計算機アーキテクチャ 95−10
(1992. 8. 19)

メッセージレベルシミュレータ

堀江 健志　I. Chuang　井川 英子　林 憲一
(株) 富士通研究所
{*lions, monkey, eiko, woods*}*@flab.fujitsu.co.jp*

メッセージパッシング型並列計算機のアーキテクチャを評価する『メッセージレベルシミュレータ』について述べる. メッセージパッシングで記述されたプログラムの実行は, CPU の性能とメッセージ通信のオーバヘッドに依存する. メッセージレベルシミュレータは, 演算性能と通信性能をパラメータとして与えることにより, 並列プログラムを効率良く実行するためのアーキテクチャの問題点を明確にすることができる. 本論文では, メッセージレベルシミュレータの機能と実現方法について述べる. さらに, 実際のアプリケーションにメッセージレベルシミュレータを適用し, アプリケーションの統計情報を得るとともに, 演算性能と通信性能のそれぞれがアプリケーション全体におよぼす影響について検討する.

Message Level Simulator

Takeshi Horie　Isaac Chuang　Hideko Ikawa　Kenichi Hayashi
Fujitsu Laboratories Ltd.
1015 Kamikodanaka, Nakaharaku
Kawasaki 211 Japan
{*lions, monkey, eiko, woods*}*@flab.fujitsu.co.jp*

This paper presents *MLSim*, a message level simulator which we have developed to simulate and evaluate message-passing architectures. The performance of message-passing applications may depend critically on cpu speed, communication throughput and latency, and message handling overhead. *MLSim* can be used to rapidly investigate the effect of varying such parameters on the execution efficiency of a wide range of parallel programs. In this paper, the implementation and features of *MLSim* are described, and results from analyzing eleven real applications with *MLSim* are discussed.

1 はじめに

並列化されたアプリケーションプログラムを並列計算機で実行し高い性能を得るためには，アーキテクチャと並列化プログラム(アルゴリズム)の問題点を明らかにし，それらを改良していく必要がある．そのためには，開発したアプリケーションプログラムを実際の並列計算機で実行したとき，アーキテクチャとアプリケーションの両面から評価を行なうとともに，問題となるアーキテクチャを改良したときに得られる性能向上を定量的に評価し，次世代アーキテクチャに反映していく必要がある．

我々は，アーキテクチャの評価を目的とした『メッセージレベルシミュレータ』を開発した．メッセージレベルシミュレータは，メッセージパッシングで記述されたプログラムをシミュレーション実行し評価するツールである．メッセージレベルシミュレータは以下の機能を持つ．

- シミュレーション(入力パラメータ)
  - 演算性能
  - 通信性能
    - ネットワーク(レイテンシ，スループット，輻輳，トポロジ)
    - メッセージハンドリング(送信，受信，通信のオーバラップ)
- 性能評価
  - オーバヘッド解析(アイドル，通信)
  - 通信の統計情報
    - メッセージサイズ
    - 通信距離
    - 通信頻度
    - グラフィック表示

本論文では，メッセージレベルシミュレータとシミュレータを用いた解析結果について述べる．メッセージレベルシミュレータは，シミュレーション実行部(メッセージレベルシミュレータ)と性能解析部(パフォーマンスアナライザ)から構成される．まず，第2章でメッセージレベルシミュレータの構成と対象としているプログラムモデルについて述べる．第3章で，シミュレータ実行後，性能解析を行なうパフォーマンスアナライザについて述べる．最後に，いくつかのアプリケーションを対象としたメッセージレベルシミュレータによる解析例を示す．

2 メッセージレベルシミュレータ

2.1 メッセージレベルシミュレータの構成

2.1.1 実行方法

メッセージレベルシミュレータは，並列計算機AP1000 [1]上に実現されている．まず，アプリケーションをAP1000で実行させる．このとき，トレース情報を各プロセッサごとに格納する．トレース情報は，ホストプロセッサに格納せずそのままプロセッサに格納しておくことにより，プロセッサ台数に比例して大きくなる可能性がある巨大なトレース情報をシミュレータで扱うことが可能にる．メッセージレベルシミュレータは，このトレース情報を書き換えながらメッセージレベルのシミュレーションを実行する．図1に実行方法を示す．シミュレータでは，通信のオーバヘッド，演算性能などのパラメータを変え，何度でも同じトレース情報を利用してシミュレーションすることが可能であり，短時間に様々なアーキテクチャの評価を行なうことができる．

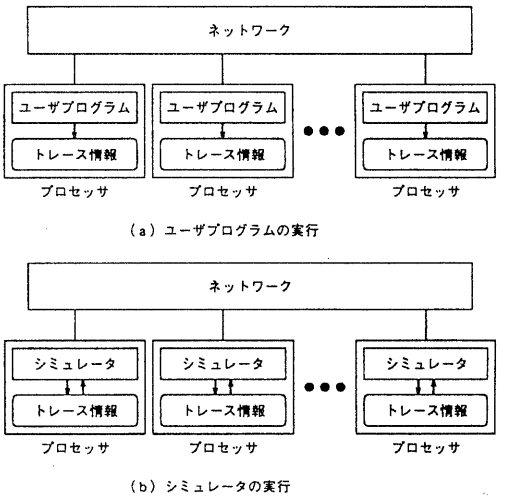


図 1: シミュレータの実行方法

2.1.2 トレース情報

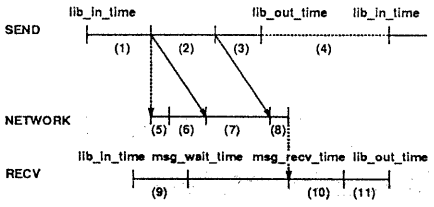
トレース情報のイベントは，大きくわけて二種類ある．第一のイベントは，通信，同期などの並列処理に必要なライブラリの開始と終了，割り込みの開始と終了，および，タスクスイッチである．それぞれのイベントの種類とそのときの時刻をトレース情報としてとる．

第二のイベントは，メッセージの送信あるいは受信などである．トレース情報として，送信したメッセージの宛先(プロセッサ番号，タスク番号)，メッセージのタイプ，メッセージのサイズ，メッセージの番号と送信時刻，受信したメッセージの送信元(プロセッサ番号，タスク番号)，メッセージのタイプ，メッセージのサイズと受信時刻，バリア同期の種類と同期発行時刻，メッセージ待ちになった時刻をとる．メッセージ番号は，送信したメッセージすべてに各プロセッサごとに付けられる逐次番号である．

2.2 モデルとシミュレーション

メッセージレベルシミュレータが対象とするメッセージパッシングで記述されたプログラムのモデルについて述べる．図2に送信と受信のモデルを示す．*send_prolog_time*などがパラメータとして与えられる．遅延モデルには，送信，受信，バリア同期についてそれぞれ基本的に以下の遅延時間が含まれている．

- 送信 … 送信時間(メッセージサイズ×転送単位時間) + 処理オーバヘッド
- 受信(コピーなし) … メッセージ受信待ち時間 + 処理オーバヘッド
- 受信(コピーあり) … メッセージ受信待ち時間 + 処理オーバヘッド + コピー(メッセージサイズ×転送単位時間)
- メッセージ読み込み … メッセージ読み込み初期化時間 + コピー(メッセージサイズ×転送単位時間)
- バリア同期 … 最も遅く同期要求したプロセッサの要求時間までの待ち時間 + 同期ディレイ + 処理オーバヘッド



(1) *send_prolog_time*
(2) *send_msg_time* × msg_size
(3) *send_epilog_time*
(4) *computation_time_factor* × computation_time
(5) *network_prolog_time*
(6) *network_delay_time* × distance
(7) *network_msg_time* × msg_size
(8) *network_epilog_time*
(9) *recv_prolog_time*
(10) *recv_msg_time* × msg_size
(11) *recv_epilog_time*

図 2: 送信と受信の遅延モデル

これらの並列処理ライブラリにかかる時間をパラメータとして与え,送信と受信, 同期におけるプロセッサ間の時間関係を保つようにシミュレーションし,全体の実行時間を求めることができる.また, アーキテクチャとして新たな機能を付加することもできる.例えば,メッセージハンドリングのオーバヘッドの削減のために,送信と演算をオーバラップさせたときの効果もシミュレーションにより求めることが可能である.

シミュレーションは, トレース情報をもとに各プロセッサでメッセージ通信とバリア同期に必要な時間を計算しながら実行する.メッセージの送信イベントに対しては, 実際にメッセージを送信する.このメッセージには, 送信時刻とメッセージサイズが記されている.メッセージ受信イベントに対しては, 送信プロセッサから該当するメッセージを受信する.受信したメッセージから受信時刻を求め, メッセージ受信に必要な時間を求める.バリア同期のイベントに対しては,バリア同期を最も遅く要求したプロセッサの要求時間をプロセッサ間で求め, バリア同期に必要なオーバヘッドを求める.

純粋なユーザプログラムの実行時間は, ライブラリ終了と次のライブラリ開始の間の時間として求めることができる.この時間も計算性能をパラメータとして変更することができる.

これ以外にも, リングバッファのサーチの影響, 送信と計算のオーバラップ, メッセージのユーザ領域への直接受信などの効果をシミュレーションできる.

2.3 ネットワークシミュレーション

メッセージレベルシミュレータでは, ネットワークの遅延, とくに, 輻輳によるアプリケーションプログラム全体への影響を求めるため,クロックレベルのネットワークシミュレータを開発し, ネットワークの輻輳を考慮にいれたシミュレーションを可能としている.

ネットワークのシミュレーションを行なうときは,2.2で述べた方法とシミュレーションの方法が多少異なる.メッセージの送信においては, 送信先プロセッサに送信時刻情報のメッセージを送信するのではなく,ネットワークシミュレータにメッセージを送信する.ネットワークシミュレータは,それを受信し, 送信時刻で送信情報メッセージをソーティングする.すべてのプロセッサにあるメッセージレベルシミュレータがメッセージ受信待ちあるいはシミュレーション終了状態になると, ネットワークシミュレータは実行を開始する.ネットワークからメッセージが出力されると(CPUからみるとメッセージを受信したとき), ネットワークシミュレータがメッセージレベルシミュレータに対してメッセージを送信し, 受信したプロセッサが再びシミュレーションを開始する.これを繰り返すことによりネットワークの輻輳による遅延を考慮したシミュレーションを行なうことができる.

3 パフォーマンスアナライザ

パフォーマンスアナライザは, メッセージレベルシミュレータと同様にAP1000上で動作し, プログラムの性能解析を行なう.

従来の並列システムに対するパフォーマンスアナライザは [2, 3, 4], 大規模なシステムへ適用するための課題である, (1) トレース情報の巨大化にともなうデータの収集と解析, (2) 膨大なデータの抽象化と表示方法, について解決がなされていなかった.本ツールは, これらの課題を解決し, 大規模なシステムへ適用することを可能としている.

パフォーマンスアナライザは,アプリケーションプログラムが直接出力したトレース情報, もしくは,メッセージレベルシミュレータが出力したトレース情報をもとに, 個々のプロセッサの内部状態, プロセッサ全体の稼働率とアイドル率, メッセージ転送量などをグラフィック表示あるいは統計情報として表示する.トレース情報は, プロセッサ上のメモリにあるので, 大規模なシステムにおいてもパフォーマンスアナライザで解析することができる. また, 解析そのものも並列実行されるので高速な解析を実現している.

パフォーマンスアナライザは, メッセージレベルシミュレータと同じ構成をしている.異なる点は, トレース情報を元に解析するだけであり, トレース情報の更新は行なわない点である.

3.1 パフォーマンスアナライザの機能

パフォーマンスアナライザの機能は, 以下の通りである.

- 統計解析
  - 発生したイベントの種類と時刻
  - ライブラリコールの種類, 回数, 時間
  - タスクの実行, アイドル, 割り込みの時間
  - メッセージの転送量と転送距離
  - メッセージの種類 (送信先, 送信元, サイズ, タイプ)
  - メッセージ待ち時間と同期待ち時間
- グラフィックス表示
  - プロセッサ稼働率
  - オーバヘッド率 (アイドルと通信)
  - 各プロセッサの状態
  - メッセージ通信状態 (通信回数, 通信量, 通信距離, 受信待ちイベント, 送信トラップイベント)

統計解析については, 個々のイベントから集計したデータを数値として表示するものである.各データは, プロセッサごとにあるいは全プロセッサ間の最大 / 最小 / 平均値が求められる.グラフィックス表示については, 次節で述べる.

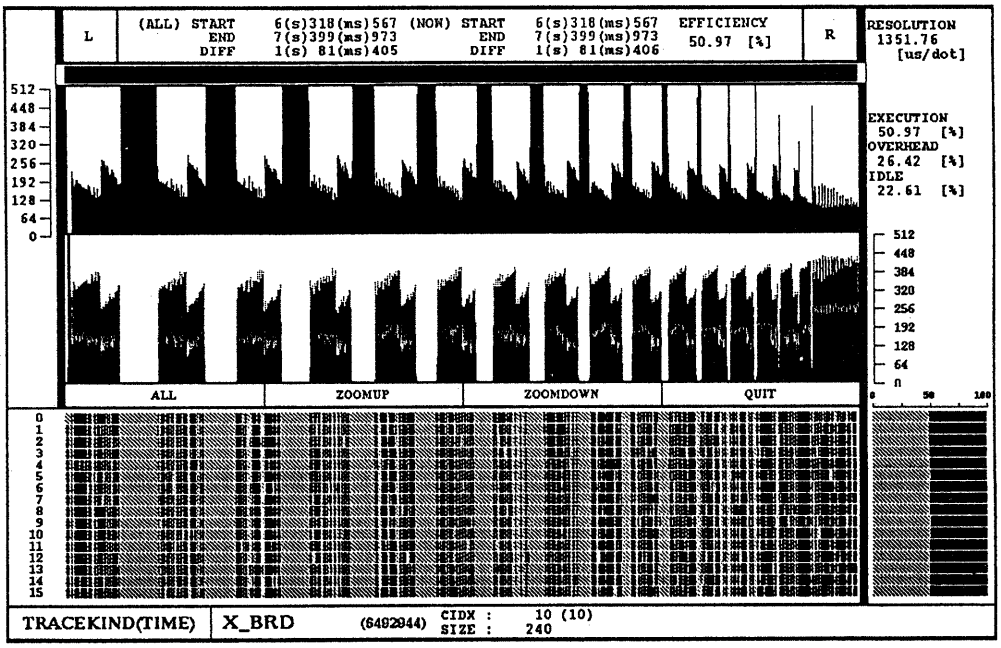


図 3: パフォーマンスアナライザの表示例

表 1: アプリケーションの内容

アプリケーション	内容	文献
LINPACK	密行列 (1000 × 1000) をブロック LU 分解によって解くプログラム.	[5]
QCD	QCD のモンテカルロシミュレーション.	[6]
SLALOM	直方体内部の光の分布をラジオシティで解く. パッチ数は, 500.	[7]
AMBER	粒子分割を用いた水とタンパク分子の分子動力学シミュレーション.	[8]
MD	領域分割を用いた液体原子の分子動力学シミュレーション.	[9]
SCG	二次元のポアソン方程式を SCG 法で解く.	[10]
OCEAN	並列言語 Dataparallel-C で記述した海流シミュレーション.	[11]
SHALLOW	並列言語 Dataparallel-C で記述した二次元グリッド *shallow-water* 方程式.	[11]
ORTHES	OXYGEN でコンパイルされた Householder 変換プログラム.	[12]
TSDE	OXYGEN でコンパイルされた SOR プログラム.	[12]
TRANSPOSE	密行列 (1000 × 1000) の転置.	

3.2 パフォーマンスアナライザの表示例

パフォーマンスアナライザの表示例を図3に示す. 表示にはX-Windowを用いた. 実行したプログラムは, 4章で述べるアプリケーションプログラムLINPACKの結果である. 表示は, いずれも横軸が時間で, 上から順番に, 稼働率, オーバヘッド(黒がアイドル, グレーが通信処理), プロセッサごとの内部状態(ここでは, プロセッサ0～15のみ)を表している. プロセッサの内部状態は, 実行, 通信, 割り込みなどの状態を色分けして示しており, プロセッサの内部状態表示の横のグラフは, 指定した時間内の各状態の割合を示している. 一番下は, イベント情報を示しており, これは, プロセッサの内部状態表示部分をクリックすることにより, イベントの種類とそれに関連する情報を知ることができる. また, オーバヘッドを表示している箇所に, メッセージ通信状態を表示することができる. なお, 表示する時間の範囲は, 任意に指定することが可能である.

4 解析例

ここでは, メッセージレベルシミュレータを用いた解析例を示す. 解析例として, まず, アプリケーションのオーバヘッドの内容, 通信情報(回数, 距離, メッセージサイズなど)を求める. これらの統計情報については, HsuとAnnaratoneがメッセージパッシングのモデルを想定し, 実測あるいはシミュレーションにより求めている [13, 14]. しかし, 実測においてトレースによるオーバヘッドが含まれていたり, シストリック的に通信を行なうモデルで, 現在のメッセージパッシング型並列システムが備えているハードウェア機構を十分に反映したシミュレーションになっていなかったりした.

次に, 通信性能と演算性能を変化させたときの全体性能への影響をシミュレーションにより求める.

4.1 アプリケーション

解析対象としたアプリケーションを表1に示す. これらは, いずれもAP1000のメッセージパッシングライブラリを使用して並列化されたプログラムである. コンパイラが出力するコードもこのメッセージパッシングライブラリを使用したコードを出力する. 解析する部分は, アプリケーション全体ではなくプロセッサ側で実行されている部分だけである. 従って, ホストコンピュータとの通信は含まれていない.

4.2 シミュレーション結果

表3は, メッセージシミュレータによりシミュレーションし, パフォーマンスアナライザによりアプリケーションの統計情報を求めたものである. メッセージレベルシミュレータの通信, 同期などのパラメータは, AP1000の基本的な通信性能を実測し求めた. なお, このシミュレーションにおいては, ネットワークの輻輳あるいは通信距離による遅延は考慮していない.

まず, AP1000の実行時間とメッセージレベルシミュレータの実行時間との比較を行なう. 多くのアプリケーションで, その差は数パーセント以下である. TRANSPOSEでは, おおきな差が出ているが, これはネットワークの輻輳のためと考えられる. 一般にメッセージレベルシミュレータによるシミュレーションは, ネットワークの輻輳があまりない場合においては, 正確な実行を行なうことができることを示している.

パフォーマンスアナライザは, AP1000で実行して出力されたトレース情報をそのまま読み込み解析することができる. しかし, AP1000の実行においてトレースの出力による大きなオーバヘッドを生み出す可能性がある. 実際, AP1000の実行において, トレース出力を行なうとトレースを行なわない場合に対して実行時間が10～40%も遅くなってしまう. メッセージレベルシミュレータにより, トレースによるオーバヘッドのない解析を行なうことができるようになった.

表3において, 実行時間の単位は秒, トレースサイズはすべてのプロセッサでの合計値, メッセージサイズは, 1通信あたりの平均のメッセージのサイズ, 通信回数(1対1通信), 放送回数, GOP回数は, いずれも1プロセッサあたりの平均回数, 通信距離は, 1対1通信1回あたりの平均の通信ホップ数である. GOPは, グローバル演算(プロセッサ間での総和などを求める演算)である. ユーザ, 通信, アイドル, 送信, 受信, 放送, GOP, 同期, メッセージ読込はいずれも全実行時間の各項目の時間の割合を示している. アイドル時間には, 受信におけるメッセージ待ち時間と同期待ち時間が含まれる. ここでの同期は, バリア同期のことを指す. また, 放送とGOPは, すべてのプロセッサを対象, X方向のプロセッサを対象, Y方向のプロセッサを対象としたものが含まれている.

トレースサイズは, アプリケーションによっては数百Mバイトにもおよび, プロセッサのローカルメモリを利用して, シミュレーションと解析を行なう方法でなければ実現は不可能と思われる.

表3から, アイドル時間の割合がかなり大きいことがわかる. とくに, 台数を増やすと, 処理のオーバヘッド以上にアイドルの割合が増えている. これについては次節でも述べる.

4.3 通信性能の影響

アプリケーションが通信性能に対してどれだけ影響を受けるかをシミュレーションにより求めた. 演算性能は一定である. 図4は, アプリケーションごとの通信性能による性能変化を示したものである. 縦軸は通信オーバヘッドが0であるとしたとき(通信性能が無限に良い場合)の性能を1とし, 横軸はAP1000の通信性能を1とする. ここでの通信性能は, メッセージのハンドリングあるいはネットワークのスループットなどをすべて同じ尺度で考えている. なお, アプリケーションの名前の後ろの数字は, プロセッサ台数である.

この図からAP1000の通信性能が必ずしもアプリケーションの並列性を十分に引き出しているとは限らないことがわかる. また, AP1000のCPU性能をそのままにして, 現状より16倍以上高い通信性能を実現しても全体の性能はあまり向上しないことがわかる. アプリケーションのなかでORTHES64が最も通信性能の影響をうけている. 一方, SLALOM256は, 表3からユーザ実行時間の割合が45.4%ともっとも低いが, 38.1%がアイドル時間であるため, ORTHES64よりも通信性能の影響を受けていない.

ここでは, 通信オーバヘッドが0のときを最大性能として考えたが, 実際には, 通信オーバヘッドが0でもアイドル時間があるためアプリケーション全体の性能が高くなっているとは限らない. そこで, 通信オーバヘッドが0のときのユーザ実行時間とアイドル時間の割合を表2に示す.

アイドルの割合は, 通信オーバヘッドが小さくなると小さくなる傾向にあるが, 通信オーバヘッドに依存するとは言えない. プロセッサ台数が増えるとアイドルの割合が20%を越してしまう場合もあり, 大規模システム

表 3: アプリケーションの統計情報

アプリケーション	PE台数	AP1000実行時間	シミュレータ実行時間(比%)	ユーザ(%)	通信(%)	アイドル(%)	トレースサイズ(MB)	メッセージサイズ(B)
LINPACK	64	3.508	3.496 (-0.3)	77.0	10.9	12.0	58.8	368
LINPACK	128	2.219	2.200 (-0.9)	67.9	18.0	14.1	163.3	237
LINPACK	256	1.403	1.369 (-2.4)	60.3	18.7	21.0	242.6	202
LINPACK	512	1.107	1.080 (-2.4)	51.0	26.5	22.5	656.9	109
SCG	64	4.142	4.084 (-1.4)	88.9	8.6	2.5	36.4	804
SCG	256	1.255	1.229 (-2.1)	60.3	18.7	21.0	193.2	419
AMBER	64	0.989	0.965 (-2.4)	80.8	7.1	12.1	8.0	440
AMBER	128	0.634	0.634 (0.0)	74.1	10.7	15.2	14.2	127
TRANSPOSE	64	0.073	0.053 (-27.4)	50.8	49.0	0.1	1.9	1000
TRANSPOSE	256	0.049	0.023 (-53.1)	56.4	43.6	0.1	5.5	504
MD	64	3.885	3.723 (-4.2)	96.4	2.4	1.2	10.1	2201
MD	512	0.521	0.500 (-4.0)	89.1	5.9	5.0	39.0	599
QCD	64	5.121	5.053 (-1.3)	97.9	2.1	0.1	1.48	19749
OCEAN	64	7.903	8.082 (2.3)	82.4	15.7	2.0	462.0	12
SHALLOW	64	3.929	3.593 (-8.6)	92.3	7.1	0.7	56.9	457
ORTHES	64	1.534	1.691 (10.2)	46.6	41.9	11.5	202.9	8
TSDE	64	0.519	0.516 (-0.6)	73.6	3.4	23.0	3.1	361
SLALOM	64	2.35	2.366 (0.7)	69.4	10.6	20.0	72.9	207
SLALOM	256	1.05	1.100 (4.8)	45.4	16.5	38.1	208	112

アプリケーション	PE台数	通信回数	放送回数	GOP回数	通信距離	送信(%)	受信(%)	放送(%)	GOP(%)	同期(%)	メッセージ読込(%)
LINPACK	64	17	857	1000	1.8	0.0	0.0	20.0	2.8	0.0	0.0
LINPACK	128	30	810	1008	1.8	0.0	0.0	26.7	5.4	0.0	0.0
LINPACK	256	29	437	1012	2.3	0.0	0.0	32.9	6.6	0.0	0.0
LINPACK	512	14	407	1026	2.3	0.0	0.0	38.0	10.9	0.0	0.0
SCG	64	1756	0	893	1.1	2.1	5.5	0.0	3.5	0.0	0.0
SCG	256	266	0	893	1.4	6.7	16.1	0.0	14.5	0.0	0.0
AMBER	64	189	6	75	4.0	0.6	0.4	7.6	10.7	0.0	0.0
AMBER	128	383	6	75	6.0	1.3	1.0	12.8	10.8	0.0	0.0
TRANSPOSE	64	109	0	0	4.6	8.2	31.7	0.0	0.0	4.9	0.0
TRANSPOSE	256	59	0	0	8.5	9.1	22.7	0.0	0.0	8.6	0.0
MD	64	199	0	10	2.2	0.6	1.0	0.0	0.0	0.0	1.6
MD	512	199	0	10	3.6	1.6	4.3	0.0	1.5	0.0	3.5
QCD	64	28	1	0	1.0	0.5	0.0	0.0	0.0	0.0	1.5
OCEAN	64	41000	0	0	1.1	5.5	12.0	0.0	0.0	0.0	0.0
SHALLOW	64	4800	0	0	1.1	4.4	3.3	0.0	0.0	0.0	0.0
ORTHES	64	0	311	7437	0.0	0.0	0.0	32.5	21.3	0.0	0.0
TSDE	64	158	0	44	3.5	0.9	2.7	0.0	22.5	0.0	0.0
SLALOM	64	3450	0	0	2.4	3.0	22.6	0.0	0.0	0.0	5.1
SLALOM	256	3274	0	0	4.3	4.7	43.4	0.0	0.0	0.0	6.5

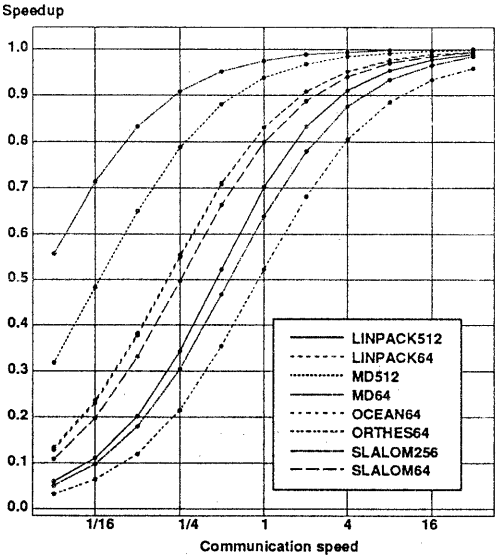


図 4: 通信性能の影響

表 2: 通信オーバヘッド 0 のときのユーザ時間 / アイドル時間の割合

アプリケーション	台数	ユーザ (%)	アイドル (%)
LINPACK	64	92.6	7.4
LINPACK	128	87.8	12.2
LINPACK	256	85.3	14.7
LINPACK	512	77.9	22.1
SCG	64	98.0	2.0
SCG	128	89.5	10.5
SLALOM	64	86.7	12.3
SLALOM	256	71.1	28.9
AMBER	64	88.5	11.5
AMBER	128	87.8	12.2
MD	64	98.8	1.2
MD	128	95.0	5.0
ORTHES	64	89.3	10.7
TSDE	64	76.7	23.3

では大きな問題である. このアイドル時間を減らすための一つの方法として, データをまとめて通信するのではなく, 通信粒度を細かくして実行可能なデータをなるべくはやく通信する方法がある. そのためには, アプリケーションプログラムの書き換えとアーキテクチャのサポートが必要である. この細粒度通信によるアイドル時間の削減については別途報告したい.

4.4 CPU 性能の影響

ここでは, 通信性能が同じで, CPU の性能を変化させたときの実行時間への影響を求める. CPU の性能が向上すると, AP1000 などでは, メッセージハンドリングにおいてソフトウェアがかなり介在するので, メッセージハンドリングのオーバヘッドも小さくなる. ここでは, まず, CPU 性能により純粋に演算性能だけが向上する場合を評価する. これは, AP1000 の CPU にベクトルユニットなどを付加することにより演算性能だけが向上する場合にあたる. 表 5にその結果を示す.

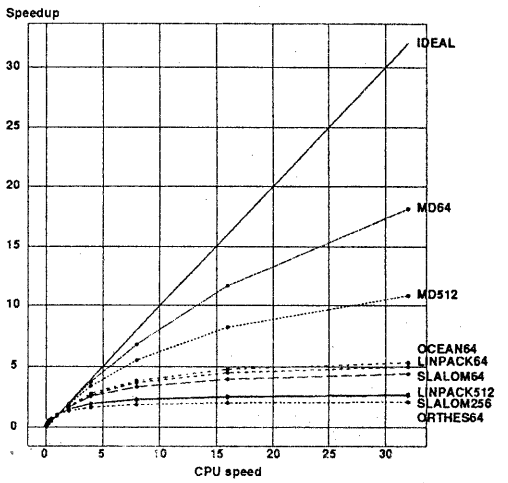


図 5: CPU 性能の影響 (演算性能のみ)

次に, CPU の性能向上に比例してメッセージのハンドリング性能も向上する場合を評価する. これは, AP1000 の CPU をより速いものに置き換え, ネットワークをそのままにした場合に相当する. 表 6にその結果を示す.

演算性能のみ向上させても, MD64 と MD512 の演算中心のアプリケーションを除いて性能の向上は望めない. MD64 と MD512 を除くと, 32 倍の CPU を持ってきても最大 5 倍程度の性能向上しか得られない. メッセージハンドリングを含めた CPU 性能の向上に対しては, OCEAN64, ORTHES64 が高い性能向上を示している. これは, この二つのアプリケーションが通信しているメッセージが細かく (表 3から, それぞれ, 平均で 8 バイトと 12 バイト), メッセージハンドリングの影響を受け, ネットワークのスループットの影響をあまり受けないからである.

5 おわりに

メッセージレベルシミュレータの構成と機能について述べた. メッセージレベレシミュレータは, 演算性能

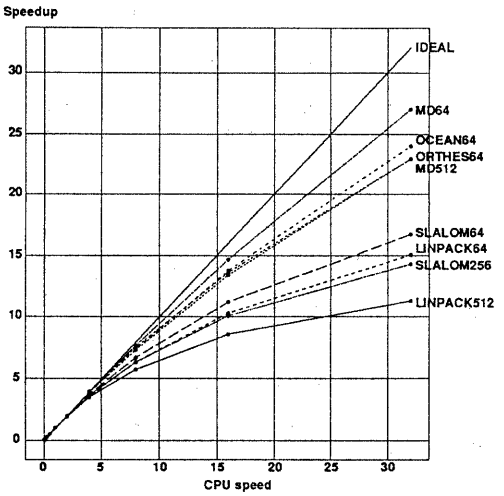


図 6: CPU 性能の影響 (メッセージハンドリングを含む)

と通信性能をパラメータとして与えることにより, アーキテクチャの評価を行なうことができる.

このメッセージレベルシミュレータを用いて AP1000 で動作しているアプリケーションを評価した. まず, AP1000 での実測値とシミュレータによる評価値とを比較し, メッセージレベルシミュレータが高い精度を持つことを示した.

次に, 性能評価機能を用いてアプリケーションのオーバヘッドの内容, 通信情報 (回数, 距離, メッセージサイズなど) を求めた.

さらに, AP1000 の通信性能を変化させたときの全体性能への影響, AP1000 の演算性能を変化させたときの全体性能への影響をシミュレーションにより求めた. その結果, 特に, メッセージサイズの小さなアプリケーションでは, AP1000 のメッセージハンドリングのオーバヘッドが大きく, 通信性能向上により全体性能がかなり高くなる (通信性能 4 倍で全体性能 1.6 倍) ことがわかった. また, プロセッサ台数が増えたときアイドル時間が増えるが, これは通信オーバヘッドが 0 の場合もやはり大きく, アイドル時間の削減のためには, 細粒度通信による並列性向上などの工夫が必要である.

以上のようにメッセージレベルシミュレータにより有効な評価結果が得られた. 今後は, メッセージレベルシミュレータの機能拡張とそれを使用した次世代アーキテクチャの評価を行なっていく予定である. 特に, 現在のシミュレータは AP1000 の物理的なプロセッサ台数を越えたシステムの評価が行なえない. これは, 仮想プロセッサを実現することにより, より大規模なシステムの評価を行なえるようにする予定である. また, アーキテクチャのより詳細な評価 (通信のオーバラップやマルチスレッドなど), 輻輳したネットワークの影響を受けた場合の性能についても検討する予定である.

謝辞

日頃御指導, 御助言いただく, 並列処理研究センター石井センター長, アーキテクチャ研究部白石部長, 並列処理研究センター第二研究室佐藤室長, 池坂主任研究員, ならびに研究室の同僚諸氏に感謝いたします. また, 本プロジェクトに御協力賜わる (株) 富士通ソーシアルサイエンスラボラトリの諸兄に深謝します.

参考文献

[1] H. Ishihata, T. Horie, S. Inano, T. Shimizu, S. Kato, and M. Ikesaka: "Third generation message passing computer AP1000", In *International Symposium on Supercomputing*, pp. 46–55, Nov. 1991.

[2] T. J. Leblanc, J. M. Mellor-Crummey, and R. J. Fowler: "Analyzing parallel program executions using multiple views", *Journal of Parallel and Distributed Computing*, 9, pp. 203–217, 1990.

[3] A. D. Malony and D. A. Reed: "A hardware-base performance monitor for the Intel iPSC/2", In *International Conference on Supercomputing*, 1990.

[4] A. D. Malony, D. A. Reed, and D. C. Rudolph: "Integrating performance data collection, analysis, and visualization", In *Parallel Computer Systems: Performance Instrumentation and Visualization*. Addison-Wesley Publishing Company, 1990.

[5] R. P. Brent: "The LINPACK benchmark on the AP1000", In *Fourth Symposium on the Frontiers of Massively Parallel Computation*, Oct. 1992.

[6] K. Akemi et al.: "QCD on the highly parallel computer AP1000", In *Nuclear Physics B*, number 26, pp. 644–646, 1992.

[7] J. Gustafson et al.: "Slalom update", *Supercomputing Review*, pp. 56–61, March 1991.

[8] H. Sato et al.: "Parallelization of AMBER molecular dynamics program for the AP1000", In *Scalable High Performance Computing Conference 92*, pp. 113–120, April. 1992.

[9] D. Brown and J. H. R. Clarke: "Parallelization strategies for MD simulations on the AP1000", In *Proceedings of the Second Fujitsu-ANU CAP Workshop*, pp. L-1–10, Nov. 1991.

[10] 速水 謙: *SCG* 法による拡散方程式の解法. コロナ社, March 1989.

[11] P. J. Hatcher and M. J. Quinn: *Data-Parallel Programming on MIMD Computers*. The MIT Press, 1991.

[12] R. Ruehl: "Evaluation of compiler generated parallel programs on three multicomputes", In *Proc. of the Sixth International Conference on Supercomputing*, June 1992.

[13] J. M. Hsu and P. Banerjee: "Performance measurement and trace driven simulation of parallel CAD and numerical applications on a hypercube multicomputer", In *The 17th Annual International Symposium on Computer Architecture*, pp. 260–269, May 1990.

[14] M. Annaratone, C. Pommerell, and Roland Ruhl: "Interprocessor communication speed and performance in distributed-memory parallel processors", In *The 16th Annual International Symposium on Computer Architecture*, pp. 315–324, May 1989.