社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report

2004－ARC－157　(10)
2004－HPC－ 97　(10)
2004／3／1

MPI プログラム実行時間予測ツール MPIETE の評価

堀井 洋† 岩渕 寿寛† 山名 早人‡

概要

本稿では、同じプログラムを様々な計算機環境で実行時間を予測する際、ある計算機環境での予測のために生成したデータを、異なる計算機環境での予測にも利用することが可能な MPI プログラム実行時間予測手法を提案する。また、提案手法を用いた MPI プログラム実行時間予測ツール MPI Execution Time Estimator (MPIETE)で、NAS Parallel Benchmarks (NPB) ver2.4 の実行時間の予測を行う。並列プログラムに対する、最適な計算機構成を決定するには、様々な計算機構成での実行時間を予測する必要がある。従来の並列プログラム予測手法は、プログラムの実行時間を予測するために、想定する計算機環境ごとに、予測に必要なパラメータを生成する必要がある。我々の提案する予測手法は、想定する計算機環境ごとに生成するパラメータと、予測対象となるプログラムごとに生成するパラメータに分けることで、短時間で同じプログラムを様々な計算機環境で実行時間を予測することが可能である。提案手法を用いて、NPB ver2.4 の EP、CG、LU の CLASS B の実行時間を予測したところ、16PU 以内では誤差 10%以内で予測可能であった。また、32PU 以上の予測では、通信部分を除いた計算部分の実行時間は 10%以内の予測誤差で予測可能であるのに対し、通信時間は、30%以上の予測誤差が生じた。また、予測に必要な PC は、予測対象 PC 2台と一般的な PC 1台である。これらの計算機を用いて予測することにより、16PU 以内の予測であれば実際の実行時間の 1/10程度、128PU 時の予測でも実際の実行時間の 10倍程度の時間で予測できることがわかった。

MPIETE: An Execution Time Estimator for MPI Programs

Hiroshi HORII†, Toshihiro IWABUCHI†, Hayato YAMANA‡

Abstract In this paper, we propose the MPI Execution Time Estimator (MPIETE), the execution time estimation tool for MPI programs, helping you to choose the best suited computing platform to execute a MPI program. Conventional execution time estimation schemes are not able to model a computing platform or a MPI program perfectly, which results in no reusable of any parameters of both the computing platform and the MPI program. On the contrary, the proposed scheme enables to reuse all the parameters of both the computing platform and the MPI program even for the estimation on another computing platform or on another MPI program. Therefore the proposed scheme estimates the execution time faster than the conventional schemes. We have evaluated the proposed scheme using EP, FT and LU from NAS Parallel Benchmarks 2.4. As the results, the error ranges of the computation time between the real execution time and the estimated time are less than 10%. However, the error ranges of the communication time are over 30%. The proposed scheme requires only two PC's of the target platform, and any one PC. Using the PC's, the execution time of the estimation for 1-16PU is 1/10 times smaller than the actual execution time.

1. はじめに

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。一般的に、PC クラスタ、Grid テストベッドといった並列計算環境は、複数のユーザ、複数の機関が共有して利用することが多く、ユーザがそれらの計算機を利用する際は、あらかじめプログラムの実行時間を予測し、並列計算環境を利用する時間を指定する必要がある。また、ユーザにとって最適な並列プログラムの計算機構成を求めるためには、試行錯誤的に、何度も同じプログラムの異なる計算機構成での実行時間を予測する必要がある[1]。

従来、MPI プログラム[2]の実行時間の予測手法として、EXCIT&INSPIRE[3]や、LAPSE[4]、MPI-SIM[5]といったシミュレータを用いた手法が提案されている。[3]の手法では、EXCIT を用いて、アセンブラコードレベルのトレースを行い、計算時間の予測を行う。また INSPIRE を用いて、ネットワークシミュレータを生成し、通信時間の予測を行う。キャッシュヒット率の予測や、ネットワークのレイテンシ・バンド幅が変化した際の通信時間を予測することが可能である。また、LAPSE[4]、MPI-SIM[5]のシミュレータを使用した予測手法では、プログラムの各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。計算ブロック実行時間情報から、通信関数を呼ぶタイミングを予測することで、非同期な通信の実行時間を予測することが可能である。しかしいずれの予測手法も、実際の実行時間に対して、数倍から数十倍の処理時間を必要とし、実行に日単位を必要とする大規模プログラムの実行時間予測には適さない。また、同じプログラムの異なる計算機環境で実行時間を求めるには、計算機環境毎に予測に必要な測定が必要となる。

一方、シミュレーションを行わずに、ソースプログラムを静的に解析することで、プログラムの挙動を解析する手法[6][7][8]が提案されている。しかし静的な解析のみでは、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、[3] [4] [5]の手法と比べ、予測精度が悪い。

本稿では、これらの問題に対して、一度生成した予測に必要なデータを、次の予測にも利用することが可能な MPI プログラム実行時間予測手法を提案し、提案手法を用いて、MPI プログラムの実行時間を予測するツール MPIETE を示す。これより 2 節に提案手法について、3 節に MPIETE について示し、4 節に MPIETE を用いて NAS Parallel Benchmarks ver2.4 の実行時間の予測結果を示し、5 節にまとめを示す。

2. 提案手法

同じプログラムを様々な計算機環境で実行時間を予測する際、ある計算機環境での予測のために生成したデータを、異なる計算機環境での予測にも利用することが可能な MPI プログラム実行時間予測手法を提案する。

提案する予測手法は、MPI 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、基礎データ(計算基礎データ、通信基礎データ、プログラム基礎データ)を用いて、それぞれのブロックごとに実行時間の予測を行う。

2.1 提案手法の予測の流れ

図 1 に計算機 A とネットワーク構成 N[1]で構成される計算機上での、プログラム P の実行時間を予測する流れを示す。なお、計算機 A とネットワーク構成 N で構成される計算機とは、計算機 A を Processing Unit(PU)とし、PU 間をネットワーク構成 N で接続する計算機を示している。

†早稲田大学理工学研究科 Graduate School of Science and Engineering, Waseda University
‡早稲田大学理工学部 School of Science and Engineering, Waseda University

[1]計算機間のスイッチ、ケーブルといった接続方法

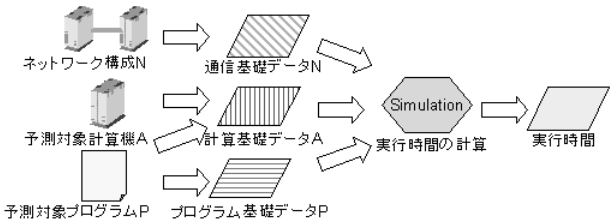

図 1 :計算機 A とネットワーク構成 N から構成される計算機の
プログラム P の実行時間予測の流れ

提案手法は、ネットワーク構成 N から通信基礎データ N を、計算機 A とプログラム P から計算基礎データ A を、プログラム P からプログラム基礎データ P を定義する。定義した基礎データをもとに、計算機 A とネットワーク構成 N で構成される計算機上での、プログラム P の実行時間を予測する。

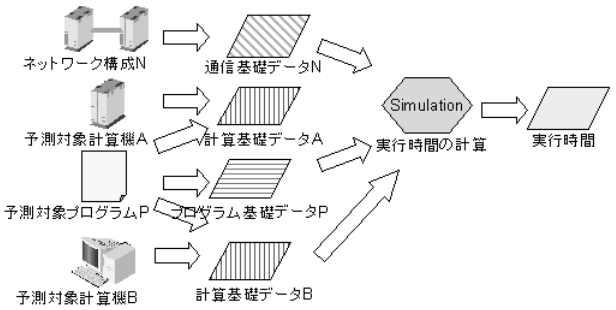

図 2 :計算機 A、B とネットワーク構成 N から構成される計算機での
プログラム P の実行時間予測の流れ

図 2 で示すように、計算機 B と計算機間のネットワーク構成 N で構成される計算機上でのプログラム P の実行時間の予測には、新たに定義する計算基礎データ B と共に、図 1 で示した通信基礎データ N とプログラム基礎データ P を用いることができる。

また、計算機 A 計算機 B を PU とし、PU 間をネットワーク構成 N で接続する、性能的に異なる PU からなる計算機上でのプログラム P の実行時間は、図 1 で示した計算基礎データ A、プログラム基礎データ P、通信基礎データ N と、図 2 で示した計算基礎データ B を用いることで、予測を行う。

上記のように提案手法は、予測を行うために定義した基礎データを次の予測に利用することができ、効率的な実行時間予測を行うことができる。

2.2 記号の定義

本提案手法で利用する記号の定義を、表 1 に示す。

2.3 プログラムのモデル化

提案手法では、予測対象プログラムをモデル化用プログラムに変換し、モデル化用プログラムからプログラムの実行時間を予測する単位となるブロックへの分割を行う。

2.3.1 モデル化用プログラムへの変換

提案手法では、通信時間を「メッセージを送受信する時間」と、「メッセージの送受信を待機する時間」に分けて予測する。MPI 通信関数の中には、メッセージの送受信と、送受信待ちを同時に行う関数があるため、それらの MPI 通信関数を宣言している文を、メッセージの送受信のみを行う MPI 通信関数を宣言する文と、メッセージの送受信待ちのみを行う MPI 通信関数を宣言する文に書き直す必要がある。具体的には、以下のように、予測対象プログラムに修正を施すことで、2.3.2 で行うモデル化のためのプログラムを生成する。

- 集団通信を行う MPI 通信関数を宣言する CALL 文を、宣言されている集団通信を行う MPI 通信関数の実際のインプリメントと同様に、1 対 1 メッセージ通信を行う MPI 通信関数を宣言する CALL 文と、DOWHILE 文、IF 文等の文の集合に変換する。
- 1 対 1 のメッセージを同期しながら送受信を行う MPI 通信関数が宣言されている CALL 文を、MPI の実際のインプリメントと同様に、1 対 1 のメッセージを非同期で送受信を行う MPI 通信関数を宣言するCALL 文と、メッセージの送受信完了待ちを行う MPI 通信関数が宣言されている CALL 文に変換する。

表 1 ：記号定義

N=プログラム内のブロック数
R=予測対象プログラムのグループサイズ
M(r)=予測対象プログラムを実行時、ランク r において、実行される延べブロック数 （0 ≦ r < R）
B(i)=ブロック ID i のブロック （ただし 1 ≦ i ≦ N）
Bo(r, i)=ランク r の i 番目に実行されるブロック （0 ≦ r < R、1 ≦ i ≦ M(r)）
T_comp(r, i) = ランク r における、B(i)を 1 回実行する際に必要な時間 （0 ≦ r < R、1 ≦ i ≦ N）
T_comm(r1, r2, size) = ランク r1 からランク r2 へ size バイトのメッセージを MPI_RECV 通信関数を利用して受信するために必要な時間[2] （ただし、0 ≦ r < R、1 ≦ i ≦ N）
Bo_from(r, i) = Bo(r, i)へメッセージ送信する Bo(r', j) （0 ≦ r < R、0 ≦ r' < R、1 ≦ i ≦ M(r)、1 ≦ j ≦ M(r')）
Bo_fromID(r, i)=Bo(r, i)へメッセージ送信する Bo(r', j)のブロック ID （0 ≦ r < R、0 ≦ r' < R、1 ≦ i ≦ M(r)、1 ≦ j ≦ M(r')）
Bo_fromRank(r, i)= Bo(r, i)へメッセージ送信する Bo(r', j)の r'の値 （0 ≦ r < R、0 ≦ r' < R、1 ≦ i ≦ M(r)、1 ≦ j ≦ M(r')）
Bo_fromIdx(r, i)=Bo(r, i)へメッセージ送信する Bo(r', j)の j の値 （0 ≦ r < R、0 ≦ r' < R、1 ≦ i ≦ M(r)、1 ≦ j ≦ M(r')）
Bo_wait(r, i)=Bo(r, i)が完了を待つ、メッセージ送受信を行う Bo(r, j) （0 ≦ r < R、1 ≦ j < i ≦ M(r)）
Bo_waitIdx(r, i)= Bo(r, i)が完了を待つ、メッセージ送受信を行う Bo(r, j)の j の値 （0 ≦ r < R、1 ≦ j < i ≦ M(r)）
T_start(r, i)= Bo(r, 0)の実行開始時刻から、Bo(r, i)の実行開始時刻までの予測実行時間 （0 ≦ r < R、1 ≦ i ≦ M(r)）
T_end(r, i)= Bo(r, 0)の実行開始時刻から、Bo(r, i)の実行終了時刻までの予測実行時間 （0 ≦ r < R、1 ≦ i ≦ M(r)）
T = 予測対象プログラム全体の予測実行時間

2.3.2 ブロックへの分割

2.3.1 で生成されたモデル化用プログラムを、以下のように、計算ブロック、通信ブロック、通信待ちブロックに分割する。

- 繰り返し[3]、条件分岐を行う文[4]や、GOTO 文、STOP 文、RETURN 文、END 文を含まない、実行される連続した文の集合を、計算ブロックとする。
- 1 対 1 のメッセージを非同期で送受信を行う MPI 通信関数[5]が宣言されている CALL 文を、通信ブロックとする。
- メッセージの送受信完了待ちを行う MPI 通信関数[6]が宣言されている CALL 文を、通信待ちブロックとする。

図 3 に、プログラムのモデル化の例をあげる。なお、図 3 のソースコード内の、……で示す文は、繰り返し、条件分岐を行う文や、GOTO 文、STOP 文、RETURN 文、END 文を含まない実行する文とする。また、変数名 rank の値は、ランクの値を示すものとする。

[2] MPI_RECV 関数を宣言した CALL 文が実行された時点で、メッセージの受信が開始されるものとする
[3] DO 文、DOWHILE 文、ENDDO 文
[4] IF 文、ELSEIF 文、ELSE 文、ENDIF 文
[5] 2 つ以内のランク間でのみ通信可能な通信関数
[6] すでに実行した通信関数が行うメッセージ通信の完了を待つ関数

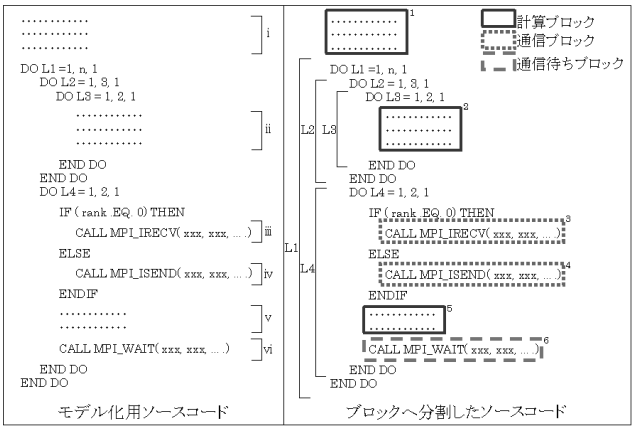

図 3 :プログラムのブロックへの分割例

i 、ii 、v は、繰り返し、条件分岐を行う文や、GOTO 文、STOP 文、RETURN 文、END文を含まない、連続した実行する文の集合のため、計算ブロックとする。また、iiiは、iiiで宣言されているMPI_IRECV関数が 1 対 1 のメッセージを非同期で受信を行うMPI 通信関数のため、通信ブロックとする。ivは、ivで宣言されているMPI_IRECV 関数が 1 対 1 のメッセージを非同期で受信を行うMPI 通信関数のため、通信ブロックとする。viは、viで宣言されている MPI_WAIT 関数が送受信完了待ちを行うMPI 通信関数のため、通信待ちブロックとする。

なお、全てのブロックには、固有の数字（ブロック ID）が振られるものとする[7]。本提案手法では、ブロックの総数をN とし、ブロック ID が i（1 ≦ i ≦ N）のブロックを B(i)とする。

2.4 基礎データ

MPI プログラムの実行時間予測に必要な、基礎データについて示す。

計算基礎データ T_comp(r, i)

予測対象計算機で、ランクr の B(i)を 1 回実行するために必要な時間（ただし、0 ≦ r < R、1 ≦ i ≦ N で、B(i)は計算ブロック）

通信基礎データ T_comm(r1, r2, size)

予測対象計算機で、ランクr1 からランクr2 へ、メッセージサイズ size バイトのメッセージを、MPI_RECV 関数を使用して受信するのに必要な時間[8]（ただし、0 ≦ r1 < R、0 ≦ r2 < R）

プログラム基礎データ Bo(r, i)

プログラム を実際に実行した際、ランクr で i 番目に実行するブロック（ただし、0 ≦ r < R、1 ≦ i ≦ M(r)）

プログラム基礎データの例として、図 3 で示したプログラムの、ランク0 とランク1 のプログラム基礎データを図 4 に示す。

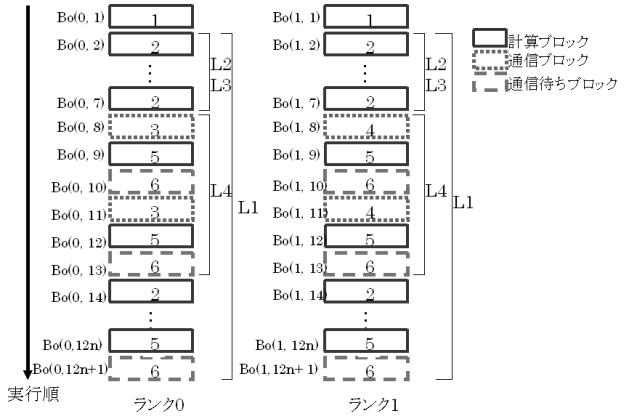

図 4： プログラム基礎データの例

2.5 プログラムの実行時間の予測方法

提案手法では、以下のような条件を仮定し、プログラム全体の実行時間を求める。

- 同じ計算ブロックの実行時間は、常に同じである
- 通信ブロックの実行時間は、常に 0 である[9]
- 同じサイズのメッセージを受信するに必要な時間は常に同じである
- メッセージを送信する通信待ちブロックの実行時間は、常に 0 である[10]

上記の条件で、プログラム全体の実行時間 T を以下のように求める。

まず、ランクr において、T_start(r, i)、T_end(r, i)を以下のように求める。（ただし、0 ≦ r < R、1 ≦ i ≦ M(r)）

◆ i = 1 のとき、T_start(r, 0)を式 1 より求める。

$$T\_start(r, 1) = 0 \quad \cdots(式1)$$

◆ i ≠0 のときは、T_start(r, i)を式 2 より求める。

$$T\_start(r, i) = T\_end(r, i-1) \quad \cdots(式2)$$

◆ Bo(r, i)が計算ブロックの時は、T_end(r, i)を式 3 より求める。

$$T\_end(r, i) = T\_start(r, i) + T\_comp(r, B\_id(r, i)) \quad \cdots(式3)$$

◆ Bo(r, i)が通信ブロックの時は、通信ブロックの実行時間は 0 のため、T_end(r, i)を式 4 より求める。

$$T\_end(r, i) = T\_start(r, i) \quad \cdots(式4)$$

◆ Bo(r, i)が通信待ちブロックで、Bo_from(r, i)がメッセージを送信するブロックの場合は、メッセージを送信する通信ブロックの実行時間は、常に0 であるため、T_end(r, i)を式 5 より求める。

$$T\_end(r, i) = T\_start(r, i) \quad \cdots(式5)$$

◆ Bo(r, i)が通信待ちブロックで、Bo_from(r, i)がメッセージを受信するブロックだった場合は、T_end(r, i)を式 6 より求める。

$$T\_end(r, i) = \max(T\_start(r, i), T\_end(r, Bo\_waitIdx(i)) + T\_comm(r, B\_fromRank(r, Bo\_waitIdx(i)), B\_size(B\_fromRank(r, Bo\_waitIdx(i)), B\_fromIdx(r, Bo\_waitIdx(i)))) \quad \cdots(式6)$$

次に、プログラム全体の実行時間 T を式 7 より求める。

$$T = \max(T\_end(0, M(0))), T\_end(1, M(1))), \ldots, T\_end(r, M(r))) \quad \cdots(式7)$$

2.6 提案手法の予測対象プログラム

本提案手法の予測対象とするプログラムは、以下の条件を満たすプログラムである。

- Fortran で記述されている
- MPI 関数を利用した通信以外の通信を行わない
- プログラム を実行した際、各ブロックの実行回数が常に同じである

3. MPIETE

2 節で示したMPI プログラム実行時間予測手法を実現するMPI プログラムの実行時間を予測するツール MPI Execution Time Estimator（MPIETE）について示す。

3.1 MPIETE の機能

MPIETE は、以下の機能を持つ。

- MPI プログラム より、計算基礎データ取得プログラムとプログラム基礎データ取得プログラムを生成する
- ユーザが指定した通信基礎データを取得する、通信基礎データ取得プログラムを生成する
- 計算基礎データファイル、通信基礎データファイル 、プログラム基礎データファイル を用いて、プログラムの実行時間を予測する

なお、上記で示した 3 つの基礎データ取得プログラムは、全て Fortran とMPI 関数を利用したプログラムである。

3.1.1 計算基礎データ取得プログラムの生成

計算基礎データ取得プログラムは、（1）計算基礎データ取得用ソースコードの挿入後、（2）計算基礎データ取得プログラムの簡略化を行うことで、自動生成される。

計算基礎データ取得ソースコードの挿入

計算基礎データを取得するため、予測する MPI プログラムファイルを 2.3.1 によってモデル化用プログラムに変換した後、以下に示す値を求

---

[7] 図 3 においては、右図のそれぞれのブロックに示されている数字

[8] ランクr2 において、MPI_RECV 関数を宣言した CALL 文が実行された時点で、メッセージの受信が開始されるものとする

[9] 通信ブロックは、通信を開始するだけなので、実行時間は 0 とすることができる

[10] 実際には、OS の通信バッファに空きがない場合は、送信することはできなず、送信待ち時間が発生する

めるための、ソースコードを挿入する。

- 計算基礎データ取得ソースコードを実行した際の各計算ブロックの総実行回数と総実行時間

上記で示した値を求めるソースコードを、予測対象プログラムの以下で示す部分に挿入する。

- 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
- 当該計算ブロックのみをループボディとして持つ DO ループが存在しない場合は、計算ブロックの前後

図 3で示したプログラムの場合、計算ブロック1は計算ブロック1の前後に、計算ブロック2はループL2の前後に、計算ブロック5は計算ブロック 5 の前後に、各計算ブロックの総実行回数と総実行時間を求めるソースコードを挿入する。

なお、計算基礎データ取得プログラムは、計算ブロックごとに、計算ブロックの総実行時間から計算ブロック総実行回数を割った値を、計算基礎データファイルとして出力する。

計算基礎データ取得プログラムの簡略化

計算基礎データ取得プログラムの実行を簡略化するために、2.3.1 によって生成された計算基礎データ取得プログラムに、以下で示す文以外の文を、コメントアウトする。

- 当該計算ブロックのみをループボディとして持つ DO ループ内の文
- 当該計算ブロックのみをループボディとして持つ DO ループ内で参照されている配列変数の、インデックスとして参照される変数を、定義する文
- コメントアウトされることによって、各ブロックの総実行回数が変化する文
- コメントアウトされることによって、プログラム中で行われる MPI 通信関数による通信の送り先ランク、受信先ランクが変化する文

上記に示す以外の文をコメントアウトした計算基礎データ取得プログラムに対し、以下の修正も施す。

- 予測対象区間を実行した際に実行される、当該計算ブロックのみをループボディとして持つ DO ループ以外の DO ループの繰り返し回数を、最大 2 回とする。

なお、繰り返し回数を変更することにより、プログラム中で行われる MPI 通信関数による通信の送り先ランク、受信先ランクが変化する場合、もしくは、通信ブロック、通信待ちブロックの総実行回数が変化する場合、繰り返し回数の変更を行わない。

3.1.2 通信基礎データ取得プログラムの生成

通信基礎データ取得プログラムは、ユーザによって入力された、測定するメッセージサイズの上限と増加幅をもとに自動生成される。

通信基礎データ取得プログラムは、MPI_SEND 関数、MPI_RECV 関数を利用して、ランク0 からランク1 へ、メッセージの通信を行い、通信するメッセージサイズごとの MPI_RECV 関数を実行するのに必要な時間を測定する。通信するメッセージのサイズは、0 からユーザが指定した増加幅ずつ増加させ、通信するメッセージのサイズが、ユーザが指定した上限を超した時点で、プログラムは終了する。

3.1.3 プログラム基礎データ取得プログラムの生成

プログラム基礎データ取得プログラムは、(1) プログラム基礎データ取得用ソースコードの挿入後、(2) プログラム基礎データ取得プログラムの簡略化を行うことで、自動生成される。

プログラム基礎データ取得ソースコードの挿入

MPI プログラムファイルを、2.3.1 によってモデル化用プログラムに変換した後、以下に示す値をプログラム基礎データファイルに出力するための、ソースコードを挿入する。

- 実行しているブロックのブロック ID
- 実行している通信ブロック、通信待ちブロックで宣言されている MPI 通信関数で参照されている変数の値

上記のソースコードを挿入したプログラムを実行することにより、ランクごとのプログラム基礎データファイルが生成される。

プログラム基礎データ取得プログラムの簡略化

プログラム基礎データ取得プログラムの実行を簡略化するために、以下で示す文以外の文を、コメントアウトする。

- コメントアウトされることによって、各ブロックの総実行回数が変化する文
- コメントアウトされることによって、プログラム基礎データが変化する文

3.1.4 MPI プログラムの実行時間の算出

MPIETE は、2.5 で示した方法で、ユーザが入力する以下に示す基礎データファイルから、計算基礎データ、通信基礎データ、プログラム基礎データを読み込み、MPI プログラムの実行時間を求める。

- 全てのランクの計算基礎データファイル
- 1 つ以上の通信基礎データファイル
- 全てのランクのプログラム基礎データファイル

3.2 実行時間の予測の流れ

MPIETE を用いて MPI プログラムの実行時間を予測するには、ユーザは、(1) MPIETE を用いて各基礎データ取得用プログラムを取得し、(2) 生成した各基礎データ取得プログラムを実行することで各基礎データファイルを取得し、(3) 各基礎データファイルを MPIETE に入力することで、MPI プログラムの実行時間を予測する。

3.2.1 基礎データ取得プログラムの取得

ユーザは、MPIETE に、予測対象プログラムのファイルパスと、実行時間を予測する区間を入力することで、計算基礎データ取得プログラム、プログラム基礎データ取得プログラムを取得する。なお、予測の対象とする区間とは、プログラム内の MPIETE が実行時間を予測する区間を示す。予測する区間の開始行、終了行は、実際に予測対象プログラムを実行時、以下の条件を満たさなくてはならない。

- 開始行、終了行で指定された文は、1 回のみ実行される文である
- 終了行は、開始行より先に実行してはならない

通信基礎データ取得プログラムは、ユーザが以下の入力を行うことで、MPIETE により生成される。

- 測定するメッセージサイズの上限（バイト単位）
- 測定するメッセージサイズの増加幅（バイト単位）。

3.2.2 基礎データファイルの取得

3.2.1 によって生成した基礎データ取得プログラムを実行することで、ユーザは各基礎データファイルを取得する。

計算基礎データ取得プログラムの実行

計算基礎データ取得プログラムは、**予測対象計算機 1PU 上で、**3.3 で示すトークン型 MPI 通信方式で実行することで、計算基礎データファイルを出力する。

通信基礎データ取得プログラムの実行

通信基礎データ取得プログラムは、**予測対象計算機の2PU 上で**実行することで、通信基礎データファイルを出力する。

プログラム基礎データ取得プログラムの実行

プログラム基礎データ取得プログラムは、**任意の計算機上で**実行することで、プログラム基礎データファイルを出力する。

3.3 トークン型 MPI プログラム実行方式

MPIETE が計算基礎データ取得プログラムを実行するための、トークン型 MPI プログラム実行方式について提案し、その実行方法を示す。

トークン型 MPI 実行方式は、以下に示すように、トークンを持つランクのプロセスのみが、計算を行う実行方式である。

- MPI_INIT 関数が呼ばれたら、ランク0 にトークンを渡し、ランク0 以外のプロセスはトークン待ち状態にする。
- トークンを受け取ったランクのプロセスは計算を実行する。
- MPI の送受信関数が実行された場合、送受信するメッセージのデータ（送信元、送信先、タグ）をトークンに加える。
- MPI の送受信待ち関数が実行された場合、送受信を行うメッセージのデータがトークンに存在したら、トークンから受信を行うメッセージのデータを削除する。受信を行うメッセージのデータがトークンに存在しなかったら、次のランク[11]にトークンを渡す。

トークン型 MPI 実行方式で MPI を実行した例を、図 5 に示す。

[11]次のランクとは、実行しているランクに 1 を加えた数のランクであり、実行しているランクに 1 を加えた数がグループサイズと同じ場合は、次のランクは 0 となる

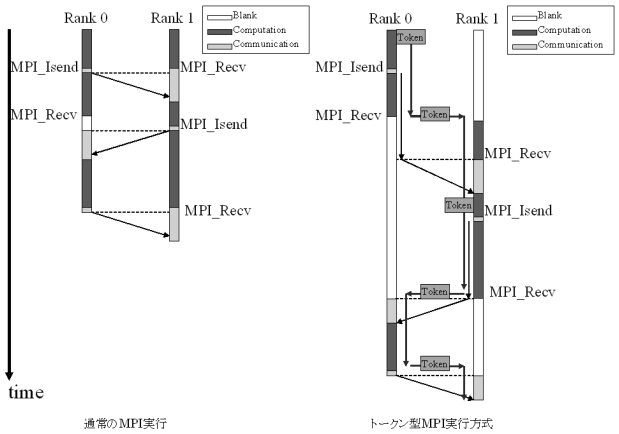

図 5 :通常の MPI 実行と、トークン型 MPI 実行方式

図 5では、同一の MPI プログラムを、通常の MPI 実行と、トークン型 MPI 実行方式で実行した場合の、計算、通信、受信待ちの時間の推移を示している。トークン型 MPI 実行方式では、常にトークンを持つプロセスのみが計算の実行を行っているため、同時に複数のプロセスで計算が行われることはない。

なお、トークン型 MPI プログラム実行方式は、MPIETE が提供するライブラリに定義されている MPI 関数を使用することで、実行することが可能である。

4. 提案手法の評価

3 節で示した、MPIETE を用いて、NAS Parallel Benchmark (NPB) ver2.4 の実行時間を予測することで、提案手法の評価を行う。

本提案手法の検証には、MPIETE と NPB ver2.4 のベンチマークを用いて、実際の実行時間と予測実行時間の比較、実際の実行時間と予測に必要な時間の比較を行う。なお、比較に利用したプログラムは、NPB ver2.4 の 8 つのプログラムのうち、カーネルベンチマークの EP、CG と、アプリケーションベンチマークの LU の、3 つのプログラムである。それぞれのプログラムの実行時間予測対象は、それぞれのベンチマークが性能評価を求める対象となる区間とし、問題サイズW、A, B の、1 から128PU 台で実行時の実行時間を求め、検証を行った。

本章では、4.1 に NPB ver2.4 について、4.2 に予測対象とする計算機構成について、4.3 に NPB ver2.4 の実行時間の予測結果について示し、4.3.2 に考察を示す。

4.1 NAS Parallel Benchmarks ver2.4

NAS Parallel Benchmarks (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、MPI を利用した Fortran または C で記述された、5 つのカーネルベンチマークと、3 つのアプリケーションベンチマークで構成されている。それぞれのベンチマークは、Make 時にプロセス数を 2^n、もしくは、n^2 と指定することで、プログラムを実行する PU 台数を指定することができる。また、PU 台数と同様に、Make 時に問題のクラスをW、A、B、C と指定することで、計算の対象となる問題サイズを指定することができる。

4.2 予測対象計算機

予測対象となる計算機構成を表 2 に示す。

表2 :予測対象計算機構成

Node	128
CPU	Intel Pentium4
Clock	2.4GHz
L2 Cache	512KByte
Memory	1Gbyte SDRAM
Network	100 Base TX Ethernet
Network Seitch	NETGEAR Gigabit Switch
Compiler	gcc 2.96
Compiler Option	-O3
MPI Library	MPICH ver 1.2.5

また、予測対象計算機のネットワーク構成は、ノード16 台を1 つのスイッチに接続し、ノード16 台をまとめた8 つのスイッチを、上流で1 つのスイッチにカスケードしている。

4.3 NPB ver2.4 の実行時間の予測

MPIETE を用いて、NPB ver2.4 の、EP、CG、LU の実行時間を予測し、予測実行時間と実際の実行時間を比較する。なお、予測する対象とする区間は、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する。また、NPB ver2.4 では、計算時間、通信時間を分けて測定することができないため、通信を行っている文の前後に、時間を測定するソースコードを挿入することにより通信時間を取得し、全体の実行時間から、通信時間を引いた時間を、計算時間とする。

4.3.1 予測実行時間と実際の実行時間

図 6 に、CLASS B の EP、CG、LU の、1 から128PU 上での予測実行時間と、実際の実行時間を示す。

4.3.2 予測に必要な時間と実際の実行時間

CLASS B の EP、CG、LU の、予測に必要な時間と、実際の実行時間の比較を、図 7 に示す。また、表 3 に、実行時間の算出を行った計算機構成を示す。なお、プログラム基礎データは、予測対象計算で実行する必要がないため、プログラム基礎データ取得プログラムの実行時間は、予測に必要な時間には含めないものとする。

4.4 考察

4.3 で示した NPB ver2.4 の実行時間の予測結果と、予測に必要な時間をもとに、予測精度と、MPIETE を用いた予測の効率性について、考察を行う。

4.4.1 予測精度

4.3 で示した予測結果をもとに、MPIETE が予測する EP、CG、LU の予測実行時間の予測誤差を、表 4 に示す。なお、予測誤差は、式 8 より求めた。

$$\text{予測誤差}(\%) = \frac{|\text{予測実行時間} - \text{実際の実行時間}|}{\text{実際の実行時間}} \times 100 \quad \cdots(\text{式}8)$$

表 4 より、計算時間の予測誤差が全て 12%以内であるのに対し、ほとんどの通信時間の予測誤差が 30%以上であることがわかる。

上記で示した、通信時間の予測誤差が大きいことの原因として、通信のコンテンションがあげられる。MPIETE では、3.1.2 で示したように、MPI_RECV 関数を利用して、メッセージの受信時間を測定し、測定した通信時間を通信基礎データとしている。しかし、実際の通信では、1 つのランクが送信と受信を同時に行う通信が存在する。送信を行いながら、受信を行う通信は、通信のコンテンション が発生する可能性がある。また、4.2 で示したように、予測対象計算機は、ノード16 台を1 つのスイッチ に接続し、ノード16 台をまとめた 8 つのスイッチを、上流で 1 つのスイッチにカスケードしているため、異なるスイッチ に接続しているノード間で、同時に 2 つの送受信が行われる場合、通信のコンテンション が発生する可能性がある。

4.4.2 予測の効率性

4.3.2 に示したように、32PU 以上の予測に必要な時間は、実際の実行時間より大きくなるものの、同じプログラムを 4PU で実行する時間よりは、短時間で予測することが可能である。また、128PU の予測に必要な PU 台数は 1PU なのに対し、実際に実行した場合は、128PU 必要になることを考慮に入れると、MPIETE を用いた MPI プログラム予測手法は、効果的に MPI プログラムの実行時間を予測できるといえる。

4.5 まとめ

以下に、検証のまとめを示す。

- 計算時間は、予測誤差 12%以内で予測が可能である
- 通信のコンテンション が発生する可能性がある通信の通信時間は、予測誤差が 30%以上となる
- 予測する PU 台数が増えると、実際の実行時間 と比較し予測に必要な時間は大きくなるが、1PU で予測が可能であるため、効果的に MPI プログラムの実行時間を予測できる

また、今後の課題を、以下にあげる。

- 通信のコンテンション が発生する可能性のある通信の通信時間を予測する
- 予測する PU 台数が増えた場合でも、短時間で MPI プログラムの実行時間を予測する

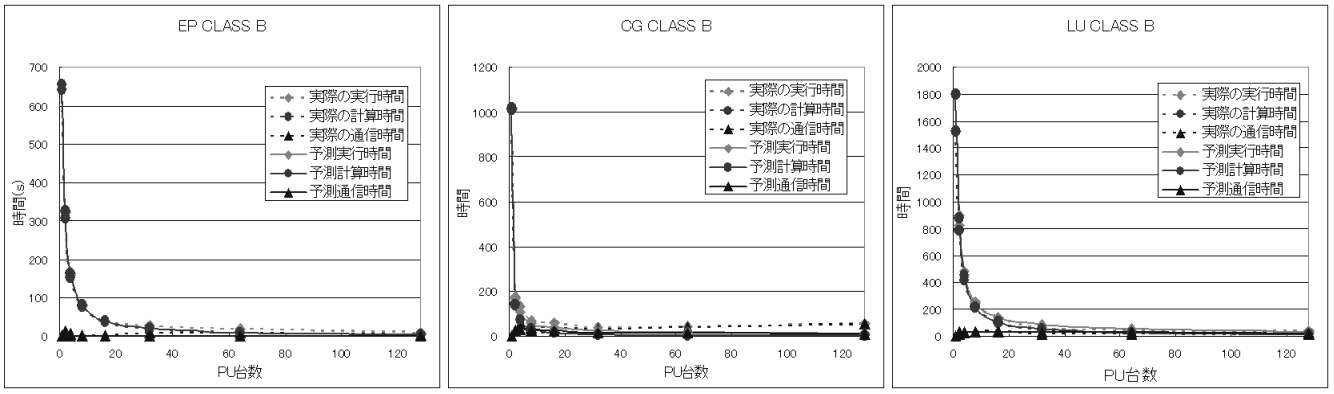

図 6 :予測実行時間と実際の実行時間

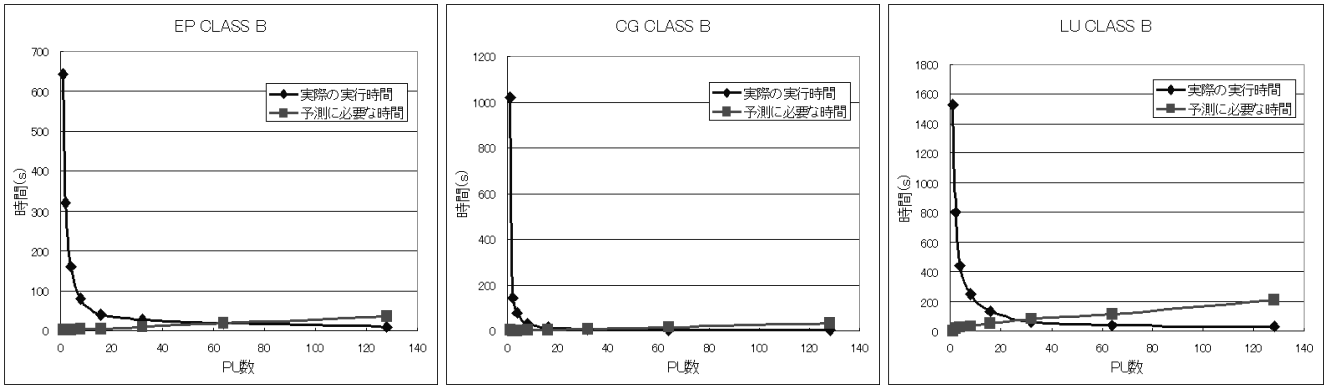

図 7 :実際の実行時間と予測に必要な時間

表3 :実行時間の算出を行った計算機構成

CPU	Intel Pentium4
Clock	3.0GHz
L2 Cache	512KByte
Memory	1Gbyte RDRAM
Compiler	Visual C++ .NET 2003
Compiler Option	/O2 /Ot /G7

表4 :EP、CG、LU CLASS B の計算時間と通信時間の予測誤差 (%)

	EP						
PU	2	4	8	16	32	64	128
計算時間	6.40	6.78	6.03	6.83	2.45	2.92	2.56
通信時間	100.00	100.00	99.99	99.97	99.99	99.99	99.98
	CG						
PU	2	4	8	16	32	64	128
計算時間	2.37	3.80	11.69	1.98	9.66	10.29	4.28
通信時間	17.23	38.34	36.90	44.82	52.42	59.91	81.13
	LU						
PU	2	4	8	16	32	64	128
計算時間	11.53	9.68	4.43	6.15	2.99	8.84	9.36
通信時間	44.80	16.89	10.27	13.88	85.10	66.69	28.64

5. おわりに

本稿では、同じプログラムを予測する際、一度生成した予測に必要なデータを、次の予測にも利用することが可能な MPI プログラム実行時間予測手法を提案し、提案する手法を用いて MPI プログラムの実行時間を予測するツール MPIETE を示し、検証した。

MPIETE の予測誤差の主要因は通信誤差時間によるものである。このため、今後は通信時間の予測精度を上げるため、ネットワークの予測は、他のネットワークシミュレータの利用を検討していく。

参考文献

[1] Asim YarKhan, Jack J. Dongarra : “Experiments with Scheduling Using Simulated Annealing in Grid Environment” Workshop on Grid Computing, June 7, 2002

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

[3] 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..

[4] Dickens P., Heidelberger P., and D. Nicol: “Parallelized Direct Execution Simulation ofMessage-Passing Parallel Programs”, IEEETrans. on Parallel and Dist. Systems, Vol.7,No.10, pp.1090-1105, Oct. 1996.

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

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

[7] Thomas Fahringer, Hans P. Zima: “A StaticParameter based Performance Prediction Toolfor Parallel Programs”, Proc. of 1993 ACMInt. Conf. on Supercomputing, pp.207-219,July, 1993.

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