.1-1 右図の赤枠で囲われた部分が計算ブロックであり, 緑枠で囲われた部分が通信ブロックである. そして計算ブロック, 通信ブロックには各々一意のブロック ID を振る.

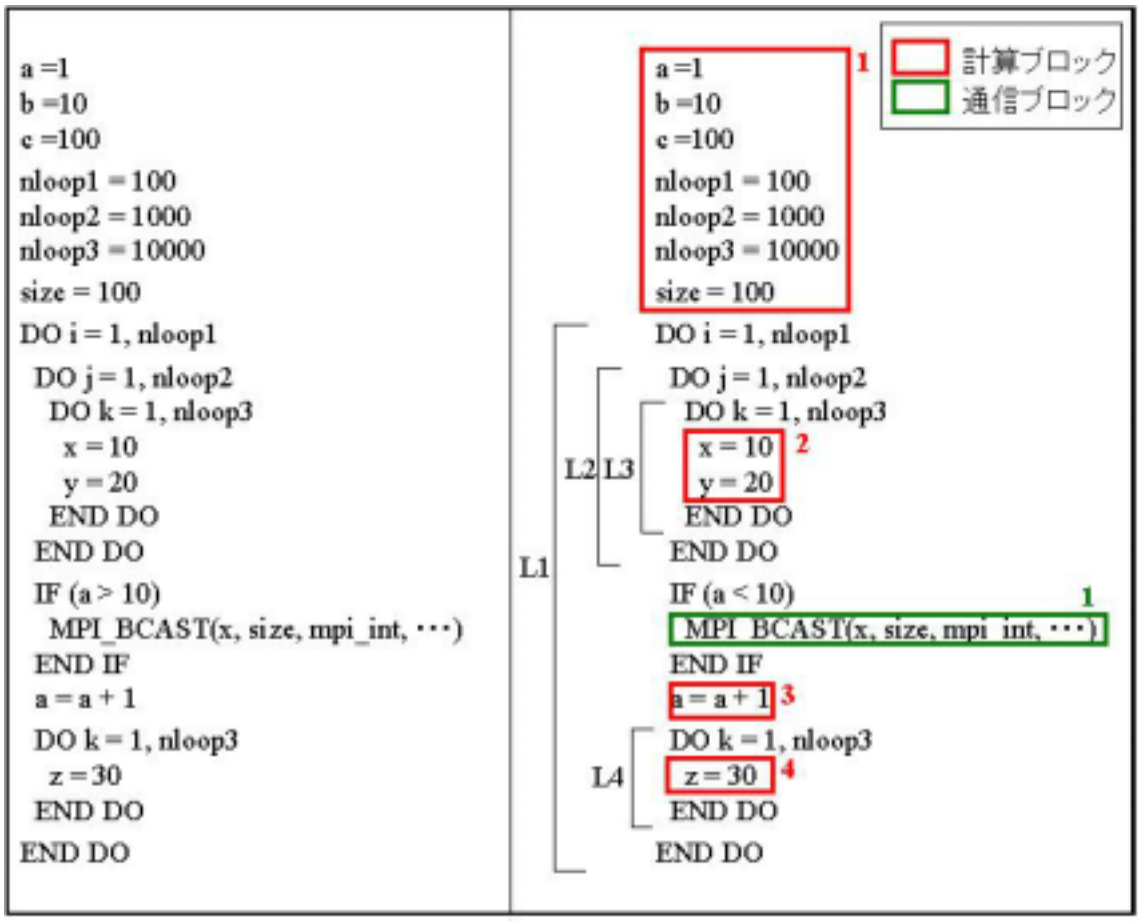

図 3.1-1 プログラムのブロック分割例

3.2 計算ブロック基礎データの取得方法

本節では, 計算ブロック基礎データの定義及び, 計算ブロック基礎データ取得方法を述べる.

計算ブロック基礎データとは, 計算ブロック 1 回分の実行時間であり, 計算ブロック基礎データは, 予測対象プログラムを予測対象計算機上で実際に実行し, 得られた以下の値から算出する. 以降, 計算ブロック基礎データを $Tb\_comp(i)$ (*i* は計算ブロック ID) とする.

- $Te\_comp(i)$ ： 予測の際に実行した計算ブロックの総実行時間
- $Ne\_comp(i)$ ： 予測の際に実行した計算ブロックの総実行回数

なお, 上記で示す値を求めるための測定文は, 計算ブロックの前後ではなく, 「当該計算ブロックのみをループボディとして持つ DO ループの前後」に挿入する. これはキャッシュ効果, コンパイラ最適化を考慮した計算ブロック基礎データを取得するためである. ただし, 計算ブロックが DO ループ構造内にない場合, 測定文は計算ブロックの前後に挿入する. 図 3.1-1 のプログラム例では, 計算ブロック 1, 3 は計算ブロックの前後に, 計算ブロック 2 は L2 ループの前後に, 計算ブロック 4 は L4 ループの前後に測定文を挿入する.

また, 計算ブロック基礎データの取得を簡略化するため, 以下の処理を施す.

(1) 計算ブロック基礎データを変化させない実行文をコメントアウト
(2) 計算ブロック基礎データ測定対象外の DO ループの繰り返し回数を削減

(1)で述べた計算ブロック基礎データを変化させる実行文とは

- 計算ブロック内の文
- 計算ブロック内で利用される配列変数の, インデックスとして参照される変数を決定する文
- 計算ブロック基礎データ測定対象の計算ブロック実行回数を決定する文
- IF 文等の, 計算ブロックの計算順序を決定する文

であり, これ以外をコメントアウトする.

(2)で述べた DO ループ繰り返し回数の削減に関しては, キャッシュ効果を考慮すると, ある程度ループを回す必要がある. 本稿では, 繰り返し回数を 1/10 に削減した.

図 3.1-1 のプログラム例に対して, 図 3.2-1 に上記した修正を施したプログラムを示す. 図 3.2-1 右図が修正後のプログラムである. 赤点線で囲われたブロックが, 計算ブロック基礎データを測定する区間を意味する. また, (1) より, 通信ブロック 1 は受信データが計算ブロック基礎データを変化させないためコメントアウトし, (2)より L1 ループの繰り返し回数を削減する.

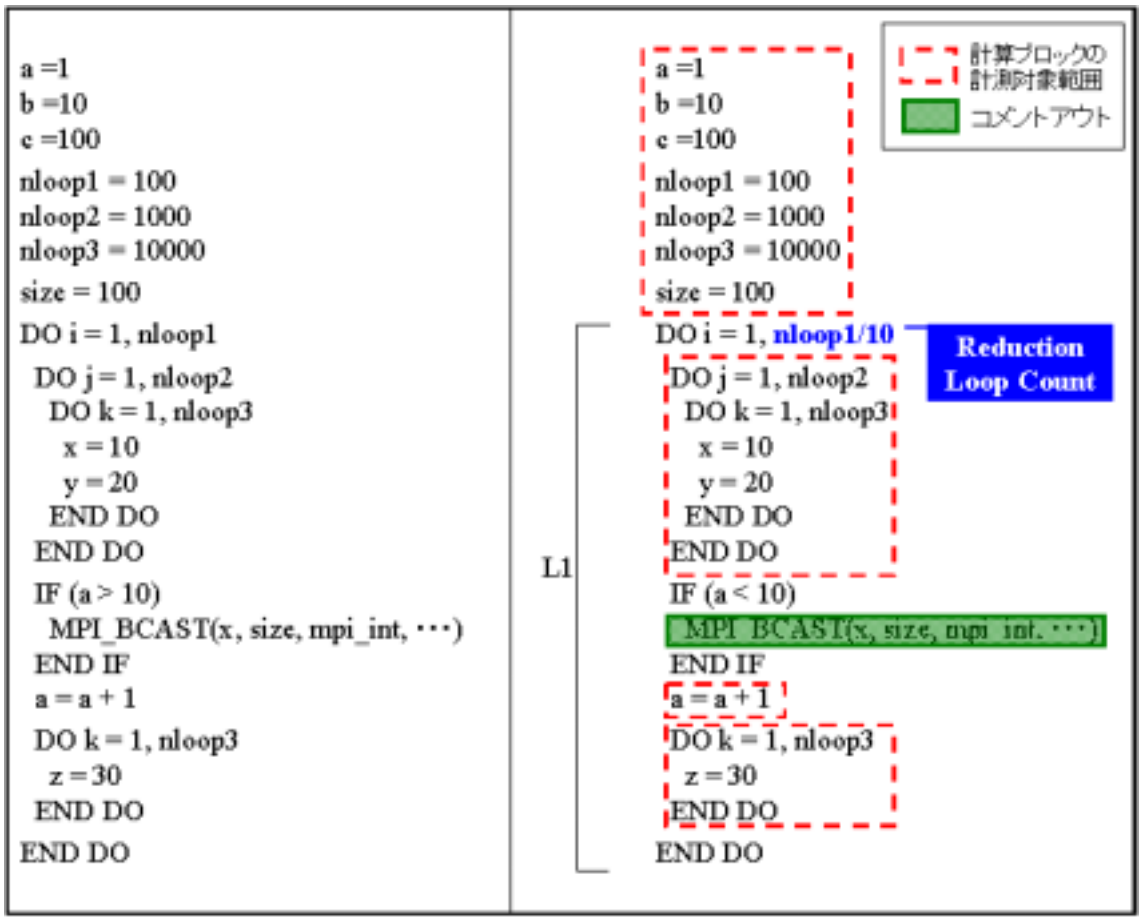

図 3.2-1 計算ブロック基礎データ取得プログラム例

そして，上記した修正を施した予測対象プログラムを，予測対象計算機上で実行することで計算ブロック基礎データを高速に取得する．この簡略化した計算ブロック基礎データの取得方法を，計算ブロックの簡易実行と呼ぶ．

3.3 通信ブロック基礎データの取得方法

本節では, 通信ブロック基礎データの定義及び, 通信ブロック基礎データの取得方法を述べる. 本節で述べる通信部分の予測手法が, 以下に示す[10]の手法の問題点を解決するものである.

- 予測方法が MPI 規格の実装依存である
- 通信コンテンションの発生による, 通信の待ち時間を予測できない

MPIETE[10]では, 通信ブロック基礎データは, 予測対象計算機上のネットワークにおける, 通信サイズに比例した通信時間であった. この通信時間と通信トレースファイルから通信時間を算出していくが, 通信コンテンションの発生に伴う通信の待ち時間を予測できず, 通信性能の低下を特定できない. そのため, 本手法では, 以下のように通信ブロック基礎データの定義を変更し, 上記した問題の解決を図る.

通信ブロック基礎データとは, 通信ブロック 1 回分の実行時間であり, 通信ブロック基礎データは, 予測対象プログラムを予測対象計算機上で実際に実行し, 得られた以下の値から算出する. 以降, 通信ブロック基礎データを $Tb\_comm(j)$（j は通信ブロック ID）とする.

- $Te\_comm(j)$ ： 予測の際に実行した通信ブロックの総実行時間
- $Ne\_comm(j)$ ： 予測の際に実行した通信ブロックの総実行回数

なお, 上記で示す値を求めるための測定文は, 通信ブロックの前後に挿入する.

また, 通信ブロック基礎データの取得を簡略化するため, 以下の処理を施す.

(1) 通信ブロック基礎データを変化させない実行文のコメントアウト
(2) プログラム全体の通信順序を変化させない限り, DO ループの繰り返し回数の削減

(1)で述べた, 通信ブロック基礎データを変化させる実行文とは,

- 通信データサイズを決定する文
- 通信相手及びメッセージ ID を決定する文
- IF 文等の, 通信順序を決定する文

であり, これ以外をコメントアウトする.

(2)で述べた DO ループの繰り返し回数の削減に関しては, 通信コンテンションの発生を考慮すると, ある程度ループを回す必要がある. 本稿では, 繰り返し回数を 1/10 に削減した. なお, 通信毎に通信データサイズや通信相手･メッセージ ID が変化する場合, ループ回数の削減は行わない.

図 3.1-1 のプログラム例に対して, 図 3.3-1 に上記した修正を施したプログラムを示す. 図 3.3-1 右図が修正後のプログラムである. 緑点線で囲われたブロックが, 通信ブロック基礎データを

測定する区間を意味する．また，塗りつぶされた赤枠は，(1) よりコメントアウトする計算ブロックを意味し，(2)より L1 ループの繰り返し回数を削減する．

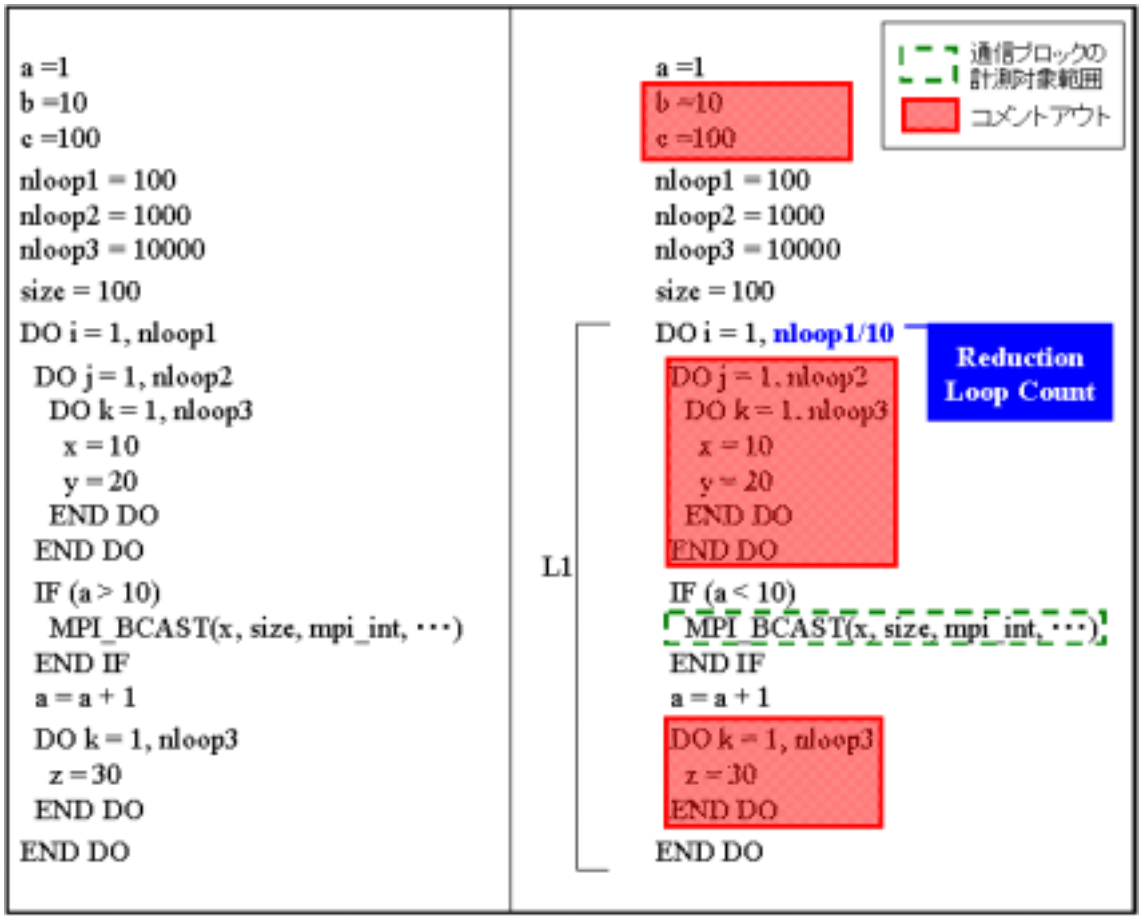

図 3.3-1 通信ブロック基礎データ取得プログラム例

そして，上記した修正を施した予測対象プログラムを，予測対象計算機上で実行することで通信ブロック基礎データを高速に取得する．この簡略化した通信ブロック基礎データの取得方法を，通信ブロックの簡易実行と呼ぶ．

3.4 ブロック実行回数の取得方法

本節では, 各ブロックの実際の実行回数を取得する方法を示す. ブロック実行回数は以下の方法で取得する. 以降, 計算ブロック, 通信ブロックの実際の実行回数を $Nr\_comp(i)$, $Nr\_comm(j)$ とする.

(1) 各ブロックの実行回数を変化させない実行文のコメントアウト
(2) ループボディにブロック実行回数を決定する実行文がない限り, 繰り返し回数を削減する

(1)で述べたブロック実行回数を変化させる実行文とは,

- 繰り返し回数を決定する文
- IF 文等の, 他のブロックの計算順序・通信順序を決定する文

であり, これ以外をコメントアウトする.

(2)で述べた繰り返し回数の削減は, 本稿では 1 回に削減した.

図 3.1-1 のプログラム例に対して, 図 3.4-1 に上記した修正を施したプログラムを示す. 図 3.4-1 右図が修正後のプログラムである. 塗りつぶされた部分が(1)よりコメントアウトするブロックを意味し, (2)より該当ループの繰り返し回数を削減する.

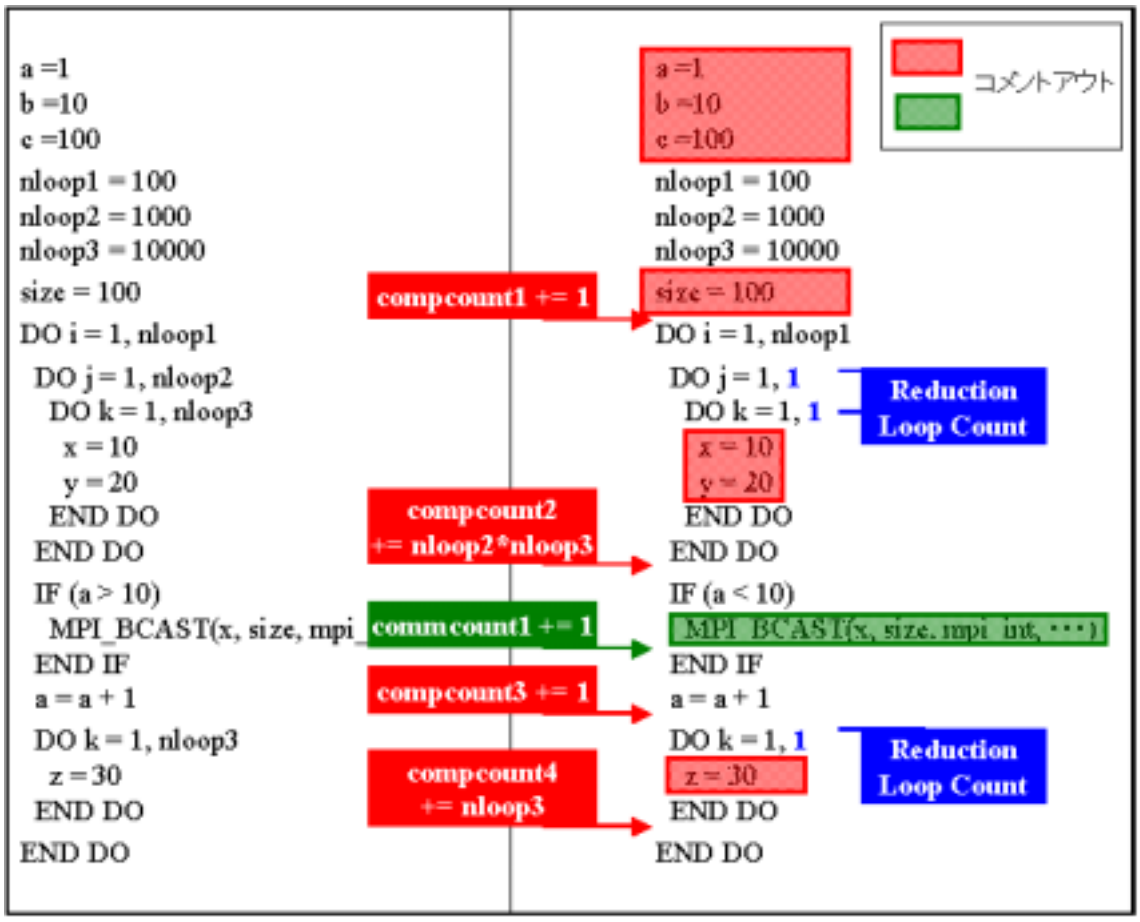

図 3.4-1 ブロック実行回数取得プログラム例

そして，上記した修正を施した予測対象プログラムを，予測対象計算機上で実行することでブロック実行回数を高速に取得する.

3.5 プログラム全体の予測時間算出方法

本節では, プログラム全体の予測時間算出方法を示す.

3.2～3.4 で示した方法で得た基礎データを用い, 以下の式よりプログラム全体の予測時間 Te_all を算出する.

$$Tb\_comp(i) = \frac{Te\_comp(i)}{Ne\_comp(i)}$$

$$Tb\_comm(j) = \frac{Te\_comm(j)}{Ne\_comm(j)}$$

$$Te\_all = \sum_i Tb\_comp(i) \times Nr\_comp(i) + \sum_j Tb\_comm(j) \times Nr\_comm(j)$$

第4章 MPIETE2

本章では, 筆者が開発した, MPI プログラムの実行時間を予測するツール MPI Execution Time Estimator 2 (MPIETE2) について示す. MPIETE2 は, MPIETE[10]で利用される予測手法の中で, 第 3 章で示した通信部分の予測手法に改良を加え, 通信コンテンションによる待ち時間を考慮した通信時間の予測を可能にしたものである. 本章ではこれより, MPIETE2 の予測対象プログラム, MPIETE2 の機能, MPIETE2 を用いた予測の流れを示す.

4.1 MPIETE2 の予測対象プログラム

MPIETE2 で予測可能なプログラムの条件を以下に示す.

- 予測対象計算機上で実行可能である
- FORTRAN77 で記述されている
- 通信は, MPI 関数で記述された通信のみである
- 同じプログラムを実行した際, 常にブロック実行回数が一定である

1 つ目の条件に関しては, MPIETE2 は簡易実行により基礎データおよび実行回数を取得する. それ故, 予測対象計算機上での実行可能が予測の前提となる.

2 つ目, 3 つ目の条件に関しては, MPIETE2 はプログラムの字句解析･構文解析を行い, ブロックに関する処理を行う. それ故, 構文解析可能もしくは, 字句解析で通信ブロックとして認識可能が予測の前提となる.

4 つ目の条件に関しては, 手法の特性上, 実際の実行時と予測の実行時で, ブロック実行回数が一致することが前提となる. 一般に, 入力セットが異なるとブロック実行回数は変化する. 本手法では, 実行時に入力セットは同じものを利用するものとする. ただし, 入力セットが一致しても, ブロック実行回数が異なるものに, ランダマイズドアルゴリズムがある. ランダマイズドアルゴリズムは, NP 問題等解くのに指数時間がかかるような問題に対して, 入力セットをランダムに抽出し, 計算することで, 逐次的に解くよりも高速に解を導き出す手法である. 本アルゴリズムを適用しているアプリケーションは, 入力セットが毎回異なり, ブロック実行回数が実行毎に変化するため, 本手法の予測対象プログラムから除外する.

4.2 MPIETE2 の機能

MPIETE2 の, MPI プログラムの実行時間を予測するための機能を示す.
MPIETE2 は, 以下の機能を持つ.

- MPI プログラムより, 計算ブロック基礎データ取得プログラムを自動生成する
- MPI プログラムより, 通信ブロック基礎データ取得プログラムを自動生成する
- MPI プログラムより, ブロック実行回数取得プログラムを自動生成する
- 計算ブロック基礎データファイル, 通信ブロック基礎データファイル, ブロック実行回数データファイルより, プログラムの実行時間を予測する

また, MPIETE2 の仕様を表 4.2-1 にまとめる.

表 4.2-1 MPIETE2 の仕様

プログラム言語	C++
使用コンパイラ	Visual C++ .NET 2003
行数	19635
モジュール数	813

4.3 MPIETE2 による実行時間予測の流れ

図 4.3-1 に, MPIETE2 による実行時間予測の流れを示す.

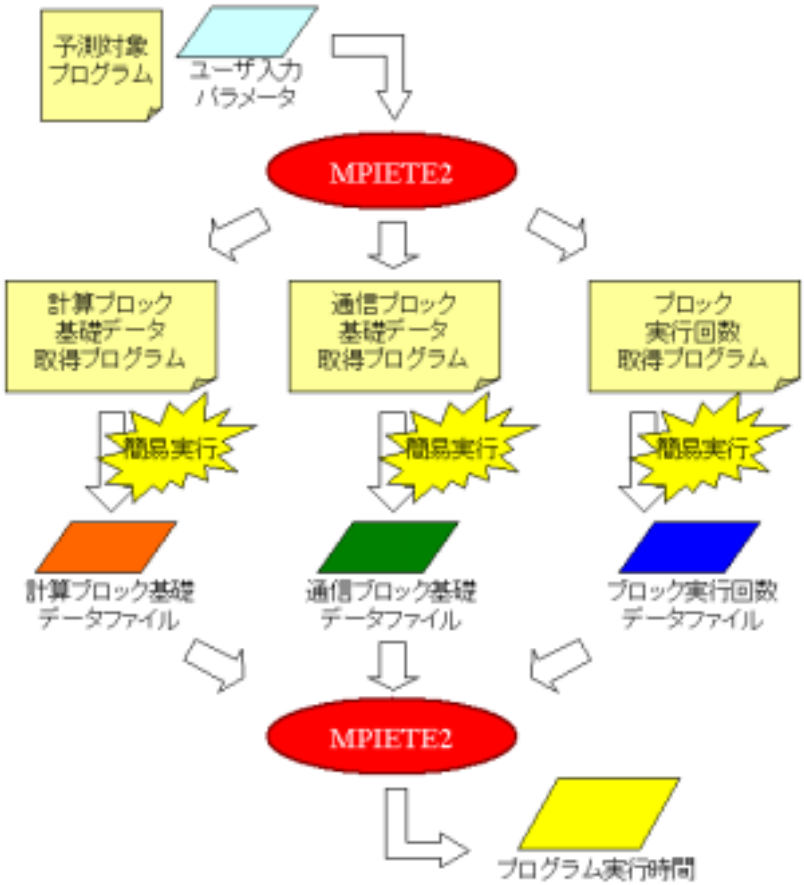

図 4.3-1 MPIETE2 を用いた実行時間予測の流れ

MPIETE2 では, 以下の手順で実行時間予測を行う.

(1) ユーザが, 予測対象プログラムファイルとユーザ入力パラメータを入力する
ユーザ入力パラメータは以下のものである
- プログラムファイル名及びプログラムファイルパス
- 実行時間予測対象区間 (開始行, 終了行)

(2) MPIETE2 が, 予測対象プログラムに対して字句解析, 構文解析及び, 基礎データに合わせた変数依存解析を行い, 以下 3 つの基礎データ取得プログラムを自動生成する
- 計算ブロック基礎データ取得プログラム

- 通信ブロック基礎データ取得プログラム
- ブロック実行回数取得プログラム

(3) ユーザが, MPIETE2により生成された3つのプログラムファイルを予測対象計算機上で実行し, 各基礎データを取得する
基礎データは以下の3つからなる
- 計算ブロック基礎データ
- 通信ブロック基礎データ
- ブロック実行回数

(4) ユーザが, 取得した各基礎データを MPIETE2 に入力する

(5) MPIETE2 が, 入力された各基礎データからプログラム全体の予測実行時間を算出する

第5章 提案手法の評価

本章では, 第 4 章で示した MPIETE2 を用いて, NAS Parallel Benchmarks ver2.4 (NPB2.4) の実行時間を予測し, 予測実行時間と実際の実行時間の比較を行う.

本章ではこれより, まず, 5.1 で NPB2.4 の概要を, 5.2 で予測対象計算機構成を示す. 続いて 5.3 で NPB2.4 の実行時間予測結果を, 5.3.3 で予測に必要な処理時間を示し, 最後に 5.5 で予測結果についてのまとめを行う.

5.1 NAS Parallel Benchmarks

本節では, 提案手法の評価に用いた NAS Parallel Benchmarks (NPB)の概要を示す.

NPB 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. NPB は, 5つの Parallel Kernel Benchmarks と3つの Parallel CFD (流体力学, Computational Fluid Dynamics の略) Application Benchmarks から構成されており, 並列コンピュータの実行性能を知る上で, 権威あるベンチマークの 1 つといえる.

また, NASA Ames Research Center が Web 上で発行しているレポート"NAS Parallel Benchmark Results"では, プログラム実行時間と, Y-MP/1 または C90/1 との性能比, 標準価格をもとにした価格性能比などについて報告される. 現在更新が止まっているため, 最新の情報を得ることはできない. しかし NPB は広く利用されていて, 多くのサイトで評価結果を公開しているため, 多くのプラットフォームと測定結果を比較することが可能である.

NPB は Kernel Benchmarks として EP, CG, FT, IS, MG が, CFD Application Benchmarks として LU, SP, BT が用意されている. プログラム言語としては IS のみ C 言語での記述で, その他は Fortran で記述されている. 各プログラムには A , B , C , W(orkstation), S(ample)の5つの CLASS が用意されている. これら CLASS による相違点は, 基本的に, 問題サイズ (配列サイズ) , 反復回数である. 問題サイズ, 反復回数は, CLASS A, B, C の順で大きくなる. NPB の 8 つのプログラムは, 全ての CLASS において, 実行する PU 台数を選択することができるが, 選択できる台数に制限があり, SP , BT では n^2 (=1,4,9,16,･･･)台, その他のプログラムは 2n (=1,2,4,8,16 ,･･･) 台での実行が可能である.

NPB は MPI 版のみでなくシリアル版も用意されており, OpenMP の研究にも広く利用される.

本稿では, EP, CG, FT, MG の CLASS B を実行時間の予測の対象とする.

5.2 予測対象計算機

本節では, 予測対象計算機の構成を示す.

予測対象計算機は, 128 ノードの PC から構成される性能ホモなクラスタである. 表 5.2-1 に各ノードの計算機構成を, 図 5.2-1 に予測対象計算機のネットワーク構成を示す.

表 5.2-1 予測対象計算機の各ノードのスペック

Node 数	128
CPU	Intel Pentium4 2.4GHz
L2 Cache	512Kbyte
Memory	SDRAM 1Gbyte
Network	1000BaseTX Ethernet
Network Switch	NETGEAR Gigabit Switch GS516T 1 台 NETGEAR Gigabit Switch GS524T 8 台
Compiler	PGI ver5.1
Compiler Option	-fast
MPI Library	MPICH ver 1.2.5

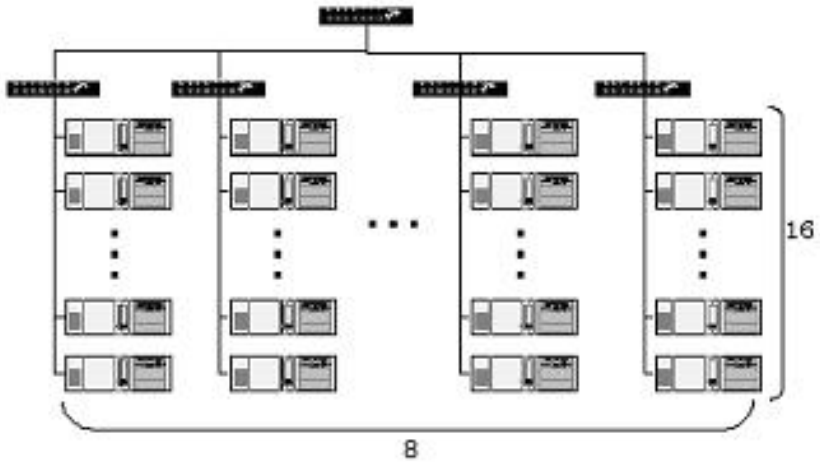

図 5.2-1 予測対象計算機のネットワーク構成

図 5.2-1 に示すように, 予測対象計算機は, ノード 16 台を 1 つのスイッチ (モデル GS516T) に接続し, ノード 16 台をまとめた 8 つのスイッチ (モデル GS524T) を, 上流で 1 つのスイッチにカス

ケード接続している．また，ノードとスイッチ間，スイッチとスイッチ間の接続には，全て 1000BaseTX の Ethernet を利用している．

5.3 実行時間の予測結果

本節では, MPIETE2 を用いて, NPB2.4 の EP, CG, FT, MG の実行時間を予測し, プログラム毎の予測実行時間を示す. なお, 予測対象とする区間は, それぞれのプログラムが性能評価の対象としている区間とし, 実行時間予測誤差は, 以下の式より求める. 単純に, 実際の実行時間に対する予測実行時間の割合としないのは, 計算部分, 通信部分の差分が互いに相殺しない予測誤差を求めるためである.

$$\text{予測誤差} = \frac{\left|\text{実際の計算時間} - \text{予測計算時間}\right| + \left|\text{実際の通信時間} - \text{予測通信時間}\right|}{\text{実際の実行時間}} \times 100 \ [\%]$$

5.3.1 EP の実行時間予測結果

EP 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ベンチマークである．EP は並列計算に適しており，この問題を解く際に通信をほとんど必要としないことから，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行性能を示すものと言える．通信は，Double Precision 型でデータサイズ 1 のデータを 2 回と，データサイズ 10 のデータを 1 回送っているのみである．

図 5.3-1，表 5.3-1 に，EP の予測実行時間と実際の実行時間を示す．

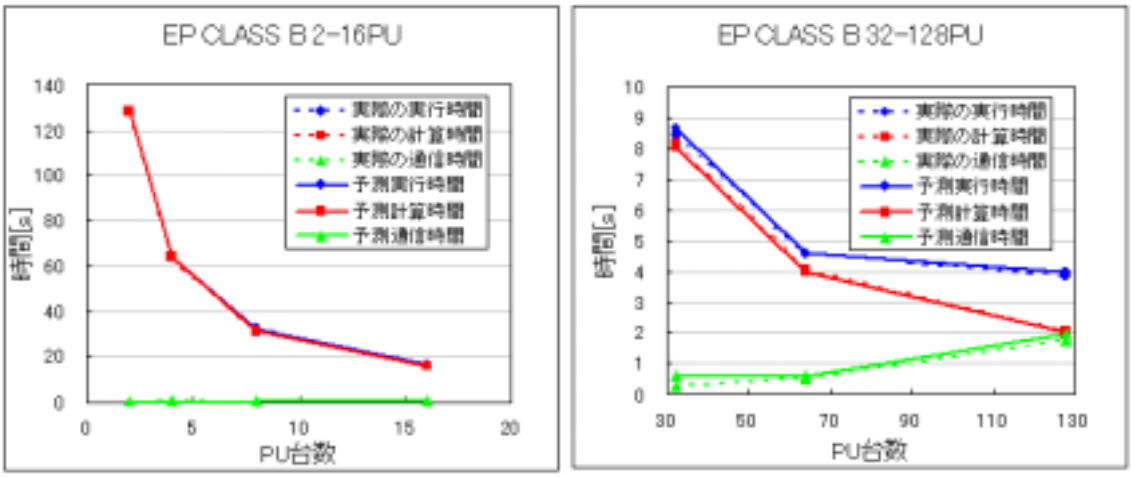

図 5.3-1 EP CLASS B の実際の実行時間と予測実行時間

表 5.3-1 EP CLASS B の実際の実行時間と予測実行時間

PU 台数		2	4	8	16	32	64	128
計算部分	実際	128.047	63.940	31.927	15.995	8.205	4.054	2.062
	予測	128.471	63.944	31.312	15.885	8.070	4.004	2.003
通信部分	実際	0.000	0.294	0.232	0.268	0.268	0.537	1.793
	予測	0.002	0.003	0.700	0.702	0.602	0.602	1.977
全体	実際	128.048	64.234	32.159	16.263	8.473	4.591	3.855
	予測	128.472	63.946	32.012	16.587	8.672	4.606	3.979
予測誤差[%]		0.33	0.46	3.37	3.35	5.54	2.49	6.31

単位[s]

表 5.3-1 から，EP の予測誤差はすべての実行 PU 台数で 7%以内である．また，図 5.3-1 から分かるとおり，最大性能を示す PU 数の予測が可能である．特に 128PU の実行で，通信の性能低下を予測できている．

5.3.2 CG の実行時間予測結果

CG は, 共役勾配法を用いて, 大規模で正値対称な疎行列の最小固有値の近似値を算出するベンチマークである. CG の通信では, PU 台数が増えると 1 回あたりの送受信メッセージサイズは減る. しかし逆に通信回数が増え, PU 毎では通信の総メッセージサイズは一定である. ただし, PU 毎では総メッセージサイズは一定だが, PU 台数が増えるためにプログラム全体での送受信メッセージサイズは増加していく.

図 5.3-2, 表 5.3-2 に, CG の予測実行時間と実際の実行時間を示す.

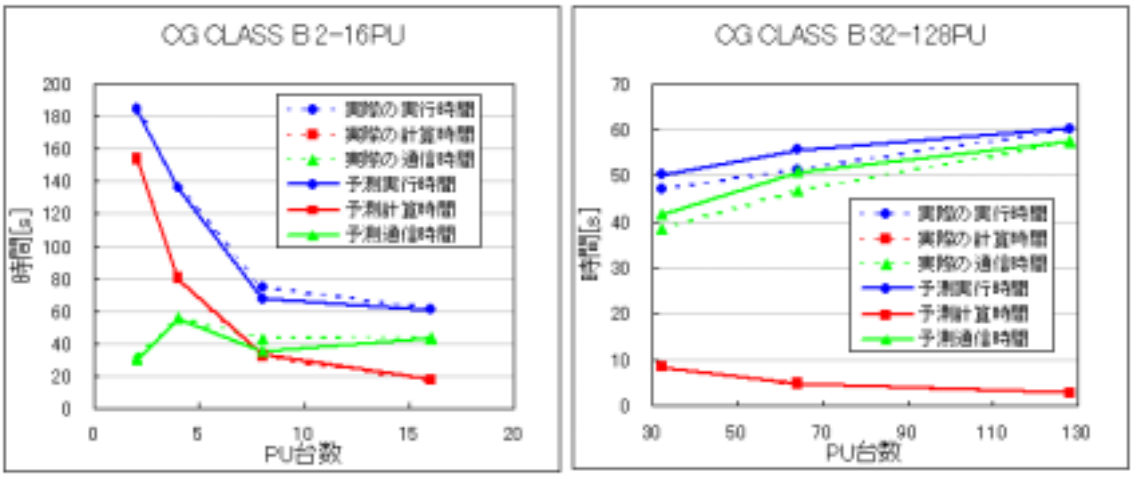

図 5.3-2 CG CLASS B の実際の実行時間と予測実行時間

表 5.3-2 CG CLASS B の実際の実行時間と予測実行時間

PU 台数		2	4	8	16	32	64	128
計算部分	実際	128.047	63.940	31.927	15.995	8.205	4.054	2.062
	予測	128.471	63.944	31.312	15.885	8.070	4.004	2.003
通信部分	実際	0.000	0.294	0.232	0.268	0.268	0.537	1.793
	予測	0.002	0.003	0.700	0.702	0.602	0.602	1.977
全体	実際	128.048	64.234	32.159	16.263	8.473	4.591	3.855
	予測	128.472	63.946	32.012	16.587	8.672	4.606	3.979
予測誤差[%]		1.35	0.84	13.57	3.15	7.05	8.39	0.46

単位[s]

表 5.3-2 より, CG の予測誤差はすべての PU 台数で 14%以内である. 図 5.3-2 から分かるとおり, 最大の実行性能を示す PU 数の予測が可能である.

5.3.3 FT の実行時間予測結果

FT は, 三次元偏微分方程式を FFT および逆 FFT を用いて解くベンチマークである. FT のベンチマーク測定区間での通信は, 集団通信関数 MPI_Alltoall を行っている. PU 数の増加に伴い, 各 PU のメッセージサイズは減るが, PU 間の通信回数が増加するため, 総メッセージサイズは一定である.

図 5.3-3, 表 5.3-3 に, FT の予測実行時間と実際の実行時間を示す.

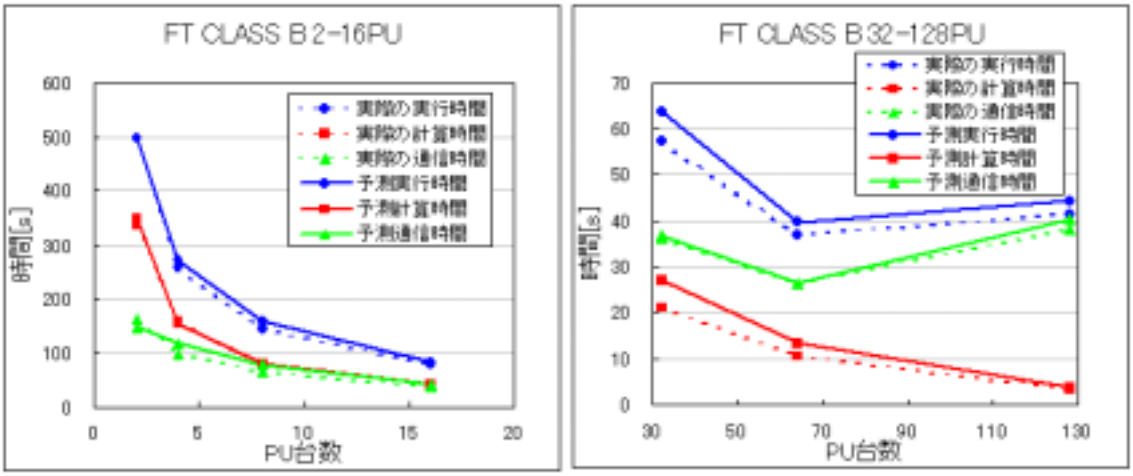

図 5.3-3 FT CLASS B の実際の実行時間と予測実行時間

表 5.3-3 FT CLASS B の実際の実行時間と予測実行時間

PU 台数		2	4	8	16	32	64	128
計算部分	実際	337.277	158.480	80.015	40.372	21.052	10.438	3.117
	予測	348.906	154.619	79.945	41.379	26.913	13.185	3.880
通信部分	実際	162.288	98.329	65.361	37.575	36.163	26.198	38.155
	予測	146.972	116.158	77.691	42.013	36.827	26.602	40.347
全体	実際	499.565	256.809	145.375	77.947	57.215	36.636	41.272
	予測	495.878	270.777	157.636	83.392	63.740	39.787	44.226
予測誤差[%]		5.39	8.45	8.53	6.99	11.40	8.60	7.16

単位[s]

表 5.3-3 より, FT の予測誤差はすべての PU 台数で 12%以内である. 図 5.3-3 から分かるとおり, 128PU の実行時で通信性能の低下を予測し, 最大の実行性能を示す PU 台数の特定が可能である.

5.3.4 MG の実行時間予測結果

MG は, 三次元ポアソン方程式を, 簡略化したマルチグレッド法で解くベンチマークである. MG の通信は 1 対 1 が主であり, 階層的なグリッド上で, 隣接した PU 同士が通信を行う.

図 5.3-4, 表 5.3-4 に, MG の予測実行時間と実際の実行時間を示す.

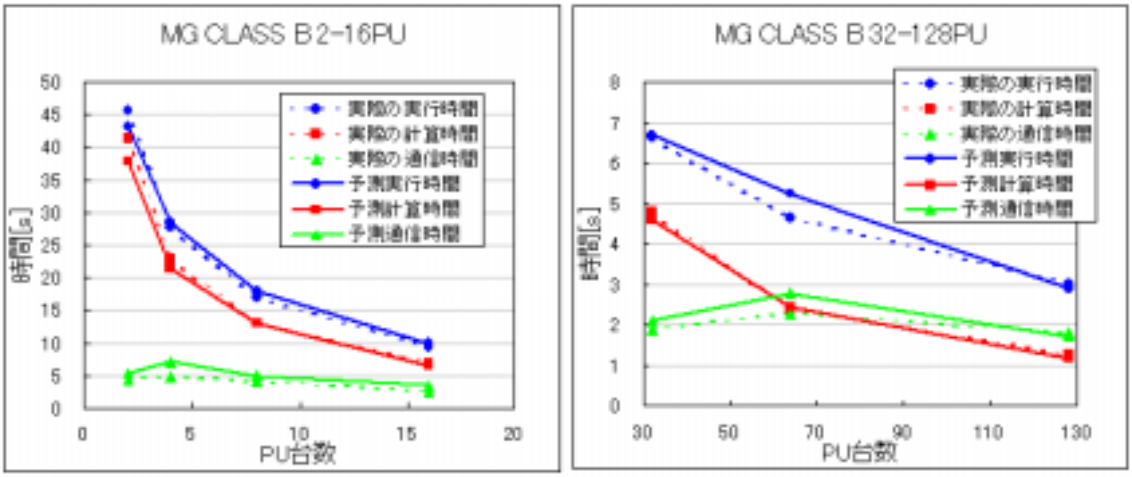

図 5.3-4 MG CLASS B の実際の実行時間と予測実行時間

表 5.3-4 MG CLASS B の実際の実行時間と予測実行時間

PU 台数		2	4	8	16	32	64	128
計算部分	実際	41.298	22.994	12.961	6.909	4.762	2.357	1.251
	予測	37.874	21.438	12.914	6.574	4.595	2.438	1.171
通信部分	実際	4.402	4.686	3.930	2.426	1.864	2.289	1.772
	予測	5.277	6.967	4.953	3.358	2.093	2.779	1.728
全体	実際	45.700	27.681	16.890	9.335	6.626	4.645	3.024
	予測	43.152	28.405	17.867	9.932	6.689	5.217	2.899
予測誤差[%]		9.41	13.87	6.34	13.56	5.98	12.31	4.14

単位[s]

表 5.3-4 より, MG の予測誤差はすべての PU 台数で 14%以内である. 図 5.3-4 から分かるとおり, 最大の実行性能を示す PU 数の予測が可能である.

5.4 予測に必要な処理時間

本節では, MPIETE2 を用いた, 予測に必要な処理時間を示す.
予測に必要な処理時間は, 以下に示す 3 つの処理時間からなる

(1) MPIETE2 が行う, 基礎データ取得プログラムの生成時間
(2) 予測対象計算機上で行う, 基礎データ取得プログラムの実行時間
(3) MPIETE2 が行う, 基礎データからの実行時間算出時間

上記 3 つの処理時間のうち, (1), (3)に関しては, 表 5.4-1 に示す計算機上の実行で, 処理時間は 1 秒未満であるため, 本稿では割愛する.

表 5.4-1 MPIETE2 の処理を行う計算機構成

CPU	Pentium M 1GHz
L2 Cache	1Mbyte
Memory	SDRAM 256Mbyte
Compiler	Visual C++ .NET 2003
Compiler Option	/O2 /Ot /G7

本節ではこれより, (2)にかかる処理時間を示す.
(2)の基礎データ取得プログラムの実行時間とは, 予測対象計算機上で行う, 予測対象区間の予測に必要な処理時間のことであり, 以下の3つの処理時間からなる.

- 計算ブロック基礎データ取得プログラムの実行時間
- 通信ブロック基礎データ取得プログラムの実行時間
- ブロック実行回数取得プログラムの実行時間

以下, 表 5.4-2 に基礎データ取得プログラムの実行時間を, 図 5.4-1 に実際の実行時間と予測に必要な処理時間の比較を示す.

表 5.4-2 基礎データ取得プログラムの実行時間

Prog.	基礎データ取得プログラム実行時間		
	計算ブロック基礎データ	通信ブロック基礎データ	ブロック実行回数
EP	23.49	4.79	0.17
CG	108.81	13.12	31.85
FT	90.94	120.11	10.48
MG	0.79	6.68	7.52

単位[s]

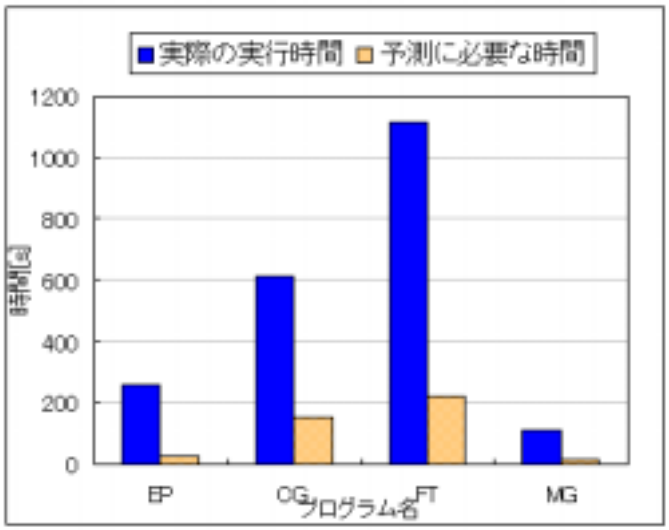

図 5.4-1 2-128PU の実際の総実行時間と予測に必要な処理時間の総和

図 5.4-1 より, 実際の実行時間と比較して, EP では約 1/10, CG では約 1/4, FT では約 1/5, MG では約 1/7 の時間で, 最大性能を示す PU 台数の特定が可能であり, MPIETE2 を用いた予測手法の有効性を示すことができた.

5.5 実行時間予測結果のまとめ

MPIETE2 を用いて, NPB2.4 EP, CG, FT, MG の実行時間を予測し, 実際の実行時間と比較したところ, 以下のことがわかった.

- 予測誤差は, 2~128PU すべてで 14%以内であった
- 予測には, 約 1/4 の時間がかかる
- すべてのプログラムで, 最大の実行性能を示す PU 数の特定が可能である

第 2 章で示した関連研究では, [10]を除くと, 予測に必要な処理時間は, 実際の実行時間以上かかる. また, [10]では通信コンテンションの発生による実行性能の低下を予測することができない. 上記した予測結果は, これら問題点を解決するものであり, 本手法を実装した MPIETE2 の有効性を示すことができた.

第6章 おわりに

本章では, 本論文のまとめを示し, 今後の研究方針について述べる.

本稿では, 通信性能の低下による並列プログラムの実行性能の低下を予測し, 最大の実行性能を示す PU 数を特定する手法を提案した. また, 提案した手法を用いて MPI プログラムの実行時間を予測する MPIETE2 を示し, MPIETE2 を NPB2.4 の EP, CG, FT, MG を用いて検証を行った. 検証の結果, 問題サイズ CLASS B, 2~128PU の実行では, 実際の実行時間と比較して, 14%以内の予測誤差で予測可能であり, 予測に必要な処理時間は, 1/4 程度の時間であった. また, 4 つのプログラムすべてで最大の実行性能を示す PU 数を予測するが可能であり, 第 2 章で述べた従来研究に対する優位性を示すことができた.

最後に, 今後の研究方針を以下に挙げる.

- 提案手法における, ループ回数削減方法の確立
- MPIETE2 の C 言語への対応

前者に関しては, 本稿では 1/10 の回数に静的に決めていた. この方法では, 予測手法を適用できないプログラムが存在する可能性がある. 今後はより多くのプログラムに本手法を適用し, 検証を重ねていく. 後者に関しては, MPICH は C 言語からも利用可能なライブラリである. 今後は Fortran だけでなく, C 言語を利用した MPI プログラムの予測を可能にし, MPIETE2 の汎用性の向上を図る.

参考文献

[1] Marc Snir,Steve Otto,Steven Huss-Lederman,David Walker,Jack Dongarra: “MPI -The Complete Reference,The MPI Core second edition ”The MIT Press,1998.

[2] K. Kubota, M. Sato, K. Itakura, T. Boku : “Accuracy of fast performance prediction by instrumentation tool EXCIT" Proc. of HPC Asia'98, Singapore, pp.1031-1038, 1998.

[3] Taisuke Boku,Tomoaki Harada,Takashi Sone, Hiroshi Nakamura,and Kisaburo Nakazawa: “INSPIRE:A general purpose network simulator generating system for massively parallel processors ”,Proc.of PERMEAN ‘95, pp.24-33,1995.

[4] Kazuto Kubota, Ken’ichi Itakura, Mitsuhisa Sato, Taisuke Boku; “Practical Simulation of Larege-Scale Parallel Programs and Its Performance Analysis of the NAS Parallel Benchmarks”, Proc. of Euro-Par, pp.244-254 , 1998.

[5] Maurice Yarrow and Rob Va der Wijngaart:”Communication Improvement for the LU NAS Parallel Benchmark:A Model for Efficient Parallel Relaxation Schemes ”,NAS Technical Report NAS-97-032,1997.

[6] Edward Rothberg,Jaswinder Pal Singh,and Anoop Guputa: “Working Sets,Cache Sizes,and Node Granularity Issues for Large-Scale Multiprocessors ”,Proc.of ISCA ’93,pp.14-25,1993.

[7] David Bailey, Tim Harris, William Saphir, Robva der Wijngaart, Alex Woo, and Maurice Yarrow: “The NAS Parallel Benchmarks 2.0,” NASA Ames Research Center Report,NAS-05-020,1995.

[8] U. Legedza, W. E. Weihl: “Reducing Synchronization Overhead in Parallel Simulation,” 10th Workshop on Parallel and Distributed Simulation PADS96, pp 86-95, 1996.

[9] Sundeep Prakash, Ewa Deelman, Rajive Bagrodia: “Asynchronous Parallel Simulation of Parallel Programs” IEEE Transactions on Software Engineering, 2000, vol. 26, pp. 385-400, 2000.

[10] 堀井洋, 岩渕寿寛, 山名早人:“MPI プログラムの簡易実行結果を用いた実行時間予測ツー

ル MPIETE の評価", 情報処理学会研究報告 (HPC), Vol.2004, No.20, pp.55-60, 2004
[11] Gropp, W., and Lusk, E.: "Users guide for MPICH, a portable Implementation of MPI", Mathematics and Computer Science Division, Argonne National Laboratory, University of Chicago, 6 July 1998.

謝辞

本研究は, 筆者が早稲田大学大学院修士課程在学中になされたものである.

本研究を進めるにあたり, 終始あたたかいご指導を賜りました, 山名早人教授に心から感謝の意を表します.

研究業績

種類別	題名	発表年月日	発表掲載誌	連名者
1. 講演 (研究会等)				
○(1)	MPIETE2:MPI プログラム実行時間予測ツール MPIETE の通信時間予測誤差に関する改良	2005 年 3 月	情処学研報 HPC2005 (掲載予定)	岩渕 寿寛 杉田 秀 山名 早人
(2)	MPI プログラム実行時間予測ツール MPIETE の評価	2004 年 3 月	情処学研報 HPC2004	堀井 洋 岩渕 寿寛 山名 早人
○(3)	MPI プログラムの簡易実行による実行時間予測手法の評価	2003 年 8 月	情処学研報 HPC2004	岩渕 寿寛 堀井 洋 山名 早人
(全国大会等)				
○(4)	MPI プログラムの簡易実行による実行時間予測	2003 年 3 月	第 65 回情処全大	岩渕 寿寛 堀井 洋 山名 早人