2004－HPC－98　(3)
2004／4／13

部分的なタスク実行による マスタスレーブ型並列プログラムの高速な性能予測

水谷泰治[†]　伊野文彦[†]　萩原兼一[†]

本稿では，マスタスレーブ型並列プログラム（MS プログラム）の実行時間を高速に予測するための手法を提案する．提案手法は，直接実行する MS プログラムのタスク数を削減することで，予測に要する時間を短縮する．これを実現するために，まず，一部のタスクのみの実行時間を直接実行によって測定し，その実行時間を線形補間することで残りのタスクの実行時間を推定する．次に，MS 法のシミュレーションによって各タスクを処理するプロセッサを決め，そのプロセッサ上でのタスクの実行時間を推定する．提案手法により，実測の実行時間と比較して 8%以内の誤差で，かつ実測の実行時間の 1/8 以下の時間で MS プログラムの実行時間を予測できた．

Fast Performance Prediction of Master/Slave Programs by Executing Subset of Tasks

Yasuharu Mizutani,[†] Fumihiko Ino[†] and Kenichi Hagihara[†]

In this paper, we propose a method that is capable of rapidly predicting the performance of master/slave (MS) parallel programs. We realize a fast performance prediction of MS programs by reducing the number of tasks executed directly. In order to estimate the execution time of all tasks, we use the following two methods. One is linear interpolation of the time which is measured by executing few tasks, and the other is a simulation of MS paradigm to determine which task is assigned to which slave. The experimental result shows that our method predicts the execution time of MS programs within 8% error. Furthermore, it takes a factor of 8 less time compared with the measured execution time.

1. はじめに

近年，クラスタ計算およびグリッド計算に関する技術の発展に伴い，演算性能および通信性能が不均一な並列計算環境の利用が広まっている．このような環境に適したプログラミングパラダイムとしてマスタスレーブ（MS）法がある．MS 法ではプロセッサ集合をマスタとスレーブに役割分担することで動的負荷分散を実現する．マスタはスレーブへの仕事の割り当てを管理し，スレーブは割り当てられた仕事を処理する．マスタがスレーブに割り当てる仕事をタスクと呼ぶ．

MS 法に基づく並列プログラム（MS プログラム）の性能は，タスク粒度，タスクの割当順序，マスタ数およびスレーブ数というパラメータに依存する．これらのパラメータの適切な値は並列計算環境の演算性能や通信性能に依存し，不適切な値は MS プログラムの性能を低下させる．並列計算環境ごとに適切な値を調べるための手段として，性能予測は有用である．

† 大阪大学大学院情報科学研究科コンピュータサイエンス専攻
Department of Computer Science, Graduate School of Information Science and Technology, Osaka University

性能予測によってパラメータの適切な値を調べる場合，予測の高速化は重要である．なぜならば，性能予測するパラメータの組合せは多く，性能予測が低速であればパラメータの適切な組合せの発見に多くの時間を要するためである．

これまでに並列プログラムの性能予測手法として，直接実行方式[1)~4)] と簡易実行方式[5),6)] が提案されている．直接実行方式では，並列プログラムを実行することでその動作を正確に再現する．したがって，性能を精度よく予測できるが，少なくともプログラムの実行と同等の時間を性能予測に要する．一方，簡易実行方式では直接実行方式に比べて高速に性能予測できる．プログラムの処理フローおよびループ構造を静的に解析し，ループの繰り返し 1 回分の処理を実行することで，ループ全体の実行時間を推定する．

既存の簡易実行方式は，以下の 2 つの理由により MS プログラムの性能を予測を正確にできない．まず，MS プログラムのタスクがループからなるとは限らない．また，ループからなる場合も，繰り返し回数が計算結果に依存するなどの理由により，簡易実行によってループ全体の実行時間を精度よく推定できない．次に，タスクの割当先は動的に決まるため，プログラム

の実行前にどのスレーブがどのタスクを処理するかわからない．したがって，特に性能不均一な並列計算環境では，ループ 1 回分の実行時間としてどのスレーブでの測定結果を用いればよいかわからない．

本論文では，これらを解決する高速な性能予測手法を提案する．提案手法は，簡易実行の処理単位をタスクとし，測定したタスクの実行時間を線形補間することで残りのタスクの実行時間を推定する．また，推定した実行時間を用いて MS 法の動作をシミュレーションすることで，タスクを実行するスレーブを特定する．

以降では，まず 2 章で既存の性能予測手法およびそれらを MS プログラムに適用する上での問題をまとめる．次に，3 章で提案する性能予測手法を述べる．その後，4 章で提案手法の性能予測に要する時間および予測精度を評価し，5 章でまとめを述べる．

2. 関 連 研 究

直接実行方式の 1 つとして，MPI-SIM[1] が提案されている． MPI-SIM は，並列離散事象シミュレーションによってメッセージ通信仕様 MPI を使用した並列プログラムの実行時間を予測する．MPI-SIM は，並列プログラムを直接実行しながら計算，送信および受信という 3 種類のイベントを取得し，各々のイベントに対してシミュレータ内部の仮想時計を進めることで並列プログラムの実行時間を予測する．MPI-SIM は，シミュレーションを並列実行することで性能予測に要する時間の短縮を実現するが，並列プログラムを直接実行するため，実測の実行時間と性能予測に要する時間との比（S_f, slowdown factor）は一般に 1 以上となる．また，CLUE[2], Anastasia[3] および MSE[4] も直接実行方式に基づいて性能予測する．直接実行方式は，プログラムの動作を詳細に再現できるが， S_f が大きくなるという問題がある．

一方，簡易実行方式[5),6)] は，ループの繰り返し数が固定的に定まる並列プログラムに対して S_f を 1/1000 から 1 程度にでき，高速に性能予測できる．

既存の簡易実行方式は，処理の流れが一定，ループの繰り返し回数がループ開始前にわかるような並列プログラムに対して有効である．しかし，MS プログラムにおいては，処理の流れやループの繰り返し回数は静的またはそれらの実行直前にはわからないことが多い．なぜならば，タスクにおけるループの繰り返し回数は計算結果に依存することが多いためである．そのため，既存の簡易実行方式では，ループ 1 回あたりの実行時間がわかっても，プログラム全体の実行時間を正確に予測できない．

上述の簡易実行方式に対して，本研究では簡易実行の単位をタスクとする．一部のタスクを実行することでそれらの実行時間を測定し，残りのタスクの実行時間を推定する．そして，推定した実行時間を用いて MS 法の動作をシミュレーションすることでタスクを処理するスレーブを特定し， MS プログラムの実行時間を予測する．

3. 提案する性能予測手法

提案手法は，以下の M1 および M2 からなる．

M1：一部のタスクの実行による実行時間の推定

M2：並列計算モデルを用いた MS 法のシミュレーション

M1 では，まず MS プログラムの全タスクからなる集合の部分集合を実行（**部分実行**）することで，その部分集合に属する各タスクの実行時間を測定する．次に，後述する線形補間によって，測定した実行時間から残りのタスクの実行時間を推定する．

M2 では，MS 法の動作をシミュレーションすることで，MS プログラムの実行時間を予測する．このとき，高速に性能予測するには，マスタとスレーブの処理に係る通信時間を高速に見積もる必要がある．そのために，並列計算モデルを利用する．

以降では，まず本研究におけるタスクについて述べ，M1 および M2 による性能予測の詳細について述べる．

3.1 タスク集合

本研究では，タスクの入力パラメータによって MS プログラム中の各タスクを識別する．N 個の入力パラメータをもつタスクを N 組 $(i_1, \cdots, i_N)$ と表し，この N 組を要素とする集合 $\mathcal{I}$ を以下のように定義する．

$$\mathcal{I} = \{(i_1, i_2, \cdots, i_N) | \ 1 \leq i_k \leq c_k, \ 1 \leq k \leq N\}$$

$\mathcal{I}$ を**タスク集合**と呼ぶ．c_k は，k 次元目 $(1 \leq k \leq N)$ の引数が取り得る入力値の総数を表す．ここで，i_k が取り得る値の集合を I_k と表す．

部分実行するタスクの集合 $\mathcal{I}_s$ は，後述する線形補間の処理を簡単にするために以下のように選択する．

$$\mathcal{I}_s = \{I_{s,1} \times \cdots \times I_{s,N} \mid I_{s,k} \subseteq I_k, 1 \leq k \leq N\}$$

ここで，演算子 $\times$ は集合の直積を表す．

3.2 部分実行によるタスクの実行時間の線形補間

対象問題の各タスクの実行時間を推定するとは， $\mathcal{I}$ に対して，$\{T_N(\boldsymbol{i}) \mid \boldsymbol{i} \in \mathcal{I}\}$ を求めることである．ここで， $T_N(\boldsymbol{i})$ は $\boldsymbol{i}$を入力とするタスクの実行時間を表し，以下の漸化式（1）で定義する．

$$\begin{aligned} T_k(\boldsymbol{x}) = (p_{N-k+1}(\boldsymbol{x}) - 1) \cdot T_{k-1}(\boldsymbol{b}_{N-k+1}) + \\ p_{N-k+1}(\boldsymbol{x}) \cdot T_{k-1}(\boldsymbol{u}_{N-k+1}) \end{aligned} \quad (1)$$

この式は N 次の線形補間を表す．ここで，

$$\begin{aligned} \boldsymbol{x} &= (x_1, x_1, \cdots, x_N), \\ \boldsymbol{b}_k &= (x_1, \cdots, B(x_k), \cdots, x_N), \\ \boldsymbol{u}_k &= (x_1, \cdots, U(x_k), \cdots, x_N), \\ p_k(\boldsymbol{x}) &= (x_k - B(x_k))/(U(x_k) - B(x_k)), \\ B(x_k) &= \max(\{i \mid i \leq x_k \text{ かつ } i \in I_{s,k}\}), \\ U(x_k) &= \min(\{i \mid x_k \leq i \text{ かつ } i \in I_{s,k}\}) \end{aligned}$$

である．$\boldsymbol{b}_k$ と $\boldsymbol{u}_k$ の定義から $T_0(\boldsymbol{x})$ は部分実行によって測定した実行時間となる．

図 1 に部分実行と線形補間によってタスクを実行時間を推定する例を示す．この例では $N = 1$，$c_1 = 10$，$\mathcal{I}_s = \{1, 4, 7, 10\}$ である．まず，$\mathcal{I}_s$ に含まれるタスクを実行することでそれらの実行時間を測定する．次に，測定した実行時間を線形補間することで，残りの

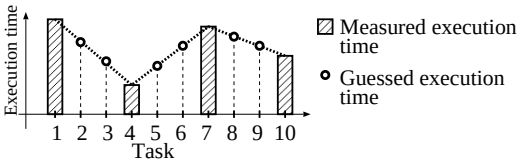


図 1 部分実行と線形補間によるタスクの実行時間の推定

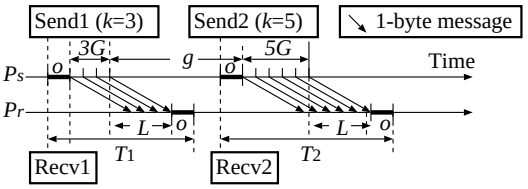


図 2 LogGP モデルにおける 1 対 1 通信の例

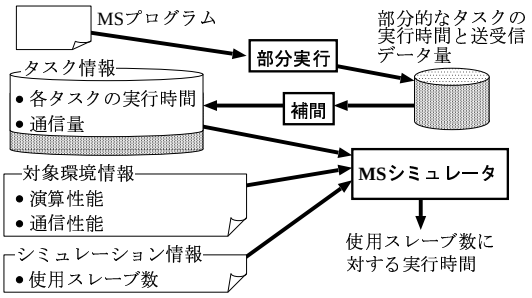


図 3 シミュレーションによる性能予測の流れ

タスクの実行時間を推測する．

3.3 MS 法のシミュレーション

推定したタスクの実行時間と並列計算モデルを用いて， MS 法の動作をシミュレーションする．

以降では，まず並列計算モデルおよび並列計算モデルを用いた MS 法のシミュレーションについて述べる．

3.3.1 並列計算モデルによる通信時間の予測

並列計算モデルでは，並列計算環境の通信性能をパラメータ化し，そのパラメータを用いた式によって通信時間を表現するため，通信性能を高速に予測できる．

本研究では，LogGP[7] の改良モデル[4] を利用する．LogGP モデルは， 通信遅延（L），通信オーバヘッド（o），メッセージを連続して送受信できる最短の時間間隔（g），長いメッセージを送信する際の 1 バイトあたりの時間間隔（G）， プロセッサ数（P）の 5 つのパラメータによって通信をモデル化する．改良モデルでは，到着メッセージの検査オーバヘッドを表現するために o をプロセッサ数の線形関数とする[4]．

図 2 に，LogGP モデルにおける 1 対 1 通信の例を示す．図 2 はメッセージ長 k が 3 の通信（Send1）と 5 の通信（Send2）を表している．メッセージ長 k の通信に要する時間は $o + kG + L + o$ となる．

3.3.2 シミュレーションの流れ

図 3 に性能予測の流れを示す．図 3 において矢印は入出力を表す． MS シミュレータ（MSS）の入力はタスク情報，対象環境情報およびシミュレーション情報の 3 種類である．タスク情報は，MS プログラムにおける個々のタスクの実行時間と通信量を表す．ユーザはまず， MS プログラムを部分実行するために，MS プログラムを修正する．次に，そのプログラムを直接実行することで，部分的なタスクの実行時間を得る．そして， 3.2 節で述べた手順によって，実行時間を線形補間する．対象環境情報は，性能予測の対象となる並列計算環境（**対象環境**）の演算性能と通信性能を表す．演算性能は，部分実行に用いる計算機との相対的な演算速度比を記述する．通信性能は， LogGP モデルのパラメータ値を記述する．シミュレーション情報はスレーブ数などの MS 法のパラメータを表す．

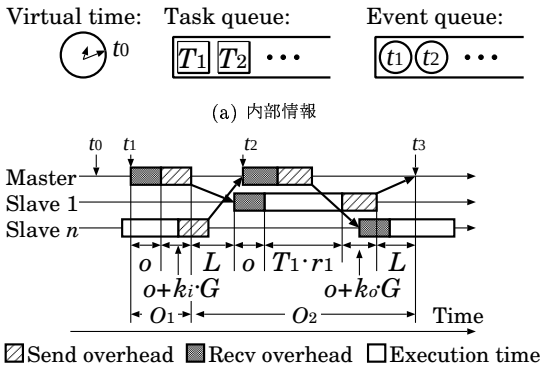


図 4 MS プログラムのシミュレーション例

MSS は内部に仮想時刻，タスクキュー $\mathcal{Q}_T$ およびイベントキュー $\mathcal{Q}_E$ をもつ．仮想時刻はシミュレーションにおいて MS プログラムの動作を開始してからのマスタの経過時間を表す．$\mathcal{Q}_T$ はタスク情報の各タスクを割当順に整列して格納する．$\mathcal{Q}_E$ はマスタにおいて発生するイベントを発生時刻順に整列して格納する．イベントはスレーブからのメッセージ到着であり，イベントの発生時刻はマスタへのメッセージ到着する時刻である．

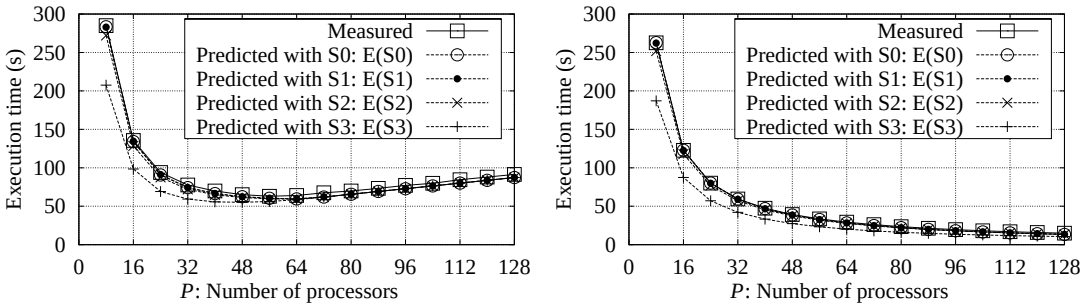
MS プログラムの実行時間は，$\mathcal{Q}_T$ と $\mathcal{Q}_E$ がともに空になるまで以下の 3 つを繰り返すことで予測する．

(P1) $\mathcal{Q}_E$ から最も早い発生時刻のイベント e を取り出す．e の発生時刻が仮想時刻よりも先であれば，仮想時刻を e の発生時刻とする．また，e の受信に対応する受信オーバヘッドに係る時間分だけ仮想時刻を進める．

(P2) $\mathcal{Q}_T$ からタスク T を取り出す． $\mathcal{Q}_T$ が空であるならば（P1）に戻る．

(P3) マスタが T の処理結果を受信するイベントを $\mathcal{Q}_E$ へ追加する．イベントの発生時刻は，並列計算モデルによって求めた通信時間と T の処理時間の合計より求める．また，T の割当に対応するマスタの送信オーバヘッドに係る時間分だけ仮想時刻を進める．

図 4 に MSS の動作例を示す．ここでは，仮想時刻が t_0 であり，$\mathcal{Q}_E$ には発生時刻が t_1 と t_2 のイベントがあるものする．まず，マスタは $\mathcal{Q}_E$ からイベント



(a) イーサネット　　(b) ミリネット

図 5　Mandelbrot 集合探索における予測精度の比較

を取り出し，仮想時刻を t_1 に進める．次に，以下に示す 2 つの時間 O_1，O_2 を算出し，マスタがスレーブからの処理結果を受信する時刻 t_3 を $\mathcal{Q}_E$ に追加する．ここで，$t_3 = t_1 + O_1 + O_2$ である．

O_1：　マスタにおけるメッセージの送受信に係るオーバヘッド $2o + k_i \cdot G$．ここで k_i は通信量を表す．

O_2：　マスタがメッセージを送信してから，タスクの処理結果がマスタに到達するまでの時間 $2L + T_1 \cdot r_1 + 2o + k_o \cdot G$．ここで，$T$ はマスタが $\mathcal{Q}_T$ から取り出したタスクの実行時間を表す．r_s はスレーブ s と部分実行に用いた計算機の演算速度比を表す．

そして，マスタは仮想時刻を O_1 だけ進め，以後のイベントを同様に処理する．

$\mathcal{Q}_E$ への e を挿入は，$\mathcal{Q}_E$ の末尾から e をシフトすることで実現する．これは，スレーブ数が過剰になったときに，e の挿入に要する時間を抑えるためである．詳細は 4.2 節で述べる．

4. 評価実験

本章では，予測精度および予測に要する時間（**計算時間**）の観点から提案手法を評価する．

4.1 実験環境

対象問題としてマンデルブロ集合探索を用いた．タスクは複素平面上の 1 点に対するマンデルブロ集合への包含判定であり，タスク総数は $1,048,576$ である．

性能予測の対象環境は 64 台の PC からなる PC クラスタである．各 PC はミリネット高速ネットワーク[8] およびイーサネットで接続しており，通信バンド幅はそれぞれ 2Gb/s および 100Mb/s である．各 PC は Pentium III 1 GHz を 2 台もち，PC クラスタ全体として 128 台の CPU をもつ．MSS は PC クラスタの 1 台を用いて動作させた．

本実験では，部分実行するタスクの数の違いによる予測精度および計算時間の影響を調べるために，4 種類のタスク集合 S0，S1，S2 および S3 を用いた．各タスク集合は，部分実行するタスクの数が異なる．**表 1** に各タスク集合の詳細を示す．表 1 においてタスクの合計実行時間は，線形補間によって推定したタスクの実行時間の合計である．

表 1　実験に使用したタスク

タスク集合	部分実行したタスクの総数	全タスク数に対する比率	タスクの総実行時間 A（秒）
S0	1,048,576	1	1820
S1	65,536	1/16	1816
S2	1,024	1/1024	1744
S3	16	1/65536	1292

表 2　実測実行時間と予測実行時間との最大誤差（%）

タスク集合	イーサネット	ミリネット
S0	8.1	4.0
S1	8.1	4.2
S2	8.2	4.2
S3	27	29

誤差 $= |100 \cdot (\text{実測実行時間} - \text{予測実行時間})/\text{実測実行時間}|$

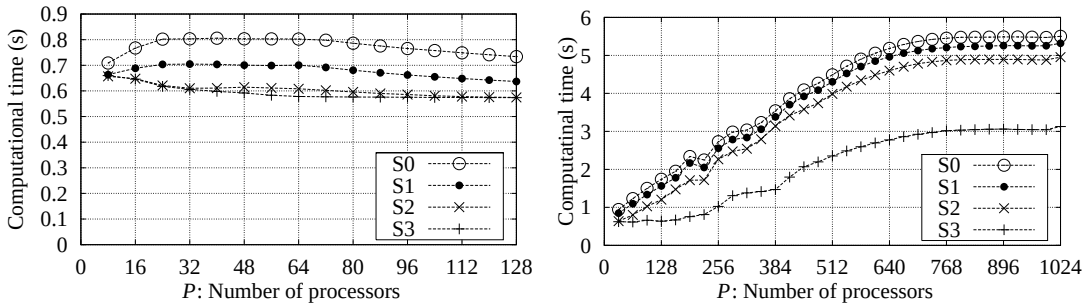
4.2 予測精度

以降では，タスク集合 Sj を用いて予測した実行時間を E(Sj) と表す．**図 5** に実測と予測の実行時間の比較を示す．また，**表 2** に予測実行時間と実測実行時間の最大誤差を示す．表 2 より，S0，S1 および S2 を用いて予測した実行時間との最大誤差は 8.2% と小さい．一方，S3 については最大誤差は 29% と大きい．このように，本実験においては，部分実行するタスク数が全タスク数の 1/1024 以上であれば，よい精度で実行時間を予測できる．

実測と予測の実行時間のずれは，シミュレーションに起因するものと，部分実行するタスク数の違いに起因するものの 2 種類が考えられる．前者に起因するずれは実測性能と E(S0) の差であり，後者に起因するずれは E(S0) と E(Sj)($j = 1, 2, 3$) の差である．

表 2 において，実測実行時間と E(S0) の誤差は小さい．よって，シミュレーションに起因するずれは小さいといえる．このずれの原因は，並列計算モデルによって予測した通信時間が，実際の通信時間とずれるためである．したがって，MSS で使用する並列計算モデルを変更または改良することで，シミュレーションに起因するずれを減少できる．

表 3 に，E(S0) と各タスク集合を用いた予測実行時間との最大誤差を示す．E(S1) と E(S0) の最大誤差は 0.5% であり，E(S2) と E(S0) との最大誤差は 4.3% で



(a) イーサネット　　(b) ミリネット

図 6　マンデルブロ集合探索における予測速度の比較

表 3　S0 を用いた予測実行時間（E(S0)）との最大誤差（%）

タスク集合	イーサネット	ミリネット
S1	0.5	0.2
S2	4.3	4.1
S3	27	29

誤差 $= |100 \cdot (\mathrm{E(S0)} - \text{予測実行時間}) / \mathrm{E(S0)}|$

表 4　タスク集合の総実行時間の差と予測実行時間の差の比較

タスク集合	P	D (*) (秒)	E(S0) との差（秒）	
			イーサネット	ミリネット
S1	8	0.6	0.5	0.6
	32	0.1	0.1	0.1
	128	0.0	0.0	0.0
S2	8	11	11	11
	32	2.5	2.2	2.4
	128	0.6	0.0	0.3
S3	8	75	76	76
	32	17	15	17
	128	4.2	0.00	3.5

(*) $D = (1820 - A)/(P - 1)$．ここで，A は表 1 における各タスク集合の総実行時間．1820 はタスク集合 S0 の A．

ある．これらの誤差は十分小さいと考える．しかし，S3 の最大誤差は 29%と大きい．この理由は，S3 と S0 のタスクの総実行時間 A（表 1）の差が 528 秒と大きいためである．**表 4** に，A の差がスレーブ 1 個あたりに与える時間 D と予測実行時間の差との比較を示す．表 4 より，S0 と各タスク集合との予測実行時間の差は，D とよく適合することがわかる．したがって，本実験においては，E(S0) と各タスク集合の予測実行時間の誤差 X（%）は，以下の式 (3) で見積もることができる．

$$X = 100 \cdot D/\{1820/(P-1)\} \quad (2)$$
$$= (1820 - A)/18.2 \quad (3)$$

ここで，式 (2) の分母は，プロセッサ数 P において通信時間を無視した S0 の予測実行時間を表す．式 (3) に S1，S2 および S3 の A の値を代入すると，それぞれ X は 0.2%，4.2%，29%となり，表 3 の結果とよく適合する．以上の議論より，本実験で用いた MS プログラムに対しては，E(S0) との誤差を x%以内とするためには，式 (3) より $A > 1820 - 18.2x$ を満たすタスク集合を用いればよい．

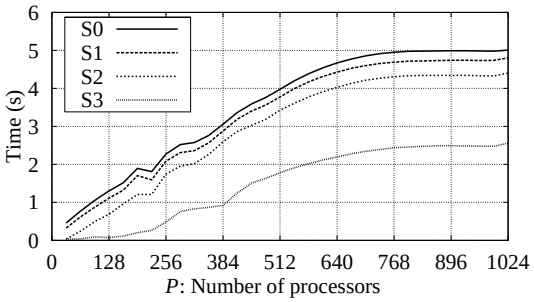


図 7　ミリネットにおける $\mathcal{Q}_E$ へのイベント挿入時間

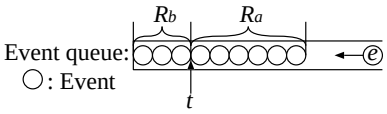


図 8　イベントキューへのイベントの挿入

4.3　予 測 速 度

図 6 に性能予測に要した時間（計算時間）を示す．実測との実行時間比（S_f：計算時間／実測実行時間）の最大値は，図 6(a) では $P = 56$ における 0.012 であり，図 6(b) では $P = 128$ における 0.12 である．このように，提案手法では，$S_f < 1$ とでき，本実験においては，実測実行時間と比べて 8 倍以上高速に性能予測できることがかわる．

表 5 に図 5 のデータ取得に要した時間を示す．表 5 より，S1，S2 および S3 では，イーサネットおよびミリネットともに実測との比 R は 1 より大きい．このように，提案手法はタスクの推定に要する時間を含めても実測よりも高速な性能予測でき，有用である．

図 6(b) では，$P < 640$ において計算時間の増加は P に対してほぼ線形である．一方，$P \geq 640$ では計算時間の増加はほとんど無い．これは，$\mathcal{Q}_E$ へのイベント挿入に要する時間が変化するためである．**図 7** に，図 6(b) において $\mathcal{Q}_E$ へのイベント挿入に要した時間を示す．図 7 より，計算時間の多くは $\mathcal{Q}_E$ へのイベント挿入に要する時間であることがわかる．

計算時間と予測実行時間は，以下の理由により同じ P において飽和すると考えられる．仮想時刻が t のとき，イベント e を $\mathcal{Q}_E$ に挿入する場合を考える．$\mathcal{Q}_E$

表 5 $P=8$ から $P=128$ まで 8 刻みの性能予測に要した時間の総計（秒）および実測との比率

		タスク推定		イーサネット			ミリネット		
		部分実行	線形補間	計算	合計 S	実測比 R	計算	S	R
実測		N/A	N/A	1485	1485	1	831	831	1
予測	S0	1820	N/A	12.4	1832	0.81	22.7	1843	0.45
	S1	114	2.59	10.9	127	11.7	20.8	137	6.06
	S2	1.71	2.60	9.67	14.0	106	13.8	21.1	39.4
	S3	0.020	2.78	9.48	12.3	121	13.2	16.0	51.9

実測における計算は，各 P における実行時間の合計．予測における計算は，各 P における計算時間の合計とタスク情報の読込時間（3.00 秒）の和．S = 部分実行 + 線形補間 + 計算．R = 実測の S/S.

表 6 ミリネットでの性能予測においてイベントの挿入に要した総シフト回数（単位：10^6 回）

タスク集合	P		
	64	256	1024
S0	6	20	70
S1	5	18	67
S2	2	14	63
S3	0.2	6	38

ではイベントは発生時刻順に整列しており，**図 8** に示すように t よりも後および前の発生時刻をもつイベントの数をそれぞれ R_a および R_b とする．3.3.2 節で述べた手順より，e の発生時刻は t 以後となるため，e を $\mathcal{Q}_E$ に挿入する計算量は $O(R_a)$ となる．R_b はマスタが t までに処理できないイベントの数であり，$R_b > 0$ のときスレーブ数は過剰である．このとき，P の増加は R_b の増加につながり，R_a は増加しない．すなわち，e の挿入に要する時間は増加しない．このように，計算時間と予測実行時間は同じ P において飽和すると考えられる．

図 6 では，部分実行したタスクの総数が多いほど，計算時間は長い．この理由は，**表 6** に示すように，部分実行したタスクの総数が多いほど e に挿入に係るイベントのシフト回数が増加するためである．3.3.2 節の e の追加手順より，イベントの発生時刻はタスクの実行時間に依存する．したがって，タスク実行時間のばらつきが大きいほど e の発生時刻もばらつき，$\mathcal{Q}_E$ に挿入される位置も様々となる．その結果，挿入に係るイベントのシフトの回数が増加し，計算時間は増加する．一方，部分実行したタスク数が少ないほど，線形補間は粗くなりタスクの実行時間のばらつきは減少する．したがって，e の挿入に係るシフト回数が減少し，計算時間は短くなる．

5. おわりに

本稿では，MS プログラムの実行時間を高速に予測するための手法について述べた．提案手法では，MS プログラムを部分実行し，その結果から推定したタスクの実行時間を用いてシミュレーションする．また，評価実験において，提案手法は実測実行時間と 8%以内の精度で，かつ実測実行時間の 8 倍以上の速度で実行時間を予測できることを示した．今後の課題として，部分実行するタスクの数と予測精度の関係についての考察が挙げられる．

謝辞 本研究の一部は，科学研究費補助金基盤研究 (C)(2)(14580374)，若手研究 (B)(15700030)，および NEC ネットワークス開発研究所の補助による．

参考文献

1) Prakash, S., Deelman, E. and Bagrodia, R.: Asynchronous Parallel Simulation of Parallel Programs, *IEEE Trans. Software Enginnering*, Vol. 26, No. 5, pp. 385–400 (2000).
2) Kvasnicka, D. F., Hlavacs, H. and Ueberhuber, C. W.: Simulating Parallel Program Performance with CLUE, *Proc. 2001 Int'l Symp. Performance Evaluation of Computer and Telecommuication Systems (SPECTS'01)*, pp. 140–149 (2001).
3) 鈴木雄大, 柴田俊介, 大野和彦, 中島浩: メガスケール環境シミュレータ Anastasia における詳細シミュレーション, 情報処理学会研究会報告 2003-HPC-95, pp. 155–160 (2003).
4) Mizutani, Y., Ino, F. and Hagihara, K.: Evaluation of Performance Prediction Method for Master/Slave Parallel Programs, *IEICE Trans. Information and Systems*, Vol. E87-D, No. 4 (2004).
5) Adve, V. S., Bagrodia, R., Deelman, E. and Sakellariou, R.: Compiler-Optimized Simulation of Large-Scale Applications on High Performance Architectures, *J. Parallel and Distributed Computing*, Vol. 62, No. 3, pp. 393–426 (2002).
6) 岩渕寿寛, 堀井洋, 山名早人: 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, 情報処理学会研究会報告 2003-HPC-95, pp. 131–136 (2003).
7) Alexandrov, A., Ionescu, M.F., Schauser, K.E. and Scheiman, C.: LogGP: Incorporating Long Messages into the LogP Model for Parallel Computation, *J. Parallel and Distributed Computing*, Vol. 44, No. 1, pp. 71–79 (1997).
8) Boden, N.J., Cohen, D., Felderman, R.E., Kulawik, A. E., Seitz, C. L., Seizovic, J. N. and Su, W.-K.: Myrinet: A Gigabit-per-Second Local-Area Network, *IEEE Micro*, Vol. 15, No. 1, pp. 29–36 (1995).