2003－HPC－95　(23)
2003／8／5

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

岩渕寿寛[†]　堀井洋[†]　山名早人[‡]

概要　本稿では、我々がこれまでに提案している MPI プログラムの実行時間予測手法を NAS Parallel Benchmarks ver.2.3 に適用し、2〜128PU の PC クラスタ上で評価した結果を示す。実行時間予測は、対象とするアプリケーションの実行に最適な PU 台数を特定するための有効な手法である。従来提案されている予測手法は、実行時間を予測するだけでなく、各種オーバヘッドの詳細な情報を得ることを目的にしている場合が多く、長時間のシミュレーションが必要であった。我々の手法では、実行時間を得ることに目的を限定し、実行対象機上で簡易実行を行うことにより、実際の実行より速く予測することが可能となる。2〜128PU での評価の結果、実際の実行時間の $O(10^{-1})$〜$O(10^{-3})$の時間で予測が可能であり、CG、IS を除くプログラム 6 つの 32PU までの実行で、予測誤差が 10%以内に収まることを確認した。

Evaluation of Execution-time Prediction Method of MPI Programs based Simple Execution

Toshihiro IWABUCHI[†]　Hiroshi HORII[†]　Hayato YAMANA[‡]

Abstract　In this paper, we show evaluation results of our execution-time prediction method which simply executes MPI program on 2PU. We predict the execution time of NAS Parallel Benchmarks ver.2.3 on 2-128PU. Execution time prediction is effective technique to determine the optimal number of PU for some target applications. The most existing methods are not only for predicting execution-time but for obtaining information of various overhead, and hence need the long simulation time. On the other hand, since our purpose is to obtain execution time only, our method can predict faster than actual execution of some target applications. As a result of evaluation using 2-128PU, our method is $O(10^{-1})$-$O(10^{-3})$ times faster than actual execution. The error rates of our method are within 10% on 2-32PU, except for CG and IS.

1. はじめに

本稿では、MPI[1]プログラムを複数の PU を用いて実行したときの実行時間を、短時間で精度よく予測する手法について論じる。MPI プログラムのような並列プログラムは、一般に同一プログラムを様々な PU 台数で実行することが可能である。しかしながら PU 台数を増やして実行しても、プログラム中の通信時間の増加やプログラムの逐次性部分の影響で、期待する台数効果がそのまま実行時間に反映しない場合がある。このような背景のもと、想定するプラットフォームでの実行に先立って、MPI プログラムの実行時間を予測することが重要になってきている。並列プログラムがどの程度の実行時間になるのか、またどの PU 台数でピーク性能を得られるかを実行前に知ることは、コストパフォーマンスの向上、プログラムの特性を知る目安となる。

従来、MPI プログラムの実行時間の予測手法として、EXCIT&INSPIRE[2]や、LAPSE[3]、MPI-SIM[4]といったシミュレータを用いた手法が提案されている。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。

この問題に対して、我々はこれまでに、プログラムの簡易実行による短時間での実行時間予測手法を提案した[8]。本稿では、[8]の手法の有効性を検証するため、128PU のクラスタにおいて Nas Parallel Benchmarks ver.2.3(NPB2.3)の実行時間予測を行い、予測結果とともに考察を述べる。

以下、2 節で従来手法、3 節で簡易実行によ

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

る実行時間予測手法の概要、4 節で手法の有効性の評価結果を示し、5 節で結果に対する考察を述べ、6 節で今後の課題とまとめを行う。

2. 従来手法

従来、MPI プログラムの実行時間の予測手法として、EXCIT&INSPIRE[2]や、LAPSE[3]、MPI-SIM[4]等のシミュレータを用いた手法が提案されている。

[2]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[3]、MPI-SIM[4]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[2][3][4]の手法では、実際の実行時間に対し、シミュレーションに数倍以上の時間を必要とする。

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[5][6][7]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、[2][3][4]の手法と比べ、予測精度が悪い。

3. 簡易実行による予測手法[8]

2 節で述べた従来手法の問題点に対して、我々は[8]において、[2][3][4]の手法と比較し短時間で、かつ[5]の手法と比較し精度の高い予測手法を提案した。本手法では、対象となる MPI プログラムを、2PU 上で簡易実行することで得られる基礎パラメータを用いてプログラム全体の実行時間を予測する。

3.1 プログラムのモデル化

本手法では、プログラムを最内ループのループボディを構成する「計算ブロック」と MPI 通信関数を含む文である「通信ブロック」とに分割し、ブロック単位で実行時間予測を行う。以下にプログラムを計算ブロックと通信ブロックに分割する方法を述べる。

(1) ループ構造を持たない部分は、繰り返し回数が 1 回のループとする。
(2) 最内ループのループボディを 1 つの計算ブロックとする。
(3) (2)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 関数を含む文の前後で 2 つの計算ブロックに分割する。また、MPI 通信関数を含む文を通信ブロックとする。

図 1 に示す例では、(1)により①が、(2)により②③④が各々1 つの計算ブロックとなる。(3)により⑥が通信ブロックとなり、⑤⑦は各々別の計算ブロックとなる。

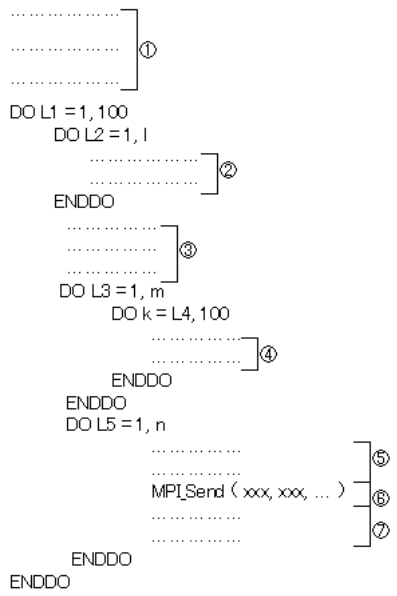


図 1：プログラムのブロックへの分割例

3.2 基礎パラメータの取得

予測に必要な、各ブロックの基礎パラメータ及び各ブロックの実行回数を取得する方法を述べる。

計算ブロックの基礎パラメータは、各計算ブロックが 1 回実行されるときの実行時間である。ただし、コンパイラ最適化やキャッシュ効果を考慮した計算ブロックの実行時間を得るため、計算ブロック 1 回分の実行時間は、当該計算ブロックのみをループボディとしてもつループ全体の実行時間を計測した後に求める。

通信ブロックの基礎パラメータは、各通信における通信サイズであり、MPI 通信関数における引数である。具体的には、メッセージサイズ、メッセージタイプ、タグが基礎パラメータとなる。これらパラメータを取得した後、2PU 間での擬似通信を行い、各通信ブロック 1 回分の実行時間を得る。

3.2.1 基礎パラメータ取得プログラム

各ブロックの基礎パラメータ取得を高速化するため、対象プログラムに以下の修正を施す。

(1) 当該計算ブロックのみをループボディとして持つループを除き、DO 文をコメントアウトする。
(2) 他の基礎パラメータを変化させない限り、通信ブロックをコメントアウトする。

以上により作成されたプログラムを、2PU で実行し、各ブロック基礎パラメータを取得する。本実行方法を簡易実行と呼ぶ。図 1 の例では、L1、L5 のループをコメントアウトする。

3.2.2 ブロック実行回数取得プログラム

次に PU 台数が p 台時のブロック実行回数を求める方法を述べる。ここでもブロック実行回数の取得を高速化するため、対象プログラムに以下の修正を施す。

(1) プログラム中のすべてのループに対して、ブロック実行回数、すなわちループ制御変数にデータ依存する計算のみを残し、他の

実行文をコメントアウトする。

(2) MPI 通信関数も、(1)と同様、ブロック実行回数に依存する通信以外をコメントアウトする。

以上の方法で作成したプログラムを実行し、各ブロックの実行回数を求める。

最後に、求めた各ブロック 1 回分の実行時間にブロック実行回数を乗じ、総和をもってプログラム全体の予測時間とする。

4. 提案手法の評価結果

NPB2.3 のプログラム 8 本を用いて、2～128PU 時の実行時間予測を行った。以下、予測処理に必要な時間と実際の実行時間の比較、及び予測時間と実際の実行時間の比較結果を示す。

なお、評価では、NPB2.3 すべてのプログラムで class B を用いた。計算ブロックの基礎パラメータは 2PU 上で取得した値を使用するが、NPB2.3 には実行 PU 台数に制限があり、SP、BT は N^2 台数での実行であるので、基礎パラメータとして 4PU 上で取得した値を使用した。

評価に用いた実行環境を表 1 に示す。本実行環境では、16 ノード毎に 1 つのスイッチで接続し、各 8 つのスイッチを 1 つの上位スイッチでまとめるネットワーク構造をとっている。また、図 2 に本実行環境上での NPB2.3 の挙動を示す。これは 2PU 実行時を基準とした速度向上率で示すが、SP、BT は実行台数制限より、4PU 実行時を基準とした。

表 1：実行環境

Node	128
CPU	Intel Pentium4
Clock	2.4GHz
L2cache	512Kbyte
Memory	1Gbyte
Network	1000BaseTX Ethernet
Network Switch	NETGEAR Gigabit Switch
MPI library	MPICH ver.1.2.5

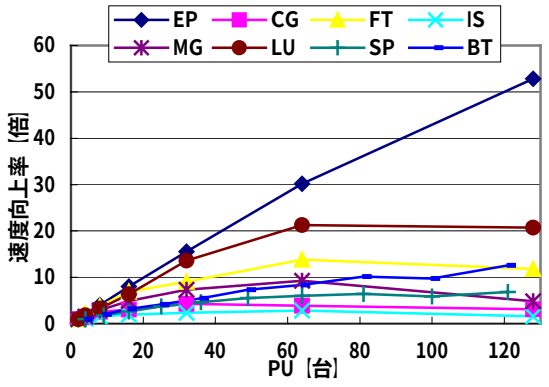


図 2：NPB2.3 の速度向上率

4.1 実行時間予測にかかる処理時間

図 3 に、予測にかかる処理時間と実際の実行時間とを比較した結果を示す。これらの値は、全実行 PU 台数の実行時間の総和である。図 3 に示すとおり、8 つのプログラムすべてにおいて、実際の実行時間と比べ $O(10^{-1})$～$O(10^{-3})$の時間で実行時間予測が可能である。

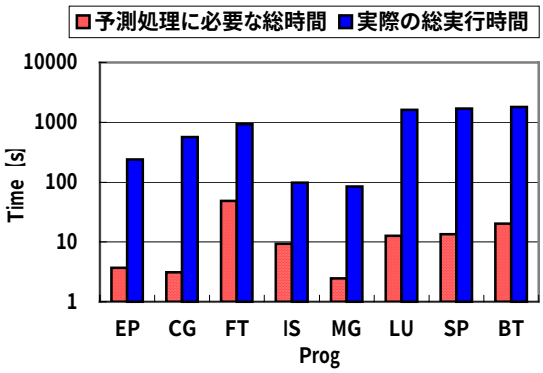


図 3：NPB2.3 各プログラムの実行時における予測処理の総時間と実際の総実行時間

4.2 実行時間予測結果

図 4 に、実際の実行時間と予測時間の比較結果を、計算部分・通信部分に分けて示す。

CG、IS を除くプログラム 6 つの実行時間予測誤差は、32PU までの実行で 10%以内に収まっている。CG、IS、MG を除くプログラム 5 つにおいては、64PU までの実行で 13%以内に収まっている。特に SP、BT においては、81PU までの実行で予測誤差は 10%以内である。

一方、CG、IS においては、4PU 以上の実行で予測誤差が 17%以上あった。

5. 考察

本節では、4.の結果をうけて実行時間予測誤差原因の考察を行う。

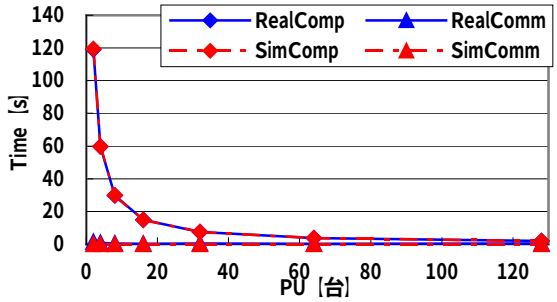
5.1 通信部分の予測誤差

CG を除く 7 つのプログラムでは、計算部分の実行時間が 10 秒以上あれば[1]、計算部分の予測誤差は 10%未満に収まっている。CG を除けば、予測誤差の主要因は通信部分の予測時間にある。

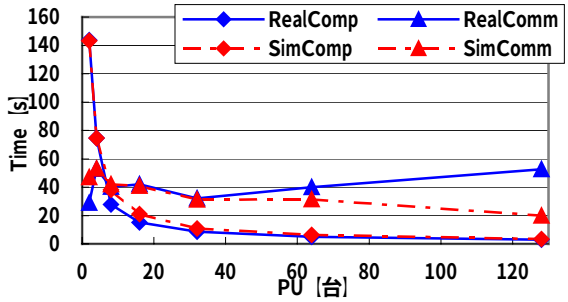
本手法における 2PU 間での疑似通信方法では、各 PU 間での通信の同期が取れていることを前提にしている。あるプロセスで送信関数を実行した時点で、通信相手のプロセスでも同時もしくは既に受信関数を実行している(図 5(a))ことを前提に、同期を取って疑似通信を行い、通信ブロックの実行時間を予測する。しかし、実際には同期が取れていない通信が存在し、通信ギャップにより通信の待ち時間が発生する場合がある（図 5(b)）。また、2PU で予測を行うため、大規模な PU 台数の実行で起こる、通信のコンテンションを考慮した予測を行うことは困難である。

通信時間が全実行時間に対して支配的である IS において、同期関数 MPI_Barrier を使用して通信時間を測定した結果を表 2 に示す。表 2 中、実際の通信時間(Treal)と、同期を取って計測した通信時間(Tsync)から見られる様に、IS においては 4PU 以上の実行において通信の待ち時間が発生していることが分かる。

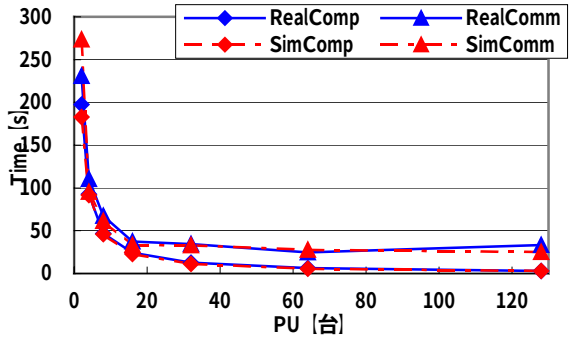
[1] 計算時間が 1 秒未満など、非常に小さい場合は相対的に予測誤差が大きくなる。



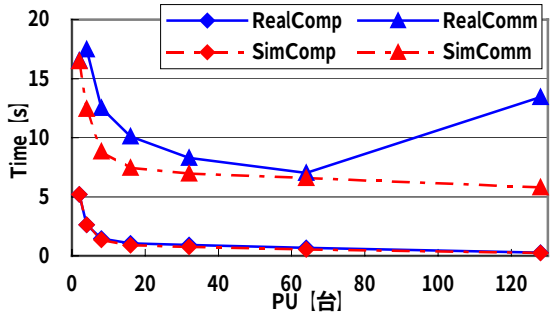
(1) EP



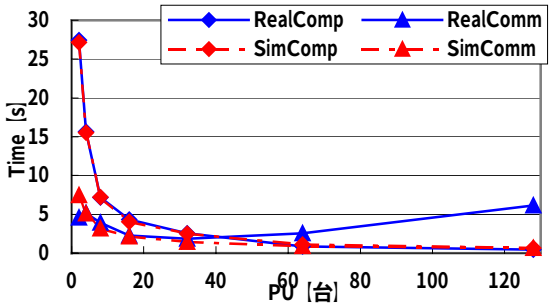
(2) CG



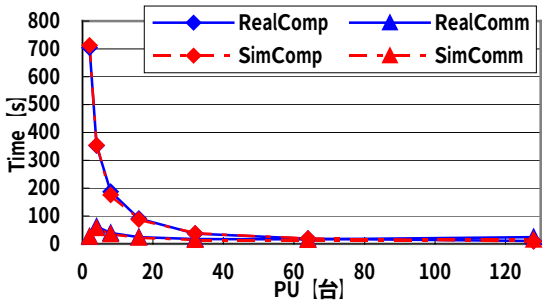
(3) FT



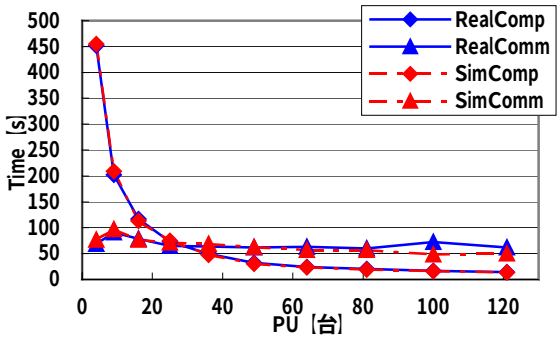
(4) IS



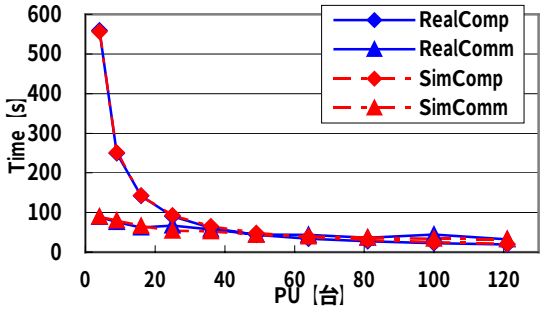
(5) MG



(6) LU



(7) SP



(8) BT

図 4：NPB2.3 計算部分・通信部分別の実行時間予測結果

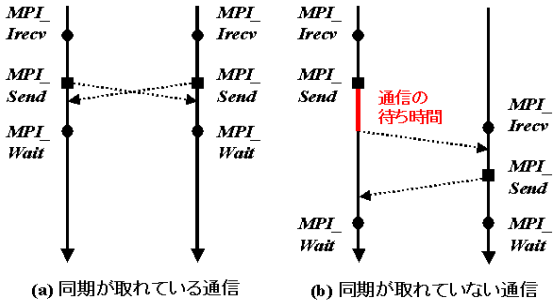


図 5：同期・非同期通信の模式図

表 2：IS の実際の通信時間、同期を取った通信時間、通信の予測時間の関係

PU	Treal	Tsync	Tsim
2	16.54	16.33	16.49
4	17.47	12.80	12.44
8	12.53	7.82	8.83
16	10.10	7.95	7.45
32	8.29	6.88	6.96
64	7.00	6.76	6.58
128	13.44	7.84	5.79

単位【s】

64PU までの Treal と Tsync の差は集団通信 MPI_Allreduce にあり、Treal では 4PU で約 3.8 秒、8PU で約 4.4 秒、16PU で約 1.5 秒、32PU で約 1.4 秒あった。一方、128PU での予測誤差原因は 64PU までとは異なり、集団通信 MPI_Alltoall にあった。IS においては、すべての実行 PU 台数で MPI_INT 型の変数を 1 個送っている。メッセージサイズが非常に小さいため、64PU までで実行時間は 0.02 秒未満であるが、128PU では約 6.1 秒あった。こちらは、本実行環境の上位ネットワークスイッチにおいて、通信のコンテンションが多く発生したためだと考える。

上記問題に対する単純な解決策として、通信の予測に 2PU を使用するのでなく、予測対象 PU 台数で通信を行い、実際に通信の待ち時間を発生させる方法が挙げられる。この際、実際の通信実行回数分すべて実行するのではなく、各通信を 1 回だけ実行し、事前に取得した実行回数を用いて予測を行い、予測にかかる処理時間の減少を図る。

IS について、この方法を用いて予測を行った結果を表 3、図 6 に示す。IS では、MPI_Allreduce の通信の待ち時間は、その手前に実行される集団通信 MPI_Alltoallv が原因で発生していたため、予測では各通信を個別に実行するのでなく、まとめて通信の実行を行った。

表 3：実際の実行時間、2PU を用いた予測処理時間、予測対象 PU を用いた予測処理時間

実際の総実行時間	2PU を用いた予測総時間	予測対象 PU を用いた予測総時間
97.63	9.30	28.06

単位【s】

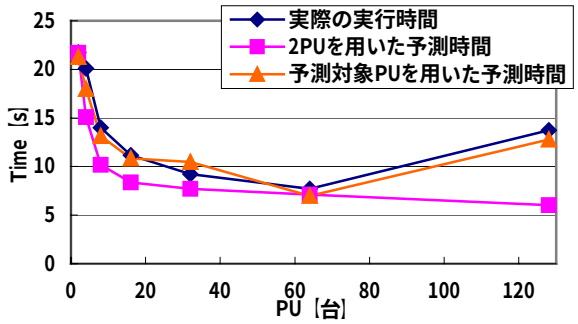


図 6：実際の実行時間、2PU を用いた予測時間、予測対象 PU を用いた予測時間の関係

表 3 より、2PU を用いた場合と比較して、通信の予測にかかる処理時間は増加したものの、図 6 に示すとおり、予測誤差に改善が見られる。また 2PU を用いた予測では特定できなかった、ピーク性能を示す PU 台数の特定も可能となった。他のプログラムにおける通信の予測誤差も、本解決策により解消が期待される。

5.2 計算部分の予測誤差

CG では 8PU 以上の実行時において、計算部分の予測誤差が大きい（表 4）。CG は他のプログラムと異なり、32PU までの予測誤差の主要因は計算部分の予測時間にある。

表 4：CG class B の実行時間・予測時間の詳細

PU	計算部分		通信部分		全体の予測誤差
	実測	予測	実測	予測	
2	143.80	143.17	28.93	35.11	3.95%
4	74.45	74.67	52.95	53.12	0.29%
8	27.80	38.20	40.17	41.99	17.97%
16	15.24	20.96	41.95	40.87	11.89%
32	8.49	10.81	32.01	31.29	7.49%
64	4.90	6.38	40.06	31.40	22.55%
128	2.88	3.37	52.74	19.79	60.14%

単位【s】

この要因は、計算時間の大部分を占める計算ブロックにおいて、配列へのランダムな参照を行っているためである（図 7）。

図 7 の最内ループにおいて、配列 colidx による配列 p への参照が行われるが、CG では配列 colidx の値が実行 PU 台数毎に異なる。そのため、配列 p への参照に関するキャッシュ効果が一定でなく、実行 PU 台数毎の計算部分基礎パラメータに差が出る（表 5）。

```
DO j=1, lastrow-fistrow+1
      sum = 0.d0
      DO k=rowstr(j), rowstr(j+1)-1
            sum = sum + a(k)*p(colidx(k))
      ENDDO
      w(j) = sum
ENDDO
```

図 7：CG 中、最も時間のかかる計算ブロック

表 5:CG で最も時間のかかる計算ブロックの実行 PU 自身の基礎パラメータ

PU	計算基礎パラメータ	計算ブロックの予測実行時間	計算ブロックの実際の実行時間
2	10.01×10^{-8}	130.05	129.89
4	10.03×10^{-8}	65.35	66.38
8	7.47×10^{-8}	32.80	24.21
16	7.60×10^{-8}	16.66	12.50
32	8.27×10^{-8}	8.30	6.77
64	8.53×10^{-8}	4.22	3.55
128	9.80×10^{-8}	2.11	2.04

単位【s】

本手法では、2PU の計算部分基礎パラメータを用いて、他の PU 台数実行時の予測を行うため、こういった場合は計算部分に関する予測誤差が大きくなる。

上記問題の解決策として、予測対象 PU 台数自身の基礎パラメータを用いて予測を行うことが挙げられる。この場合、実行 PU 台数毎の基礎パラメータ取得が必要となり、予測にかかるオーバヘッドが大きくなる。

しかし実行 PU 台数毎の基礎パラメータを取得した場合でも、各々の PU 台数で $O(10^{-1})$～$O(10^{-3})$の時間で取得が可能であり、十分実用的な処理時間での予測が可能であると考えられる。

5.3 予測誤差に関する LAPSE[3]との比較

[3]では SOR、BPS、APUCS、PGL というプログラムを対象に、64PU までの実行時間予測を行っている。予測誤差は 5～20%であり、予測処理に必要な時間は、実際の実行時間の 4 倍以上である。

本稿と評価アプリケーションが異なるため、単純な比較は困難であるが、IS の 64PU までの実行で顕著に表れた、通信の待ち時間による誤差の問題は、LAPSE に実装されている手法で解決されると考えられる。LAPSE では、実行 PU 数分すべての計算ブロックを実行し、各 PU の通信関数を実行するタイミングを予測する。これにより非同期な通信の予測にも対応するものであるが、[3]では通信時間・計算時間の内訳が記されていないため、通信時間自体の予測精度は不明である。しかし、実行 PU 数分すべての計算ブロックを実行するため、予測処理にかかる時間は、実際に実行する以上かかる。一方、我々の手法では、簡易実行により、計算ブロックの実行時間を高速に予測する。今後は我々の手法と、LAPSE の手法を組み合わせ、通信の待ち時間に対応する手法を検討していく。

6. おわりに

本稿では、[8]で提案した実行時間予測手法の有効性の評価を行った。

NPB2.3 のプログラム 8 つを用いて、2～128PU までの実行時間予測を行った結果、8 つすべてのプログラムで実際の実行時間に対し $O(10^{-1})$～$O(10^{-3})$の時間で予測が可能であった。[2][3][4]の従来手法では、予測に必要な時間は $O(1)$～$O(10)$であり、本手法の有効性が確認された。また、CG、IS を除く 6 つのプログラムで 32PU 実行時まででは、予測誤差は 10%以内であった。CG を除くプログラムでは、実行時間予測誤差の主要因は通信部分の予測時間にあり、全実行時間中、通信時間が支配的となる、IS と他のプログラム 64PU 以上の実行時においては、予測誤差は 17%以上となった。CG では他のプログラムとは異なり、32PU までの実行においては計算部分の予測誤差が問題となり、8PU 以上の実行で予測誤差は 17%以上となった。

今後の課題は、非同期な通信で発生する、通信の待ち時間に対応する手法を検討していく。これには[3]で提案された、通信関数の実行タイミングを PU 毎に予測する手法の利用が考えられる。

参 考 文 献

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

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

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

[4] Sundeep Prakash, Ewa Deelman, and Rajive Bogrodia: “Asynchronous Parallel Simulation of Parallel Programs”, IEEE Trans. on Soft ware Engineering 2000, Vol.26, pp.385-400, 2000.

[5] Maurice Yarrow, 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] 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.

[7] 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.

[8] 堀井洋, 山名早人: “実測に基づいた MPI プログラムの実行時間予測”, 情報処理学会研究報告(HPC), No.88, pp.61-66, 2001.