大規模データ並列プログラムの性能予測手法とNPB 2.3の性能評価

久保田 和人[†,☆] 板倉 憲一[††]
佐藤 三久[†] 朴 泰祐[††]

大規模な並列プログラムの挙動を予測する手法と，この手法を用いたNAS Parallel Benchmark（NPB）ver. 2.3の解析結果について述べる．提案手法は，インスツルメンテーションツールEXCITで取得した通信トレースファイルを，INSPIREを用いて生成したネットワークシミュレータで解析するものである．キャッシュサイズやネットワークのバンド幅といった様々な並列計算機のハードウェアパラメータを変化させた場合の並列プログラムの挙動を，千台規模のプロセッサ台数まで予測できる．実例としてNPB ver. 2.3のMGとLUを解析した．MG，LUともにキャッシュサイズが小さくても，さほど性能低下が起こらないことが分かった．また，通信バンド幅の影響は少ないが，通信オーバヘッドは，全体の性能に大きく影響を与えることが分かった．

Large-scale Parallel Program Simulation Environment and Performance Analysis of NAS Parallel Benchmarks 2.3

KAZUTO KUBOTA,[†,☆] KEN'ICH ITAKURA,[††] MITSUHISA SATO[†] and TAISUKE BOKU[††]

Large-scale data parallel program simulation technique is described. Message passing trace data of a parallel program is collected by the instrumentation tool EXCIT and it is analyzed by the network simulator which is generated by INSPIRE. The behavior of parallel programs on thousands of processors can be simulated. The LU and MG NAS Parallel Benchmarks 2.3 are simulated and analyzed. The influences of various conditions such as cache size and network bandwidth are measured. Experimental results show that small cache size is enough to execute these benchmark programs effectively. Network start up time affects the total execution time, however, the influence of network bandwidth is little observed on these benchmarks.

1. まえがき

本論文では，インスツルメンテーションツールとネットワークシミュレータを用いて大規模なデータ並列プログラムの挙動を予測する手法について論じ，本手法をNAS Parallel Benchmark 2.3に適用した結果を示す．並列プログラムはプログラムごとに異なった特徴を持ち，並列計算機との相性が実行時間に大きく影響を与える．個々の並列プログラムの実行時間に，ハードウェア資源の何が影響を与え，PU（Processing Unit）台数によってどう変化するかという特徴を明らかにすることは，ユーザが適切なプラットフォームやPU台数を選択したり，プログラムにチューニングを施す際の有効な指針となる．しかしながら，PU台数が大規模な場合，とりわけPU台数が千台を超えた場合には，並列計算機の動作を詳細にシミュレーションするのは困難である．なぜなら，膨大な処理時間を要したり，大規模なプラットフォームが必要であったりするからである．

提案手法ではインスツルメンテーションツールでプログラムにコードを挿入し，実行時に，時間情報を含む通信トレースファイルを生成する．これをネットワークシミュレータで解析することで，並列プログラムの挙動を予測する．予測にあたって，対象となるプログラムを計算区間と通信区間という2つの区間に分割する．また，実際の通信は行わずに予測を行う．これらにより，本手法は，以下のような特徴を持ちうる．

- 通信の内容によって処理のフローが変化しない

† 新情報処理開発機構
Real World Computing Partnership
☆ 現在，株式会社東芝研究開発センター
Presently with Toshiba Research and Development Center
†† 筑波大学
University of Tsukuba

データ並列プログラムを対象とする．

- 大規模なワーキングセット，千台規模の PU 台数を持つシステムを予測可能．
- PU 性能，ネットワーク性能をパラメータ化する．
- 通信と計算のオーバラップを考慮できる．

本手法を NAS Parallel Benchmark（NPB）[1] ver. 2.3 の class B 問題に適用し，PU 台数を千台規模まで変化させた場合について評価した．Kernel 問題から MG を，Application 問題から LU 問題を選んだ．PU 性能に関してはキャッシュサイズを，ネットワーク性能に関しては，トポロジ，バンド幅，通信オーバヘッドを変化させた．

本論文は以下の構成をとる．2 章では，インスツルメンテーションツール EXCIT[2] と EXCIT を用いた実行時間予測について述べる．3 章では，ネットワークシミュレータ生成系 INSPIRE[3] について述べる．4 章では，EXCIT と INSPIRE を用いた並列プログラムの性能予測手法について述べる．5 章では，NPB ver. 2.3 の MG と LU について予測結果を示し，それぞれの特徴を述べる．6 章では，考察を行い，7 章では，関連研究について述べる．8 章では，まとめと今後の課題について述べる．

2. インスツルメンテーションツール EXCIT

インスツルメンテーションツールは，ソースコードに付加的な情報取得のためのコードを埋め込んで，プログラム実行時にデータを収集するツールである．オブジェクトコードがダイレクトに実行されるので，シミュレーションでの情報収集より高速である．今回のデータ並列プログラムの性能予測では，実行サイクル数を実行時に計算するためのコードを挿入する．以下に，インスツルメンテーションツール EXCIT[2] を用いたコード挿入と実行時の実行サイクル数計算について述べる．

2.1 EXCIT と実行サイクル計算用コードの挿入

EXCIT[2] の構成を図 1 に示す．xxx.s はパーザーによって中間構造（intermediate representation）へ変換される．中間構造は解析部（analyzer）で解析され，インスツルメントするための情報が挿入される．今回は，

- アセンブラソースコードの基本ブロックへの分割
- 基本ブロックの先頭と load/store 命令に対するフック関数の挿入

を行う．基本ブロックの先頭では，基本ブロックカウントルーチンが，load/store 命令では，キャッシュシミュレータが呼ばれる．コード生成部（code generator）は，この結果を基に情報収集のためのコードを挿入した xxx_.s を生成する．xxx_.s は，キャッシュシミュレータ，および基本ブロックカウントルーチンとリンクされ実行オブジェクト（a.out）が生成される．

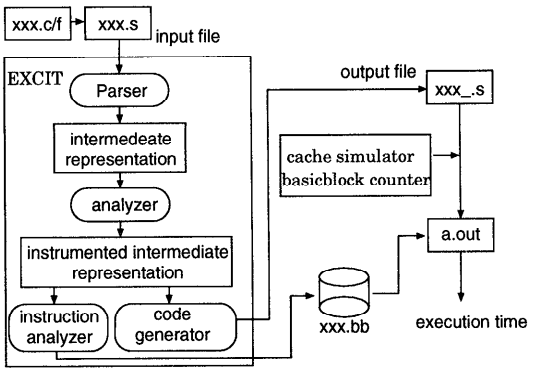


図 1 EXCIT による実行サイクル計算
Fig. 1 Execution cycle calculation by EXCIT.

命令解析部（instruction analyzer）は，各基本ブロックの標準実行サイクルを求め xxx.bb を生成する．標準実行サイクルとは，

- インストラクション/データキャッシュはすべて hit
- 浮動小数点演算命令の latency は typical 値
- 例外は起こらない

という前提で，それぞれの基本ブロックが処理に要するサイクル数と定める．すべての基本ブロックについて，1 度ずつインストラクションのシミュレーションを行うことで求める．

2.2 実行時の実行サイクル計算

実行サイクル数を計算する区間は，ユーザによってあらかじめ指定されている．実行サイクル数は，通過した基本ブロックの標準実行サイクルの総和にデータキャッシュミスのペナルティを加えることで求める．

計算区間に入ったら，counter をゼロクリアし，基本ブロックを通過するごとに，counter にその基本ブロックの標準実行サイクルを加算していく．このとき並行して，load/store 命令を通過するごとにキャッシュシミュレーションを行う．計算区間の終わりに到達したら，counter にその区間でのキャッシュミスによるペナルティを加算して，その区間の実行サイクル数とする．実際の環境では，両者はオーバラップするので本手法での評価値は，pessimistic なものとなる．しかしながら，この方法で，SPECfp92[4] ベンチマークの実行サイクル数を求め，実測値と比較したところ，すべてのベンチマークプログラムで誤差は 20%以内という結果を得ている[2]．このときに用いた CPU は

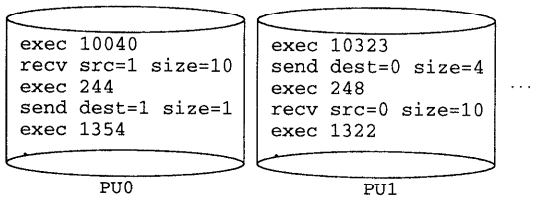


図 2 通信トレースファイル
Fig. 2 Message trace file.

MicroSparc-II である．誤差の要因としては，インストラクションキャッシュミス，TLB ミス，OS によるコンテクストスイッチにともなうキャッシュの汚れ，レジスタウインドウのオーバフロー等があげられるが，一番影響を与えたものは，扱うデータによって浮動小数点命令のレイテンシが大きく変化したということである．

3. ネットワークシミュレータ生成系 INSPIRE

INSPIRE[3] はネットワークシミュレータを生成するためのツールである．ネットワークの接続関係，動作を記述した NDF（Network Description File）を INSPIRE で処理することで，クロックレベルのネットワークシミュレータが生成される．INSPIRE を用いることで，様々な形態のネットワークシミュレーションを行える．ここでは，通信トレースファイルを用いたネットワークシミュレーションを行う．通信トレースファイルの一例を図 2 に示す．ファイルは，PU 台数分存在する．“recv” はブロックレシーブである．“exec” は，通信命令と通信命令の間に他の命令が何サイクル分消費したかを示している．2 章の方法で実行時に計算された結果である．ネットワークシミュレーション時に PU0 では，まず先頭 exec で 10040 サイクル進んだ後，recv 命令でメッセージの到達を待つ．PU1 からのメッセージが時刻 10353 に到着したとすると，次の exec で 244 サイクル消費され send 命令は時刻 10597 に開始される．なお，通信トレースファイルには，通信相手やタグ情報は記録するが，メッセージの内容は記録しない．

4. データ並列プログラムの性能予測手法

シミュレーションの対象を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プログラムに限定し，その性能予測手法について述べる．ここで扱う並列プログラムは負荷がほぼ均等に分散し，個々のノードプログラムの処理時間は，ほぼ同じであるものとする．また，通信データの内容によって通信データ量や転送先アドレス（受信バッファのアドレス），load/store のアドレスは変化しないものとする．

全体の構成は図 3 のようになる．基本的な処理の流れは，

- 並列プログラム（prog0~prog3）を後述する計算区間と通信区間に分ける．
- prog0~prog3 に EXCIT でインスツルメンテーションを施し，prog0” と prog0’~prog3’ を得る．prog” は計算区間の計測用のコード，prog’ はトレースファイル生成用のコードである．数字は PU 番号を表している．
- prog0” を実行することで計算区間の処理時間を求める．
- prog0’~prog3’，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区間の処理時間を得る．
- 計算区間と通信区間の処理時間から並列プログラムの計算時間と通信時間を得る．

というものである．

ここで，prog0’~prog3’ の実体は 1 つのプログラムである．内部の PU 番号を表す変数を順次変えて実行することで，それぞれの PU 番号に対応するトレースファイルを取得する．

4.1 データ並列プログラムの構造

データ並列プログラムの構造は図 4 のようになる．ここで，処理の内容に応じて，プログラムの各部分の名称を定義する．

- **計算区間**：計算のみ実行される部分（PU0 の太線）．この部分の処理時間を $Tcomp$ とする．
- **通信区間**：通信とその前後の計算を含んだ部分（PU0 の太線以外）．この部分の処理時間を $Tcomm$ とする．通信区間は，さらに，
  - **1 対 1 通信区間**：send/recv とその前後の計算部分．前後の計算とは，メッセージパッシングライブラリ関数内部のコピー処理等のことではなく，対象プログラムが実際に計算を行う部分である．たとえば，対象プログラムの通信を含むサブルーチンが 1 対 1 通信区間となる．この部分の処理時間 Tpp は，計算を行っている時間 Tpp_{comp} と通信を行っている時間 Tpp_{comm} にブレークダウンできる．
  - **大域通信区間**：barrier や gather 等の大域通信関数．この部分の処理時間 Tg は，1 対 1 通信区間と同様に計算部分 Tg_{comp} と通信

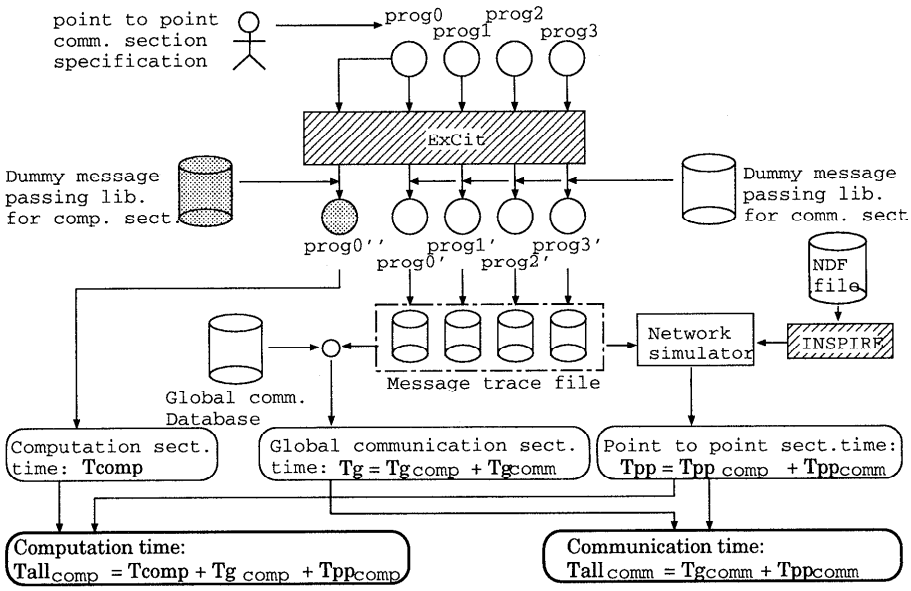


図 3 データ並列プログラムシミュレーションの流れ
Fig. 3 Overview of data parallel program simulation.

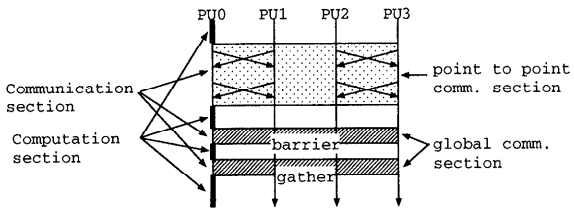


図 4 データ並列プログラムの構造
Fig. 4 Structure of data parallel program.

部分 Tg_{comm} にブレークダウンできる．ユーザが **send/recv** 等を用いて記述した大域通信関数は，1 対 1 通信区間に含める．

に分けられる．

1 対 1 通信区間の開始位置と終了位置は，ユーザがソースコードを見て，目印となるダミー関数を挿入することで指定する．プロセッサごとに通信パターンが異なる場合や，IF 文等の制御により，**send/recv** ポイントが異なるプログラムに関しては，これらの通信命令を内部に含むように 1 対 1 通信区間を設定する．

4.2 性能予測手法

ここでは 4.1 節で定義した各区間の処理時間を予測する方法について詳細に述べる．1 対 1 通信区間の指定以外の一連の処理は自動化されている．

4.2.1 計算区間の処理時間 $Tcomp$ の予測

各ノードプログラムの処理時間はほぼ同じであると仮定しているので，ここでは，PU0 の計算区間の処理時間を予測することで $Tcomp$ を求める．PU0 のノードプログラムに対して 2 章で示した方法で計算区間の処理時間を計測するようにインスツルメンテーションを施し，prog0" を作成する．prog0" を実行して処理時間 $Tcomp$ を求める．このとき，通信区間内でもキャッシュシミュレーションを行う．これにより，通信区間から抜け，再度計算区間に入ったときのキャッシュの状態を再現できる．実行に先立って，PU0 の通信区間内の通信命令は実際には何もしないダミーの命令に置き換えておく．ただし，**recv** 命令に関しては，キャッシュシミュレータを call して受信バッファ部分のメモリをキャッシュからクリアするようにする．これは，受信命令によってキャッシュが purge されるからである．

通信命令をダミーの命令に置き換えることで，PU0 のノードプログラムは，途中で停止することなく最後まで実行される．対象プログラムは，受信されたデータの内容よって処理フローが変化しないので，並列プログラムとして動作したときと同様の処理時間が得られる．

今回の仮定では，対象プログラムは PU0 のみが固有の処理を行わない，あるいは，行っても，その部分の処理時間は，全体に比べて無視できるものだと仮定している．本手法を PU0 のみ固有の処理を行うプログラムに適用することもできるが，その場合，PU0 固有の処理が存在する部分を含む形で 1 対 1 通信区間を設定することになる．

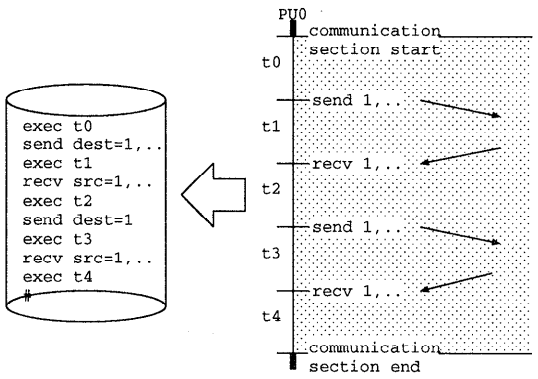


図 5 1 対 1 通信区間と通信トレースファイル
Fig. 5 Point to point communication section and its trace file.

4.2.2 通信区間の処理時間 $Tcomm$ の予測

- **1 対 1 通信区間の処理時間 Tpp 予測**
  プログラムを EXCIT で変換し，通信命令間の処理時間を計算するようにする．また，通信命令を，その地点までの処理時間を出力し，続いて，自分の通信処理の内容を出力する関数に置き換える（prog0'~prog3' が生成される）．出力される内容は，オペレーションの内容（MPI 関数の名前），相手先の PU 番号，メッセージサイズ，タグである．変換後のプログラムを実行することで，通信トレースファイルが得られる．実際の通信処理は行われない．**図 5** は，図 4 の PU0 のフローを抜きだしたものである．この例では，1 対 1 通信区間に入ると，プログラムは自分自身の処理時間を計算し始め，最初の `send` 命令で処理時間 t0 をファイルに出力し（`exec t0`），`send` 命令の内容（`send dest=1,...`）を出力する．続く `recv` 命令では，`exec t1` を出力し，`recv src=1,...` を出力する．同様の処理を続けることで，左側の通信トレースファイルが得られる．ファイルの最後の#は，1 対 1 通信区間の終了を示している．計算区間の場合と同様に，メッセージ通信を実際に行わなくてもプログラムは正しいフローで実行される．実際の通信を行っていないので，オーバフローやゼロ割りが発生する可能性がある．この場合，ハンドラを用意し処理を継続させる必要がある．ハンドラ内の処理は，性能予測上は無視されるので，オーバフローやゼロ割りの有無にかかわらず，予測される結果は同一である．

  1 つのノードプログラム内に，通信区間は何回も出現する．出現するごとに，通信トレースファイルに同様の内容を追加していく．この処理をすべての PU に施すことで，PU 数分の通信トレースファイルを生成できる．ここで得られた PU 台数分の通信トレースファイルをネットワークシミュレータにかけることで，1 対 1 通信区間の処理時間 Tpp が得られる．通信命令を前後の計算部分まで含めてシミュレーションするため，通信と計算のオーバラップを考慮することができる．
- **大域通信区間の処理時間の予測**
  個々の大域通信の処理時間 Tg は，処理の内容と PU 台数，ネットワーク性能が決まれば，一意に決めることができる．これらは，あらかじめ計算した結果を表にしておく．プログラム実行時に大域通信の種類ごとに実行された回数をカウントし，後から表を用いて処理時間を集計することで処理時間を算出する．

4.2.3 プログラム全体の計算時間，通信時間の算出

Tpp は，計算部分 Tpp_{comp} と通信部分 Tpp_{comm} へとブレークダウンできる．Tpp_{comp} は，各通信トレースファイルの `exec` エントリの時間を PU ごとに合計し，最大値をとることで求める．Tpp_{comm} は，Tpp から Tpp_{comp} を引くことで求める．すなわち，本論文では通信時間を，通信がないときのプログラムの処理時間に対する，通信によって生じたプログラムの処理時間の増分と定義する．大域通信区間に関しては，その内部の計算時間と通信時間の内訳をデータベースに登録しておく．これを個々に集計することで大域通信内の計算時間 Tg_{comp} と通信時間 Tg_{comm} を求める．最後に，並列プログラムの計算時間 $Tall_{comp}$ と並列プログラムの通信時間 $Tall_{comm}$ は，

$$Tall_{comp} = Tcomp + Tpp_{comp} + Tg_{comp}$$
$$Tall_{comm} = Tpp_{comm} + Tg_{comm}$$

から求める．

5. NAS Parallel Benchmark Program の性能予測

5.1 NAS Parallel Benchmark Program ver. 2.3

今回，評価に利用した NPB ver. 2.3 は，MPI を使って FORTRAN または C で書かれたコードである．本節では，Kernel 問題から MG，Application 問題から LU を選び，キャッシュサイズ，ネットワークのトポロジ，通信オーバヘッド（以降 μ_0 と記す），バンド幅（b/w と記す）を変化させ，PU 台数を変えた場合の処理時間の予測を行った．問題サイズは，Class B である．

表 1 CPU モデル
Table 1 CPU model.

pipeline	2 int pipe, 1 fp pipe
clock	100 MHz
L1-cache size /miss penalty	8, 16, 64 KB/5 cycle (logical, 4-way set associative)
L2-cache size /miss penalty	1 MB/20 cycle (logical, direct map)

表 2 性能予測に用いたハードウェアパラメータ
Table 2 Hardware parameters for performance evaluation.

ケース	cache (KB)	network topology	μ_0 (μs)	b/w (MB/s)
1	8, 16, 64	HXB	0	∞
2	16	C, 2d mesh, HXB, ring	0	100
3	16	HXB	0	10, 100, ∞
4	16	HXB	0, 10, 250	∞

C：完全結合，HXB：Hyper Crossbar Network

5.2 性能予測のための準備

前処理として，プログラムを 1 カ所書き換える．NPB 2.3 のプログラムでは，データサイズ等を表す変数をプログラムの先頭で PU0 が他の PU に broadcast している．この部分を，PU0 は何もしないように，他の PU では，必要な値が代入されるように書き換える．実際は，MG，LU ともに 2 行コメントアウトしただけである．続いて，1 対 1 通信区間の開始位置と終了位置を示すフック関数の組をプログラムに入れる．MG，LU とも 2 組挿入した．なお，LU に関しては，もう 1 行だけコメントアウトしている．

性能予測実験にあたって，**表 1** の条件のモデルの CPU を用いた．PU 台数は，MG は 16 台から 1024 台まで LU は 16 台から 256 台までそれぞれ変化させた．LU はプログラムおよびデータサイズの制約で Class B では，256 台が PU 台数の上限となる．NPB のプログラムでは，最外ループで同一処理を繰り返す構造をとっている．シミュレーション高速化のため，また，トレースファイルサイズ削減のために 1 回分の最外ループのみシミュレーションし，ループ回数倍して全体の処理時間としている．

MG，LU とも**表 2** に示す 4 通りのケースに関して性能予測を行っている．ケース 1 では，キャッシュサイズと並列プログラムの計算時間の関係を求めている．ケース 2 は，ネットワークトポロジと通信時間の関係を求めている．ここで，HXB（Hyper Crossbar Network）[6] と二次元メッシュは，ネットワークがそれぞれ立方体と正方形に最も近い形の構造をとるようにしている．ケース 3 は，b/w と通信時間の関係，ケース 4 は，μ_0 と通信時間の関係を求めている．μ_0 は，プログラム中でメッセージ通信関数がコールされてから，ネットワークインタフェースから実際にメッセージがネットワークに出ていくまでの時間である．通信関数内でのバッファコピーやネットワークインタフェース内で消費される時間であり，メッセージ長等によって実際には変化するが，本シミュレーションでは定数として与える．通信モデルとしてよく用いられる，

$$T = overhead + datasize/bandwidth$$

の overhead に相当するものである．T は通信時間である．この式にあてはめて考えた場合，本シミュレーションでは，$datasize/bandwidth$ の項をネットワークシミュレーションで求め，T を算出していることになる．用いた値の目安としては，Ethernet はバンド幅が 10 MB/sec 程度，この上での TCP/IP の μ_0 が 250 μsec 程度である．Gigabit LAN 上にユーザレベル通信をインプリメントして，10 μsec を切る μ_0 を実現した例[7] も報告されている．また，MPP は 100 MB/sec 程度のバンド幅を持つものが少なくない．

表 3 MG：メッセージサイズ
Table 3 MG: Message size.

PU 台数	16	64	256	512	1024
1 PU の平均通信回数	2666	2616	2431	2304	2105
平均メッセージ長（Byte）	16402	6382	4633	1900	1419
総メッセージ量（GByte）	0.699	1.06	1.44	2.24	3.06

5.3 MG kernel

MG は，三次元ポアソン方程式をマルチグリッド法を用いて解くプログラムである．Class B では，問題サイズは 256×256×256 で，ワーキングセットの大きさは，457 MByte である．1024 PU の場合 1 PU あたり 772 KByte である．通信パターンは，階層的な隣接通信であり，MG で用いられる階層的なグリッド上で隣接した PU どうしが通信を行う．**表 3** に，通信トレースファイルから求めた，PU 台数を変化させた場合の 1 PU あたりの通信回数および平均メッセージ長，全体での総メッセージ量を示す．

次に，キャッシュサイズを変えたときの PU 台数と計算時間の関係を**図 6** に示す．キャッシュサイズによる処理時間の変化は，ほとんど見られなかった．PU 台数が少ない場合，1 台あたりのワーキングセットが大きくなるのでキャッシュのミス率が高くなるが，16

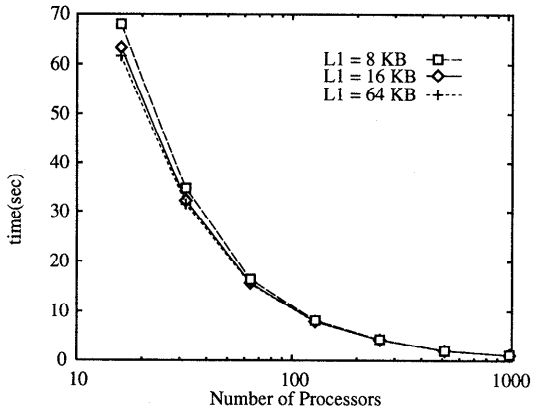


図 6 MG：キャッシュサイズと計算時間の関係（HXB, b/w=100 MB/sec, μ_0=0）
Fig. 6 MG: Cache size effect on computation time (HXB, b/w=100 MB/sec, μ_0=0).

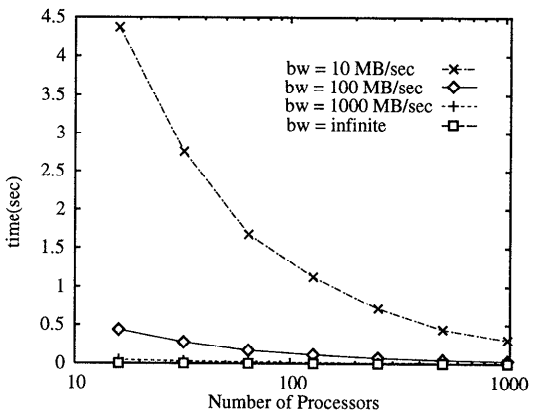


図 8 MG：バンド幅と通信時間の関係（L1cache=16 K, HXB, μ_0=0）
Fig. 8 MG: Bandwidth vs. communication time (L1cache=16 K, HXB, μ_0=0).

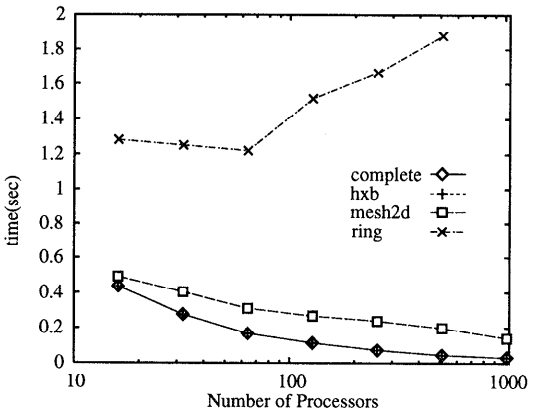


図 7 MG：トポロジと通信時間の関係（L1cache=16 K, b/w=100 MB/sec, μ_0=0）
Fig. 7 MG: Topology vs. communication time (L1cache=16 K, b/w=100 MB/sec, μ_0=0).

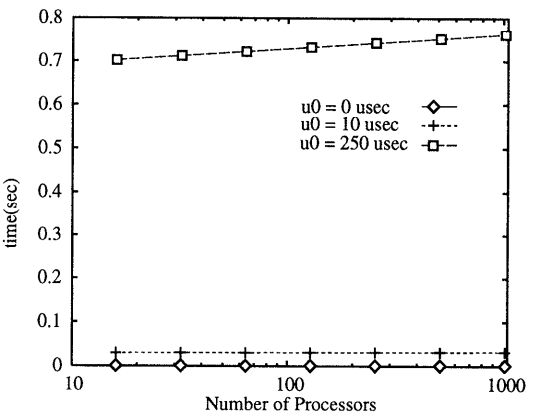


図 9 MG：μ_0 と通信時間の関係（L1cache=16 K, HXB, b/w=infinite）
Fig. 9 MG: μ_0 vs. communication time (L1cache=16 K, HXB, b/w = infinite).

台で 8 KByte の一次キャッシュでも，シミュレーション結果からヒット率は 90%以上であることが分かった．MG は，使用データのローカリティが高いと考えられる．キャッシュサイズをさらに減らせば，ある時点からミス率が急激に上がる可能性はあるが，現在の CPU は通常 8 K 程度以上の一次キャッシュを持っているので，あまり小さいキャッシュサイズの場合を議論しても意味がない．以降，ケース 1 からケース 4 の実験では，一次キャッシュのサイズを 16 Kbyte に固定している．

図 7 にトポロジを変化させたときの PU 台数と通信時間の関係を示す．完全結合網と HXB は，同程度の性能を示した．これに比べると二次元メッシュではオーバヘッドが見え始め，リングに至っては通信性能が著しく低下している．**図 8** に μ_0= 0 とし，バンド幅を変化させたときの PU 台数と通信時間の関係を示す．PU 台数が小さい場合に値が大きい．これは，PU 台数が小さい場合は，PU 間通信のデータサイズが大きくなり，バンド幅が小さいと処理時間がかかるためだと予想される．しかしながら，各 PU 台数における通信時間は計算時間から比べると 7%以下なので，プロセッサ数が千台規模以下の場合は，MG におけるバンド幅の影響はさほどないといえる．**図 9** にバンド幅を無限大とし，μ_0 を変化させたときの PU 台数と通信時間の関係を示す．μ_0 が大きい場合，影響は PU 台数にかかわらず現れる．この影響と並列プログラムの効率の関係を**図 10** に示した．16 PU を基準に計算している．具体的には，16 PU のときの処理時間に 16

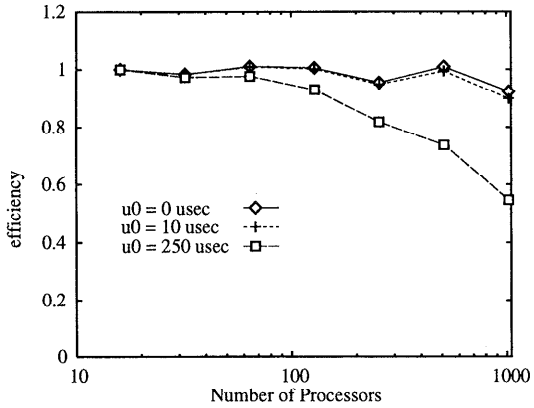


図 10 MG：μ_0 と並列化効率の関係（L1cache=16 K, HXB, b/w=infinite）
Fig. 10 MG: μ_0 vs. efficiency (L1cache=16 K, HXB, b/w=infinite).

表 4 LU：メッセージサイズ
Table 4 LU: message size.

PU 台数	16	32	64	128	256
1 PU の平均通信回数	75750	82062	88375	91531	94678
平均メッセージ長（Byte）	3050	2346	1525	1157	762
総メッセージ量（GByte）	3.69	6.16	8.62	13.5	18.4

を掛けたものを各 PU 数における処理時間に PU 数を掛けたもので割った値である．μ_0 を 250 μsec とすると 1024 PU の場合，並列化効率は 55%程度にまで下がる．

5.4 LU application

LU は，SSOR（Symmetric Successive Over-Relaxation）法を利用した Navier-Stokes 方程式のソルバである．Class B では，問題サイズは 102×102×102 で，ワーキングセットの大きさは，192 MByte である．256 PU の場合 1 PU あたり 2.06 MByte となる．通信パターンは隣接通信である．**表 4** に，PU 台数を変化させた場合の，1 PU あたりの通信回数および平均メッセージ長，全体での総メッセージ量を示す．

次に，キャッシュサイズを変化させたときの PU 台数と計算時間の関係を**図 11** に示す．MG の場合と同様にキャッシュサイズによる処理時間の変化は，ほとんど見られなかった．**図 12** にトポロジを変化させたときの PU 台数と通信時間の関係を示す．完全結合網（complete）と HXB は，ほぼ同程度の性能を示した．また，MG と異なって二次元メッシュも高い性能を示している．これは，MG は階層的な隣接通信であるのに対し，LU は純粋な隣接通信で，隣接する PE 間の

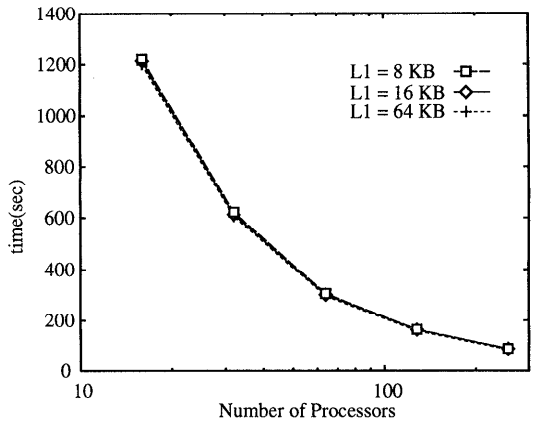


図 11 LU：キャッシュサイズと計算時間の関係（IIXB, b/w=100 MB/sec, μ_0=0）
Fig. 11 LU: Cache size vs. communication time (HXB, b/w=100 MB/sec, μ_0=0).

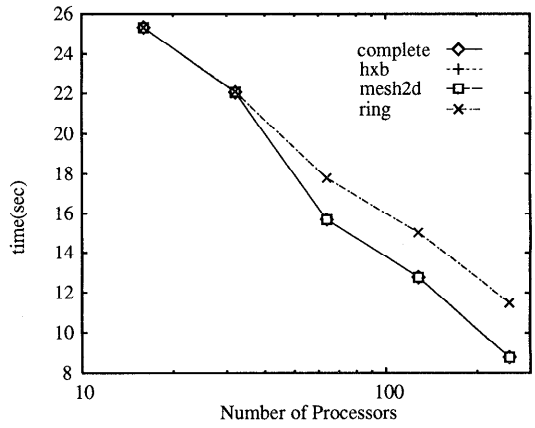


図 12 LU：トポロジと通信時間の関係（L1cache=16 K, b/w=100 MB/sec, μ_0=0 μsec）
Fig. 12 LU: Topology vs. communication time (L1cache=16 K, b/w=100 MB/sec, μ_0=0 μsec).

通信が多いためだと考えられる．リングは，64 台から通信オーバヘッドが見え始める．**図 13** に μ_0=0 とし，バンド幅を変化させたときの PU 台数と通信時間の関係を示す．全体的な傾向は MG と同様である．**図 14** にバンド幅を無限大としたときの μ_0 と通信時間の関係を示す．μ_0 の影響は，MG と同様に大きい．μ_0 と並列化効率の関係を**図 15** に示す．64 台を過ぎると，並列化効率が下がり始める．256 PU，μ_0= 250 μsec のとき，並列化効率は 60%程度まで下がる．

6. 考　　察

6.1 シミュレーション手法の高性能化

6.1.1 高　速　化

我々の手法では，高速化のためにデータパラレルプ

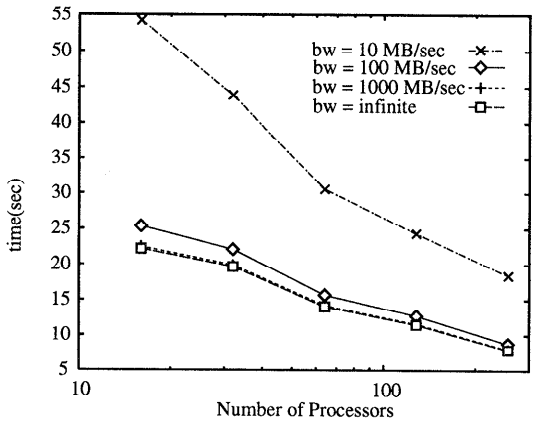


図 13 LU：バンド幅と通信時間の関係（L1cache=16 K, HXB, μ_0=0）
Fig. 13 LU: Bandwidth vs. communication time (L1cache=16 K, HXB, μ_0=0).

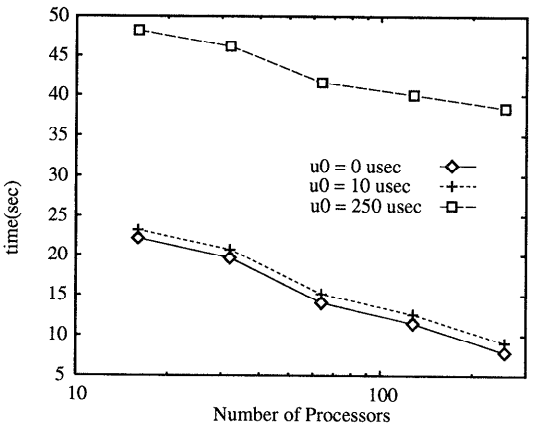


図 14 LU：μ_0 と通信時間の関係（L1cache=16 K, HXB, b/w=infinite）
Fig. 14 LU: μ_0 vs. communication time (L1cache=16 K, HXB, b/w=infinite).

ログラムを計算区間と通信区間に分けている．この場合，プログラム全体に対してインスツルメントを行ったコード（prog0" に相当）を 1 回，通信区間のみにインスツルメンテーションを行ったコード（prog0'～prog3' に相当）を PU 台数分走らせればよい．

プログラムを区間に分割せず，全体を通信区間と見なして通信トレースファイルを得ることもできる．この場合，コード全体にインスツルメンテーションをかけたコードを PU 台数分走らせなければならない．コード全体にインスツルメントをかけたプログラムの実行時間は，オリジナルコードに比べて数十倍程度であり，一方，通信区間のみにインスツルメンテーションをかけたコードの実行時間は，コードにもよるが数倍程度である．誤差と処理速度のトレードオフとなるが，データ並列プログラムの場合，計算部分の処理時間の PU ごとのばらつきはさほどないと考えられるので，プログラムを区間に分割する方法が有利であると考えている．

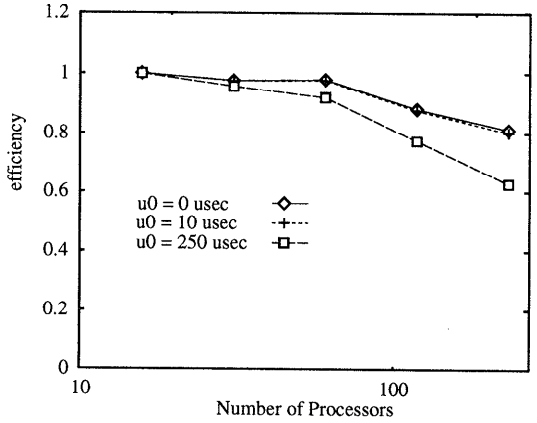


図 15 LU：μ_0 と並列化効率の関係（L1cache=16 K, HXB, b/w=infinite）
Fig. 15 LU: μ_0 vs. communication time (L1cache=16 K, HXB, b/w=infinite).

前章の NPB のシミュレーションに要した時間は以下のとおりである．通信トレースファイルの生成は，いずれの PU 台数の場合にも MG の場合 20 分程度，LU の場合は 120 分程度であった．用いた計算機は，SUN Ultra-I 200 MHz 1 台である．MG と LU の時間の差は 1 対 1 通信区間の差，すなわち，インスツルメントされたコードの走る割合によるものである．インスツルメンテーションに要する時間は，全体と比べると無視できるほど小さなものである．

ネットワークシミュレーションは，台数が多くなるほど時間を要するようになり，また，ターゲットプログラムのメッセージがネットワーク中で衝突するようになり始めると極端に処理時間が延びる．ネットワークシミュレーションにも SUN Ultra-I 200 MHz を 1 台用いたが，MG の 1024 台で，二次元メッシュの場合 150 分程度，ring の場合は 600 分かけても処理が終了しなかった．LU の 256 台の場合は，二次元メッシュだと 30 分程度，ring だと 150 分程度の処理時間であった．

通信トレースファイルの作成は 1 度行えばよく，また，プロセッサ独立に行えるので処理時間はさほど問題とならない．一方，ネットワークシミュレーションは，パラメータを変更しながらデータをとる必要があるため，処理時間が問題となる．したがって，シミュレータの高速化を検討する必要がある．

今回の実験では，トレースファイルの容量は問題

とならなかった．トレースファイルの容量は，MG，LU とも PU 数が最大のときに最大で，MG の場合は 1024 PU で約 15 MByte，LU の場合は 256 PU で約 7.5 MByte であった．通信が大量に発生しトレースファイルの容量が問題となる場合は，ファイルの圧縮や，対象プログラムの分割といった手法を考えていく必要がある．

6.1.2 高精度化

現在，仮定しているネットワークのモデルは，`mpi_send()` のような通信関数が実行されると，μ_0 のオーバヘッドをともなってネットワークに出ていくというモデルである．しかしながら，メッセージは，実際には通信関数の中でバッファ管理やデータコピーといった処理が行われてからネットワークインタフェースを経由してネットワークに出ることになる．また，OS 呼び出しのオーバヘッドも付加され，プロトコルによっては，事前に PU 間でネゴシエーションが行われる場合もある．今回のシミュレーションでは，このようなオーバヘッドを一括して通信オーバヘッド（μ_0）としたため，この部分で実機との誤差が生じる可能性がある．オーバヘッドのモデリングおよび，シミュレーション結果の精度向上に関しては今後の課題とするが，通信関数内部のコードまで含めてインスツルメントすることで，シミュレーション結果を実機に近づけられるのではないかと考えている．

6.2 シミュレーション結果の利用法

本シミュレーション結果から様々な情報を得ることができる．たとえば，問題を固定した際の CPU 性能に対するネットワーク性能のバランスポイントを見つけることができる．**図 16** は，図 15 のデータに対して，並列化効率 80%に線を引いたものである．この図はバンド幅が無限大の場合を示しているので，並列化効率 80%を実現するためには，256 台の場合は μ_0 を 10 μsec 程度にしなければならないことが分かる．また，通信時間は μ_0 に比例するという前提で計算すると，128 台の場合は μ_0 をおよそ 188.7 μsec 以下に抑えなければ並列化効率 80%は実現できないことが分かる．

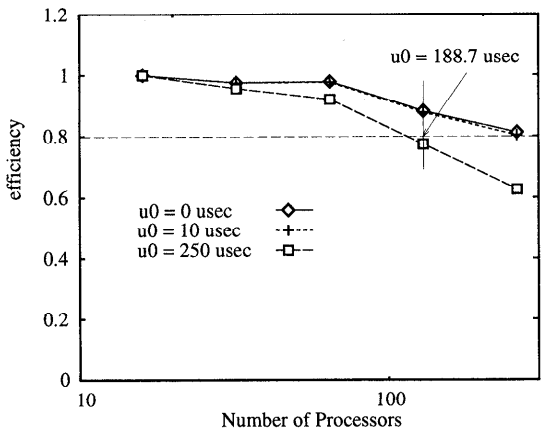


図 16 LU で並列化効率 80%実現時の CPU 性能とネットワーク性能のバランスポイント（L1cache=16 K, HXB, b/w=infinite）

Fig. 16 The balance point of CPU performance and Network performance when efficiency is 80% (L1cache=16 K, HXB, b/w=infinite).

7. 関連研究

Fahringer ら[5] の提案するシステム（PPPT）では，逐次プログラムに FORTRAN 言語レベルでインツルメントを行い，分岐の割合，ループの回数，ステートメントの実行回数を収集する．これらの情報と，プログラムから見積もった大雑把なネットワークの衝突率，キャッシュのヒット率から，プログラムを並列化した場合の処理時間を見積もっている．このほかにも，大規模な並列プログラムの挙動を予測する場合，アルゴリズムやプログラムを机上で解析する方法[8),9)] が多く試みられている．これらの手法では，全般的な傾向を予測することはできるが，実プログラムレベルでの挙動を定量的に予測することは難しい．なぜなら，実プログラムレベルでは，

- プログラムのインプリメント
- コンパイラによる最適化の影響
- 実行時のネットワーク資源の競合

といった影響が生じるからである．我々の方法では，ユーザの実プログラムで挙動を予測する．また，通信部分に関してはクロックレベルのシミュレーションを行っているので，計算と通信のオーバラップを考慮できる．

数プロセッサ程度の小規模な問題に対して，クロックレベルでネットワークまで含めたシステム全体のシミュレーションを行った例がある[10]．トレースをとってネットワークシミュレーションを行うという基本的な考え方は本手法と同様である．しかしながら，処理速度やシミュレーション環境の問題から大規模な問題に適用するのは難しい．我々の方法は，通信トレースファイルを利用して，通信区間のみをシミュレーションするので高速である．また，上記の方法では，トレースファイル取得に際し，並列プログラム全体を実行しなければならないため，適用できる PU 数が制限されるが，我々の方法では，通信トレースファイル作成処理は PU ごとに行えるので，PU 数が大規模になった

場合も対応可能である．ただし，ネットワークシミュレーションの方で扱えるPU数が制限される可能性はある．

千台規模のシステムの予測に関しては，1台分のプログラムの挙動から全体の挙動を予測した例がある[11]．この報告では，ネットワークは無衝突という前提で，メッセージサイズから通信命令の処理時間を決めている．計算部分については，コード全体のインストラクションをシミュレーションしているので精度は高い．我々の手法は，ネットワークは衝突があるという前提でネットワークシミュレーションを行っている．計算部分は，インスツルメンテーションを用いて実行時に実行時間を計算するので精度は劣るが高速である．

8. まとめと今後の課題

本論文では，大規模データ並列プログラムの性能予測手法について述べ，これを用いて，NAS Parallel Benchamrk ver. 2.3 の MG と LU の性能評価を行った．NPB Class B 問題という大きな問題について，千台規模まで解析的なモデリングを用いずに，ネットワークまで含めてシミュレーションしたのは本手法が初めてである．実験結果から，MG，LU ともキャッシュサイズは問題とならず，ネットワーク性能，特にPU数が大規模な場合，通信オーバヘッド μ_0 の影響が大であるという評価結果を得た．

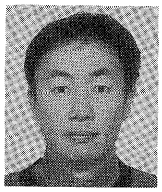
本論文で述べた手法は，シミュレーションに先だって人手でプログラムに若干の変更を加えている．これがどこまで自動化できるかについて検討する必要がある．また，各PUの負荷の偏りの調査や，本シミュレーション手法の精度の検証，高速化について取り組む予定である．さらに，通信トレースファイルをネットワークシミュレーションする手法とは別に，インスツルメントされたプログラムをオンラインで直接ネットワークシミュレータに接続する方法についても検討している．

本手法では，ハードウェアパラメータを変化させた際のプログラムの挙動を定量的に評価できる．本手法によって得られた結果は，アプリケーションユーザが並列計算機を選択する際の有効な指針を与えることができる．

参 考 文 献

1) Bailey, D., Harris, T., Saphir, W., Van der Wijngaart, R., Woo, A. and Yarrow, M.: The NAS Parallel Benchmarks 2.0, *NASA Ames Research Center Report*, NAS-05-020 (1995).
2) Kubota, K., Itakura, K., Sato, M. and Boku, T.: Accuracy of Fast Performance Prediction by instrumentation tool EXCIT, *Proc. HPC Asia'98*, pp.1031–1038 (1998).
3) Boku, T., Harada, T., Sone, T., Nakamura, H. and Nakazawa, K.: INSPIRE: A general purpose network simulator generating system for massively parallel processors, *Proc. PERMEAN'95*, pp.24–33 (1995).
4) http://www.spec.org
5) Fahringer, T. and Zima, H.: A static parameter based performance prediction tool for parallel programs, *Proc. 1993 ACM Int. Conf. Supercomputing*, pp.207–219 (1993).
6) 朴 泰祐，板倉憲一，曾根 猛，三島正博，中澤喜三郎，中村 宏:ハイパクロスバ・ネットワークにおける転送性能向上のための手法とその評価，情報処理学会論文誌，Vol.36, No.7, pp.1610–1618 (1996).
7) Tezuka, H., Hori, A., Ishikawa, Y. and Sato, M.: PM: An Operating System Coordinated High Performance Communication Library, *High Performance Computing and Networking Europe '97*, pp.708–717 (1997).
8) Yarrow, M. and Van der Wijngaart, R.: Communication Improvement for the LU NAS Parallel Benchmark: A Model for Efficient Parallel Relaxation Schemes, *NAS Technical Report*, NAS-97-032 (1997).
9) Rothberg, E., Singh, J.P. and Guputa, A.: Working Sets, Cache Sizes, and Node Granularity Issues for Large-Scale Multiprocessors, *Proc. ISCA'93*, pp.14–25 (1993).
10) Boku, T., Mishima, M., Itakura, K., Nakamura, H. and Nakazawa, K.: VIPPES: A performance preevaluation system for parallel processors, *High Performance Computing and Networking Europe'96* (1996).
11) Itakura, K., Hattori, M., Boku, T., Nakamura, H. and Nakazawa, K.: Preliminary Evaluation of NAS Parallel Benchmarks on CP-PACS, *Proc. PERMEAN'95*, pp.68–77 (1995).

(平成10年9月2日受付)
(平成11年3月5日採録)

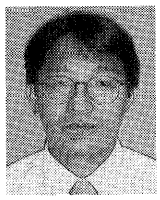


久保田和人（正会員）

昭和39年生．昭和63年早稲田大学理工学部電子通信学科卒業．平成5年同大学大学院理工学研究科博士課程修了．同年（株）東芝入社．平成7年10月より平成10年9月まで技術研究組合新情報処理開発機構に出向．工学博士．並列計算機のプログラミング環境，性能評価環境，計算機クラスタシステムに興味を持つ．

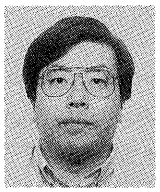
板倉　憲一（正会員）

昭和44年生．平成5年筑波大学第三学群情報学類卒業．平成11年同大学大学院工学研究科博士課程修了．平成11年筑波大学計算物理学研究センター・リサーチ・アソシエイト．工学博士．並列計算機アーキテクチャ，並列入出力・可視化機構の研究に従事．



佐藤　三久（正会員）

昭和34年生．昭和57年東京大学理学部情報科学科卒業．昭和61年同大学大学院理学系研究科博士課程中退．同年新技術事業団後藤磁束量子情報プロジェクトに参加．平成3年，通産省電子技術総合研究所入所．平成8年より，技術研究組合新情報処理開発機構つくば研究センターに出向．現在，同機構並列分散システムパフォーマンス研究室室長．理学博士．並列処理アーキテクチャ，言語およびコンパイラ，計算機性能評価技術等の研究に従事．日本応用数理学会会員．



朴　　泰祐（正会員）

昭和59年慶應義塾大学工学部電気工学科卒業．平成2年同大学大学院理工学研究科電気工学専攻後期博士課程修了．工学博士．昭和63年慶應義塾大学理工学部物理学科助手．平成4年筑波大学電子・情報工学系講師，平成7年同助教授，現在に至る．超並列処理ネットワーク，超並列計算機アーキテクチャ，ハイパフォーマンスコンピューティング，並列処理システム性能評価の研究に従事．電子情報通信学会，IEEE各会員．