ハイパフォーマンス
コンピューティング 88−11
（２００１．１０．２６）

実測に基づいたMPIプログラムの実行時間予測手法

堀井　洋　　山名早人
早稲田大学理工学部

概要

本稿ではMPIプログラムの実行時間を低コストで予測する手法を提案し, 本手法の有用性をNAS Parallel Benchmark（NPB）ver 2.3を用いて検証した. 提案する予測手法ではMPIプログラムの実行時間を計算部分と通信部分に分けて予測する. 計算部分の予測では, MPIプログラムをループ構造をもつ複数ブロックに分け, ブロックごとの計算部分の実行時間を測定し, その基礎データをもとに想定するPU台数時の実行時間を予測する. 通信部分の予測では, MPIプログラム中で行われる通信と同じサイズの通信にかかる時間をあらかじめ想定するプラットフォーム上で測定し, その基礎データをもとに通信時間を予測する.

An Estimation Scheme of the Exection Time for MPI Programs using Measured Primitives

Hiroshi Horii　　Hayato Yamana
School of Science and Engineering, Waseda Univ

Abstract

In this paper, we propose the scheme of estimating the execution time of MPI programs, and confirmed the usefulness of the scheme using NAS Parallel Benchmarks (NPB) ver 2.3.
The scheme estimates the execution time of MPI program dividing into the computation part and the communication part. In estimating the execution time of the computation, we divide a MPI program into blocks that have loop structure, measure the execution time of every block, and estimate the total execution time. In estimating the communication time, we measure the communication time with the same message size which is sent in original MPI program on the same platform. Then, we estimate the communication time on the assumed number of PU using the measured communication time.

1　はじめに

本稿では, データ並列プログラムをモデル化し, システムのPU（Processing Unit）台数が増えた際の実行時間の予測する手法について論じる.

MPIプログラム [1] のような並列プログラムでは, 一般的に, PUが実際に計算する時間以外に, 通信に必要とする時間がプログラム全体の実行時間に大きく影響を及ぼす. そのため, PU台数を増やしてプログラムを実行した場合でも, 増やしたPU台数での台数効果が表れない場合がある. プログラムを実行するプラットフォームを拡張する場合, プログラムがどのPU台数時にピーク性能を得られるのかを事前に知ることは重要である. しかし, 現状のシミュレーションでは膨大な処理時間を要する.

本稿で提案する予測手法は, MPIプログラムを計算部分と通信部分に分けて予測する. 計算部分の予測では, プログラムをループ構造をもつブロックに分割し, 一台のPUで実行することにより得られる基礎データをもとに, 対象とするPU台数時の計算時間を予測する. 通信部分の予測では, プログラム中で行われる通信と同じサイズの通信にかかる時間をあらかじめ想定するプラットフォームで測定し, その基礎データをもとにプログラム中の通信時間を予測する. 本手法を用いることで, 大規模なPU台数時の実行時間を従来の手法に比較して低コストで予測することが可能である.

本手法を用い, NAS Parallel Benchmark（NPB）[2] ver2.3のCG, EP, のCLASS W, B, をPentium4 Cluster上で予測し, 実際の実行時間との比較を行う.

本稿は以下の構成をとる. 第2節はプログラムのモデル化について述べる. 第3節では提案する予測手法について, 第4節では実際の実行時間と予測時間との比較を行う. 第5節では今後の課題を述べ, 第6節では関連研究について, そして最後にまとめを述べるものとする.

2 プログラムのモデル化

本手法では MPI プログラムをループ構造を持つ複数のブロックに分割し, 各ブロックの (1) ブロックの総実行時間, (2) ブロック内のループ回数の総和, (3) ブロックが呼ばれた回数, の基礎データを得る. MPI のソースプログラム中にこれらの基礎データを取得するためのコードを挿入し, 1〜2 台の PU で実行することで取得する.

2.1 ブロックへの分割

以下の方法に基づいてプログラムをループ構造を持つブロックへ分割する.

- サブルーチンはインライン展開する.
- ループ構造を持たない部分は, 繰り返し回数が 1 回のループと考える.
- 最内ループを 1 つのブロックとする.
- ループ内にループしか存在しない多重ループの場合は, 最上層のループを 1 つのブロックとする.
- ループ内で通信部分が宣言されているループは通信部分の前後で 2 つのブロックに分割する.

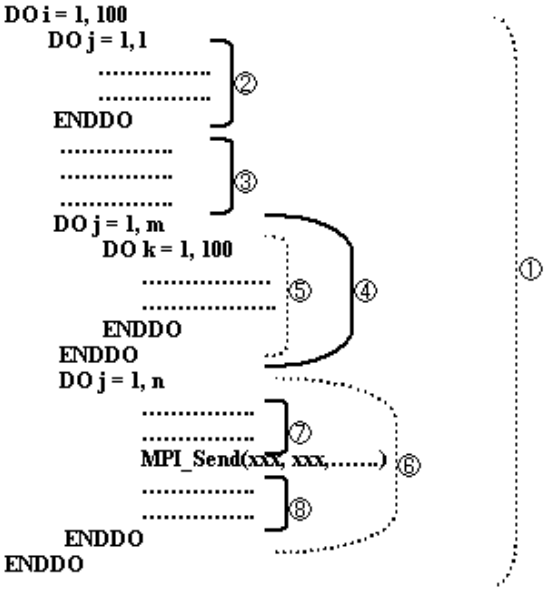

図 1: プログラムの分割例（実線部をブロックとする）

例として図 1 のプログラムをブロックへ分割する. 1 のループは, 最内ループでないためブロックとしない. 2 のループは最内ループのためブロックとする. 3 の部分はループ構造がないが, ループ回数が 1 回のループと見なし, ブロックとする. 4 のループは最内ループではないが, ループ内にループしか存在しないためブロックとする. 5 のループは内部に MPI 命令があるためブロックとしない. 7 と 8 は 3 と同様にループ回数が 1 回のブロックとする.

2.2 ブロックの基礎データの取得

前節の方法でプログラムを N 個のブロックに分割した後, 以下のブロック基礎データを得るためのソースコードを挿入した MPI プログラムを 1PU 上で実行する.

- ブロック i の実行時間（*T_comp(1, i)*）
- ブロック i の呼び出された回数 (*T_count(1, i)*)
- ブロック i 実行時に最下層ループのループボディが繰り返された回数 (*L_count(1, i)*)
  (ただし, $1 \leq i \leq N$ であり, (1, i) は, 1 台の PU でブロック i を実行したことを示す.)

なお *L_count(1, i)* は DO ループ文の引数から静的に解析できる場合は DO ループ文の引数から計算によって求める. 一方, 実行時に DO ループ文の引数が決定される場合では, 静的に解析することができないため, ループ内にループ回数をカウントするための関数を挿入し計測する.

3 提案する予測方法

PU 台数が p 台の時のブロック i の最下層ループボディが繰り返された回数 *L_count(p, i)* を求め, 2 節で求めたブロック i での実行時間を用いて, PU が p 台の時のブロック実行時間 *T_comp(p, i)* を予測する. また, プログラム中で通信されている同サイズの通信を 2 台の PU で行い, 通信にかかる時間を計測し通信の基礎データとし, その基礎データにより通信部分の予測を行う.

今回の予測対象は通信の内容によって処理のフローが変わらないデータ並列性を持つ MPI プログラムである. ここで扱っている並列プログラムは負荷がほぼ均等に分散し, それぞれのプロセスにかかる実行時間はほぼ同じであるものと仮定している.

3.1 計算部分の予測

PU 台数が p 台の時のブロック実行時の最下層ループのループボディが繰り返された回数 *L_count(p, i)* を求めることで, PU が p 台の時のブロック実行時間 *T_comp(p, i)* を予測する.

まず対象となる MPI プログラムから, 次のようなプログラムを作成し, *L_count(p, i)* を計測する.

- MPI プログラム中のループ回数に依存する変数の計算のみを残し, 他の実行文はコメントアウトする.
- MPI_Comm_Size 命令を実行した際に得られる同時実行総プロセス数を p に固定する.
- MPI_Comm_rank 命令を実行した際に得られるプロセスランクを 0 に固定する.

以上の手順によって得られたプログラムを 1PU で実行し, プロセス 0 における *L_count*（*p, i*）を測定する. この *L_count(p, i)* と, 前節で得られたブロックの基礎情報をもとに, PU 台数が p 台の時のブロックの実行時間 *T_comp(p, i)* を（1）式によって計算する.

$$T\_comp(p,i) = T\_comp(1,i) \times \frac{L\_count(p,i)}{L\_count(1,i)} \quad (1)$$

そして得られたブロック 1 からブロック n の *T_comp(p, i)* より, PU 台数が p の時の全ブロック実行時間

Tall_comp(p) を, 式（2）により予測する.

$$Tall\_comp(p) = \sum_{i=1}^{N} T\_comp(p,i) \qquad (2)$$

3.2 通信部分の予測

MPI のプログラムの通信方法は, 大きく分けてブロック通信, ノンブロッキング通信, 集団通信の 3 種類の通信方法に分けられる. 本手法ではそれぞれの通信方式に合わせて, 通信時間を 3 種類の手法で予測する. なお今回の予測対象とする通信は, すべてメッセージの通信データサイズとタイプがプログラム中で示されているものとし, また, 送信と受信を同時に行わないものとする.

3.2.1 ブロック通信予測

MPI におけるブロック通信は, send 側では送信メッセージが送信バッファに保存されるまで, receive 側ではメッセージを受信するまで, 次の処理に進むことができない通信方式である. そしてメッセージを送受信する際には, メッセージの通信データサイズを指定する必要がある.

本手法では MPI プログラム中で呼ばれているブロック通信に必要とする総時間 *Tall_bcomm(p)* を以下の方法で予測する.

まず, PU 台数が p 台の時の MPI プログラムを実行する際の, ブロック通信している MPI の送受信命令を調べ, データサイズの種類 *Mb*, i 種類目のデータサイズ *size(i)* ($1 \leq i \leq Mb$), そして *size(i)* のメッセージがプログラム中で通信されている回数 *bcount(i)* を求める. 次に, 2PU 間で実際に *size(i)* のメッセージをブロック通信させ, その際にかかる時間 *T_send(size(i))* を求め,

$$T\_bcomm(p,i) = T\_send(size(i)) \times bcount(i) \qquad (3)$$

により, MPI プログラムを実行した際の, *size(i)* のメッセージをブロック通信している時間を予測する. そして,

$$Tall\_bcomm(p) = \sum_{i=1}^{Mb} T\_bcomm(p,i) \qquad (4)$$

よりプログラム中のブロック通信に必要とする時間 *Tall_bcomm(p)* を予測する. なお, 通信先プロセスがプロセスランクと同じ, つまり自分自身にメッセージを送っている場合は, 通信バッファにメッセージを書き込んだ時点で通信が終了するため, バッファにメッセージを書き込むのに必要な時間を対象となる PU で測定し, 得た時間を通信時間とする.

3.2.2 集団通信の予測

MPI では複数のプロセスが同時に通信を行う集団通信を行うことができる. MPI の集団通信命令は, 命令ごとに違った通信形態を持っている. よってプログラム中の集団通信をしている時間を予測するには, 集団通信命令ごとの通信形態に合わせて時間を予測する必要がある.

集団通信命令の予測時間を求める例として, MPI_Bcast 命令を説明する.

プログラム中に *Mg* 個の集団通信が呼ばれているとする. その i 番目の集団通信命令が MPI_Bcast 命令であったとする. MPI_Bcast 命令は, 1 つのプロセスから全プロセスにメッセージを送信する集団通信命令で, 木構造通信という通信形態をとっている. MPI_Bcast 命令で通信するメッセージサイズを *size(i)* とし, *size(i)* のメッセージサイズを想定する 2PU 間でブロック通信するのに必要な時間を *T_send(size(i)* とすると, PU 台数が p 台で MPI_Bcast 命令を実行するのに必要な時間 *T_gcomm(p, i)* は,

$$T\_gcomm(p,i) = T\_send(size(i)) \times n \qquad (5)$$

$$(\text{ただし } 2^{n-1} < p \leq 2^n)$$

と予測できる. MPI_Bcast 以外の MPI の集団通信命令も, 通信されるデータサイズを 2PU 間でブロック通信する際に必要な時間と, 集団通信命令の通信形態を知ることで, 集団通信命令に必要な時間を予測することができる.

プログラム中で命令されているそれぞれの集団通信にかかる時間を求めることで, プログラム中のすべての集団通信に必要とする時間 *Tall_gcomm(p)* は,

$$Tall\_gcomm(p) = \sum_{i=1}^{Mg} T\_gcomm(p,i) \qquad (6)$$

と予測することができる.

3.2.3 ノンブロッキング通信予測

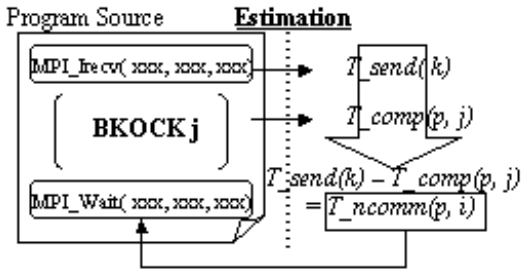

図 2: ノンブロッキング通信の予測方法

MPI ではノンブロッキング通信の関数を呼ぶことで, 通信を行いながら計算を並列で行うことができる. ノンブロッキング通信関数で呼び出された通信は, MPI_Wait 命令が呼ばれるまでに通信を終了させる必要があり, 通信が終わっていない場合は次の命令に進むことができない. ノンブロッキング通信の予測は, 以下の方法を用いて行う.

図 2 のような MPI プログラムの部分があったとする（想定する PU 台数を p 台とする）. 図 2 ではまず MPI_Irecv 命令が呼ばれ, 通信と並行してブロック j を計算し, その後 MPI_Wait 命令で通信の終了を待っている. 図 2 のようなノンブロッキング通信にかかる時間を予測するためには, 通信の終了を待つ時間, つまり MPI_Wait 命令を実行するのに必要とする時間 *T_ncomm(p, i)* を予測すればよい.

まず, ブロック通信と同様に, ノンブロック通信している MPI の送受信命令を調べ, データサイズの種類 Mn, i 種類目のデータサイズ *size(i)* ($1 \leq i \leq Mn$) を求める. そして, 想定する 2PU 間で *size(i)* のメッセージのブロック通信をするのに必要な時間 *T_send(size(i))* を実際に 2PU 間で通信させることにより求める. また PU 台数が p 台のときのブロック j の計算に必要な時間 *T_comp(p, j)* を前節までに示したブロック実行時間予測とブロック通信時間予測, 集団通信時間予測により求める. すると $T\_ncomm(p, i)$ は,

$$T\_ncomm(p,i) = max(0, (T\_send(i) - T\_comp(p,j))) \tag{7}$$

と予測することができる. そしてプログラム全体でのノンブロッキング通信に必要とする時間 *Tall_ncomm(p)* は,

$$Tall\_ncomm(p) = \sum_{i=1}^{Mn} T\_ncomm(p,i) \tag{8}$$

より予測することができる.

3.3 プログラムの全実行時間の予測

PU 台数が p 台の時のプログラム全体の実行時間 *T_All(p)* は, 以上の方法で得られたブロック部分の計算予測時間 *Tall_comp(p)* と通信部分の予測時間 *Tall_bcomm(p)*, *Tall_ncomm(p)*, *Tall_gcomm(p)* より,

$$\begin{aligned} T\_All(p) &= Tall\_comp(p) + Tall\_bcomm(p) \\ &+ Tall\_ncomm(p) + Tall\_gcomm(p) \end{aligned} \tag{9}$$

と予測される.

4 検証

今回, 本手法の検証に利用した NPB ver 2.3 は, MPI を利用した FORTRAN および C でかかれたプログラムである. 本節では 3 節で提案した手法を用いて, NPB ver 2.3 の 8 つのプログラムの内の, EP・CG の Pentium4Cluster 上での, 2〜8PU 台数時の実行時間予測を行った. なおここでは, NPB ver 2.3 が測定の対象としている区間での実行時間を対象に検証を行った. 予測のためのブロックの基礎データの取得には, CLASS W, 2PU で測定した値を使用した. オリジナルのソースと予測時間の比較は, 計算時間と通信時間に分けて比較する. オリジナルソースのままでは実行時間を計算時間と通信時間を分けて得ることができないので, 通信に必要な時間を測定するためソースコードを挿入し, 通信時間と計算時間を得ることにした.

4.1 対象システムの構成

今回ブロックの基礎データの測定, 通信の基礎データの測定, また検証の対象に利用した Pentium4Cluster の構成を表 1 に示す.

表 1: 対象測定システム

CPU	Pentium4
clock	1.4GHz
L2cache	256KByte
Memory	512MB RDRAM
Network Controller	3ComR3C920
Network	100BaseTX Ethernet
Switch	P/N: FX-16NP
OS	RedHat 7.01
Compiler	PGI 3.2.3
Compiler Option	-fast

4.2 EP

EP は典型的なモンテカルロシミュレーション問題である. コードはほぼ完全な並列化がなされており, 途中で通信は行われず, 最後にいくつかの配列と実数のリダクションを行っている.

EP の CLASS W, B の実際の計算・通信時間と, 予測した計算・通信時間を比較し, 表 2, 3 に示す. 計算時間・通信部分ともに, 2 つのクラスで高精度の予測ができており, それぞれを足した全体の予測実行時間と実際の時間との誤差は, すべてのクラス, PU 台数ともに 5 %以内に収まっている.

4.3 CG

CG は, Conjugate Gradient 法を用いて大規模疎行列の最小固有値を求める問題である. CG は PU 台数が増えるにつれ, 1 回の通信のメッセージサイズは減少するが通信回数が増えていく. すべてのプロセスは, 同じ通信を行っておらず, 他のプロセスが自身以外のプロセスにメッセージを送っている中で, あるプロセスは自分自身にメッセージを送っている部分がプログラム中に存在する. 最も予測通信時間が長いプロセスを全体の予測通信時間とし, 本来の通信時間と比較した.

CG の CLASS W, B の本来の計算・通信時間と予測した計算・通信時間の比較し, 表 4, 5 に示す.

計算部分の予測に関しては, CLASS W では誤差が 10%以内だが, CLASS B においては 2, 4PU 時で本来の計算時間と予測時間の差が大きい. これは CLASS B の PU 台数が少ない場合ではキャッシュヒット率が悪いことが原因と考えられる. ブロック実行時間の計測をキャッシュヒット率が高い CLASS W の 2PU 時をもとに行っているため, キャッシュヒット率が極端に低くなる CLASS B の少ない PU 台数の場合, 予測計算時間は本来の計算時間より短くなってしまうと考えられる.

通信部分の予測に関してはすべての CLASS, PU で誤差が 5%以内に収まっており, 高い精度で予測ができている.

5 今後の課題

5.1 高速化

提案する予測手法はデータ並列プログラムをループ構造を持つ複数のブロックに分け, 1 つのプロセスのブロックごとの計算時間, ブロック内のループ回数, ブロック

表 2: EP CLASS W の本来の計算時間・通信時間と予測した計算時間・通信時間との比較

	Computation		Communication	
#PU	Real	Estimate	Real	Estimate
2	4.164	3.942	0.000732	0.00556
4	1.967	1.971	0.0014	0.00120
8	0.986	0.986	0.00219	0.0178

単位 (s)

表 3: EP CLASS B の本来の計算時間・通信時間と予測した計算時間・通信時間との比較

	Computation		Communication	
#PU	Real	Estimate	Real	Estimate
2	126.573	125.541	0.00074	0.00550
4	63.295	62.769	0.00149	0.00126
8	31.429	31.384	0.00220	0.0138

単位 (s)

表 4: CG CLASS W の本来の計算時間・通信時間と予測した計算時間・通信時間との比較

	Computation		Communication	
#PU	Real	Estimate	Real	Estimate
2	0.941	0.957	1.464	1.557
4	0.480	0.505	3.333	3.319
8	0.323	0.295	2.909	2.844

単位 (s)

表 5: CG CLASS B の本来の計算時間・通信時間と予測した計算時間・通信時間との比較

	Computation		Communication	
#PU	Real	Estimate	Real	Estimate
2	391.003	137.427	99.607	97.936
4	189.910	70.333	197.270	197.682
8	41.526	34.996	149.158	149.290

単位 (s)

の呼ばれた回数といったブロックの基礎データを測定し, 想定する PU 台数時のブロック内のループ回数を求めることで, 想定する PU 台数時の計算時間を予測している. 本手法を利用するとブロックの基礎データを得られれば, 短時間で PU 台数が増えた場合の予測時間を算出することができる. しかし, 本手法ではブロック部分の時間情報を得るために一度全プログラムを対象となるシステム上で実行する必要があり, NPB のように CLASS が分かれていて問題サイズが小さい CLASS W のようなプログラムで測定する場合は問題ないが, 大規模な問題サイズしか持たないプログラムを少ない PU 台数で測定するのは困難である. この問題はループ回数を少ない値にして測定することで解決できる. なお 4 節での検証において, CG ではブロック時間情報を得た後の予測にかかる時間はすべての PU, すべての CLASS でも 5 秒以内である.

5.2 高精度化

本手法は以下の問題点を解決することで, さらに高精度な予測をすることが可能になると思われる.

- キャッシュ効果
- バッファの通信への影響
- 実行時のネットワーク資源の競合

キャッシュ効果の影響はブロック情報をアセンブラレベルで解析する必要がある. 対象となる問題サイズ時のブロックのアセンブラ情報より load/store 命令を解析し, 想定するシステム時のキャッシュヒット率を予測し, 予測したキャッシュヒット率とブロック基礎データより対象とするブロックの計算時間の予測が可能である. また, 今回キャッシュヒット率が低下したのは, CLASS が大きくなるとループ回数が増加し, ループ内で呼ばれる配列データをキャッシュすることができなかったことによるものと考えられる. よってループ回数からキャッシュヒット率を予測することも重要となる.

今回提案した予測手法では, 通信のための送受信バッファが常に利用可能であることを想定している. しかし実際の通信においては, 送受信バッファにまだ送受信が終了していない通信メッセージが存在していて, 前の通信が終了しなくては送受信バッファに次の通信メッセージをコピーできない場合が考えられる. 大規模なメッセージを送受信するプログラムの予測には, 送受信バッファの状況をモデル化する必要がある.

複数の PU が同時に同じ PU にメッセージを送信する場合, ネットワーク資源の競合が起こる. 本手法では 1PU の送受信状況から通信時間を予測しているため, 送受信先の PU の通信状況を知ることはできない. ネットワーク資源の競合を正確に予測するには, 全 PU の通信状況をシミュレーションする必要がある.

5.3 他プラットフォーム上での予測

本手法では 1 つのプラットフォーム上で測定したブロック基礎データより, 他システムでの実行時間を予測することが可能と考えられる.

5.3.1 各種ネットワークへの対応

ネットワークを変更した場合の予測には, 変更後のネットワーク上で, プログラム中で行われる通信と同じデータサイズを通信するのに必要な時間を得ることができれば, プログラム全体の実行時間を予測することが可能である. また通信時間情報を得るのに実測ではなくシミュレーションから得る方法も考えられ, ネットワークをパラメータ化することで通信時間情報を得られるように拡張をする予定である.

5.3.2 各種システムへの対応

プラットフォームをクロック, キャッシュ量などをパラメータ化することで, あるプラットフォーム上で得られたブロック基礎データから, 他プラットフォームでの

ブロック基礎データを予測することも今後の課題である.

6 関連研究

Fahringer らの提案するシステム（PPPT[3]）は, 逐次プログラムをメッセージパッシングプログラムに変換する際, 逐次プログラムの分岐の割合, ループの回数, ステートメントの実行回数をプロファイリングし, ネットワークの衝突率, キャッシュのヒット率を大ざっぱに予測することで, 変換したメッセージパッシングプログラムの実行時間を予測している. またこのほかにもアルゴリズムやプログラムを解析することで, 大規模な並列プログラムの挙動を解析する方法 [4, 5] が多く見られる. しかし, これらの手法はコンパイラの最適化, プログラムのインプリメントの考慮することができないため, プログラムの全般的な傾向を見ることはできるが, 実行時間を正確に導き出すことは難しい. 本稿の提案手法では, 計算部分の予測は, キャッシュヒット率がほぼ同一の場合ならば正確に予測することが可能である.

ネットワークをシミュレーションツール INSPIRE[6] を用いて予測し, また計算実行時間はアセンブラコードを解析し, キャッシュヒット率を求めることで大規模な PU 台数時のプログラム挙動を予測した例がある [7]. またネットワークをシミュレーション, 計算部分はトレースファイルより予測する例もある [8]. これらはネットワークを完全にシミュレーションし, ネットワークの衝突率や, 各種パラメータごとの通信時間の変化を得ることができ, 精度の高い予測が可能である. しかし, ネットワークシミュレーションの実行や, トレースファイルの作成は, 大規模な問題サイズでは長い処理時間を必要とする可能性がある. 本稿の手法では, 小規模な問題サイズから大規模な問題サイズの予測が可能で, 高速に予測を行うことができる.

7 おわりに

本稿ではプログラムを計算部分と, 通信部分に分割し, 実測データをもとに PU 台数が増えた時の MPI プログラムの実行時間を予測する手法を報告した.

ブロックはループ構造を持っており, ループ回数とブロック全体の実行時間をもとに, 想定する PU 台数時のループ回数を予測することでブロックの実行時間の変化を予測する. 通信部分においては, MPI の通信関数で宣言される通信メッセージサイズから, 想定する PU 上で同サイズのメッセージの通信にかかる時間を測定し, その通信時間をもとにプログラム全体の通信時間を求める.

本予測手法を NAS Parallel Benchmark ver2.3 の EP, CG を用いて検証した結果, 計算部分に関してはキャッシュヒット率が同じ場合は高い精度で予測することが可能であることがわかった. そしてブロックの基礎データ得られれば, 高速に予測することが可能であることがわかった.

今後, 通信部分の高性能化, キャッシュヒット率の予測と実行時間への影響, またさらなる低コスト化を課題に研究を行っていく.

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

[3] Thomas Fahringer, Hans P. Zima: ”A Static Parameter based Performance Prediction Tool for Parallel Programs”, Proc. of 1993 ACM Int. Conf. on Supercomputing, pp.207-219, July, 1993.

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

[5] 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.

[6] 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.

[7] 久保田和人, 板倉憲一, 佐藤三久, 朴泰祐: ”大規模データ並列プログラムの性能予測手法と NPB2.3 の性能評価” 情報処理学会論文誌 Vol.40, No. 5, pp. 2293-2304, 1999.

[8] 三島正博, 板倉憲一, 朴泰祐, 中村宏, 中澤喜三郎: ”並列計算機の仮想性能評価システム　VIPPES”, 情処研報 96-HPC-62-5, 1996.