Information Processing Society of Japan

情報処理学会研究報告
コンピューティング 58－4
（1995．10．18）

疎結合並列計算機 Paragon の性能評価

― ノード間通信性能の評価を中心に ―

川端 英之
広島市立大学
kawabata@ce.hiroshima-cu.ac.jp

津田 孝夫
京都大学
tsuda@kuis.kyoto-u.ac.jp

疎結合並列計算機 Intel Paragon の通信性能の評価を行なった。通常、大規模並列計算機としての側面が強調され、数千台規模構成による Toward Peak Performance の値が目を引くが、筆者は、並列プログラミングの際の重要な指針となる通信性能の評価を 16 ノード構成の Paragon に対して試みた。とくに、ユーザレベルで使用可能な非同期転送命令の有効性の評価を行なった。その結果、非同期転送中のメモリあるいはバス競合と見られるオーバヘッドによる演算性能の低下により、必ずしも非同期転送命令が有効であるわけではないことが分かった。

A Performance Evaluation of a Loosely Coupled Parallel Computer

— The Performance of Inter-Node Network of Paragon —

Hideyuki KAWABATA
Hiroshima City Univ.
kawabata@ce.hiroshima-cu.ac.jp

Takao TSUDA
Kyoto Univ.
tsuda@kuis.kyoto-u.ac.jp

We evaluated a loosely coupled parallel computer: Intel Paragon. Generally, the machine is thought to be a "massvely parallel supercomputer" to solve huge numerical problems, and in fact, the machine's "Toward Peak Performance" is fascinating. We tried to evaluate the machine's inter-node communication performance, especially the performance of asynchronous communication. The experimental results show that asynchronous comunication is not always good to use because bus and/or memory conflicts between application and communication processors may occur.

NII-Electronic Library Service

Information Processing Society of Japan

1 はじめに

大規模な数値計算をより高速に解くことに対する要求は絶えない。これに対し、並列計算機を用いることにより、従来の単一 CPU マシンでは得られなかったパフォーマンスを獲得しようとする研究・開発が盛んに行なわれている。そして既に多種多様の高速な並列計算機が製品化されている。

筆者は、並列プログラミングの指針となる情報を得ることを目的とし、商用並列計算機の一つである Intel Paragon XP/E-16 (以下 Paragon と記述) を例にとり、いくつかの性能測定および評価を行なった。本稿ではその調査内容および結果・評価について報告を行なう。

以下、2 章では使用した Paragon の諸元について述べる。3 章では基本的な通信性能の実測結果を示し、4 章ではアプリケーションの実行を交えた Paragon の性能実測および考察について述べる。

2 Paragon の諸元

Paragon は二次元メッシュの結合網をもつ疎結合並列計算機で、数千ノードの超並列構成が可能である。各ノード (GP ノードと呼ばれる) には計算プロセッサおよび通信プロセッサとしてそれぞれ Intel i860XP が使用されており、高速な浮動小数点演算や通信が可能である[1]。ノード間ネットワークは片道 200 MB/s の双方向通信が可能で、ワームホールルーティングがなされる。このたび使用した計算機の構成について図 1 に示す。なお実測に使用した計算機は Paragon XP/E-16 (16 GP-NODE) である。

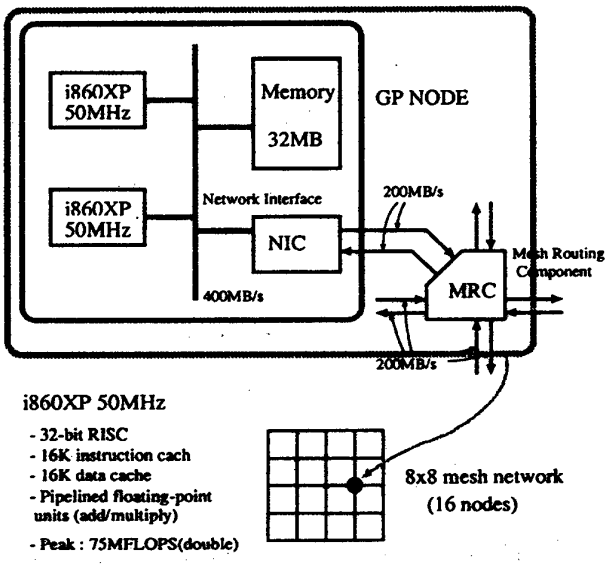


図 1: Paragon の構成

3 Paragon の通信性能

3.1 一対一通信

最初に、一対一通信の性能の実測結果を図 2に示す。図 2は、ある一対のノード間で通信を行ったときの、転送単位長に対する通信速度を示すものである。上のグラフは下のグラフの横軸を対数スケールに直したものである。ノード 0 とその他のノード (ノード 1, ..., ノード 15) との各対間 (計 15 対) について、同一の大きさのデータを往復させるのに要した時間を二分することにより転送速度を算出した。なお、受信ノードにおいてデータがシステムバッファを介することなく直接アプリケーションに渡るように、受信側が受信命令を発行したことを確認してから送信を始めるようにした。

図 2には各対ごとの測定結果を重ねて表示しているが、この図では判別不可能である。4×4 のメッシュの 16 ノード構成においては (少なくともメッシュ上のトラフィックが僅かである場合には) 、ノード間の距離は通信時間にはほとんど影響しないといえる。ノード間通信時間は、52.4 KB 単位の転送で約 6 msec であった。

ここで、t_0 及び t_1 をそれぞれ通信のレイテンシ及び 1 バイトの転送時間とし、n バイトの転送にかかる時間が以下のように表現できると仮定する。

$$t_0 + nt_1$$

図 2中に、(t_0, t_1) をそれぞれ $(130\times10^{-6}, 13\times10^{-9})$ および $(70\times10^{-6}, 11\times10^{-9})$ とした場合の曲線を記入した。これを実測で得られた曲線と比較すると、実際の値は $t_0 \simeq 130 \times 10^{-6}$ および $t_1 \simeq 12 \times 10^{-9}$ であることが伺える。つまり $t_0 \simeq t_1 \times 10^4$ であり、Paragon の通信レイテンシは非常に大きいといえる。実際図 2は、40 MB/s の転送速度を得るためには転送単位を 10KB 程度以上とする必要があることを示している。

図 2 では、転送単位が 8192 B から 16384 B に変わる時点で、転送速度が減少するか、あるいはほとんど増加していない。この原因ははっきりはしないが、i860XP の持つ 16KB のデータキャッシュにメッセージが収まっているか否かで転送に必要な時間が異なる可能性がある。

3.2 ブロードキャスト

Paragon にはブロードキャストのための特殊なハードウェアは用意されていない。Paragon のブロードキャストの性能測定を行った結果を図 3に示す。ノード 0 において、他の 15 ノードに対するブロードキャストを発行し、各ノードからノード 0 までの返信が到着するまでの時間を測定した。図 3によると、ブロードキャストが完了するまでの時間は 1 対 1 通信の場合と比較して 3 倍～ 6 倍かかっている (52.4 KB で 37 ～ 65 msec) 。また、ノードの位置によって到着時間にばらつきが見

[1] この他、一つのノードに i860XP を三個搭載した MP ノードと呼ばれるものもある。この場合、各 MP ノード自体を共有主記憶の 2CPU 並列マシンと見ることができる。

NII-Electronic Library Service

Information Processing Society of Japan

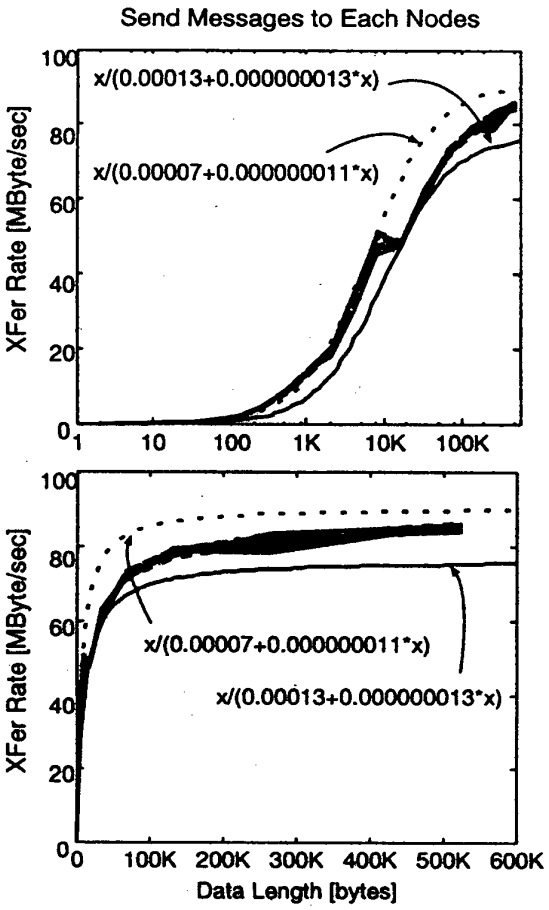


図 2: 一対一通信の速度

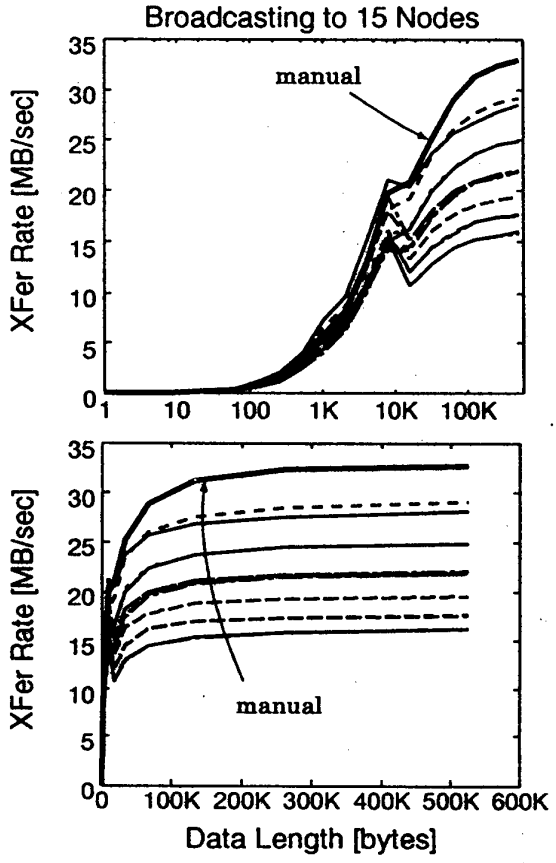
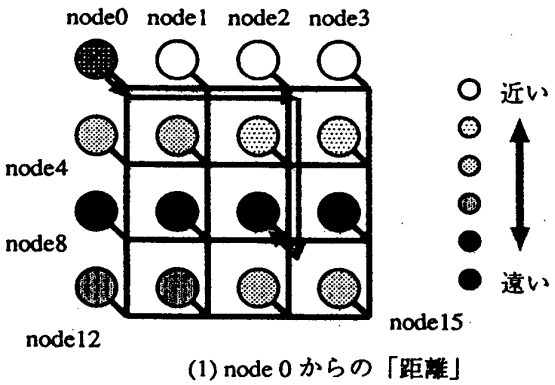


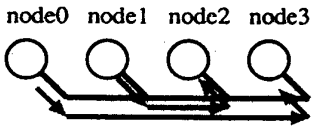
図 3: ブロードキャストメッセージ転送速度

られる。図 4 (1) は、どの ノード が ノード 0 から「遠い」かを図 3に基づいて示したものである。どのようなパターンでブロードキャストが行なわれるのかは、これだけの情報では判明しないが、段階的にデータをばらまいていることが伺える。

ここで、一対一通信を繰り返すことによる「手動の」ブロードキャストとの比較を試みた。一対のノード間での一回の送受信を一ステップとすると、一つのノードの情報を（ブロードキャスト機能を利用せずに）他の $n-1$ ノードにばらまくためには $\lceil \log_2 n \rceil$ ステップあればよい。この方法で、ノード 0 から他の 15 ノードにブロードキャストを行ない、最後に（4 ステップ後に）データを受信したノードがノード 0 に完了報告を送るまで（計 5 ステップ）の時間を測定したのが、図 3 中の “manual” と記述した太線である。52.4 KB を 15 ノードにばらまいた場合 31 msec を費やしたが、これは一対一通信における 6 msec （前節参照）を 5 倍した値にほぼ等しい。図 3より、OS 附属ライブラリのブロードキャスト機能を利用するよりも個別に一対一通信を繰り返すほうが短時間で済む場合があることが判明した。ただしこの方法では、非同期ブロードキャストを行なうことは出来ない。



(1) node 0 からの「距離」



(2) 通信路の重複

図 4: Paragon のノード間通信

NII-Electronic Library Service

Information Processing Society of Japan

3.3 通信路の重複

通信路が重複する場合の転送速度の実測を行なった。Paragon のノード間通信の経路は、送信ノードと受信ノードが同一ならば常に同一で、すなわち衝突を避けて迂回するようなことは行なわれない。図4 (2) でノード0 および 1 がそれぞれノード 3 および 2 にメッセージを送る場合、ノード 1 とノード 2 の間の通信路が両方の通信で同時に使用される。このときのデータ転送速度を測定した。結果を図5 に示す。

図5によると、通信路が重複しない場合 (図2) と比較して転送速度は一割程度低下しているが、依然として 60 MB/sec を超えるデータ転送速度が達成されている (これは同時に、ハードウェア的な能力に対して OS によるパケットハンドリングのオーバヘッドが大きいことを示すともいえる)。4×4 のメッシュでは最大でも三重の重複程度にしかならないことを考えると、各ノードが頻繁に通信を行なって通信路が込んでいる状態でも、転送単位が充分大きければ 60MB/sec 程度の転送速度が期待できるといえる。

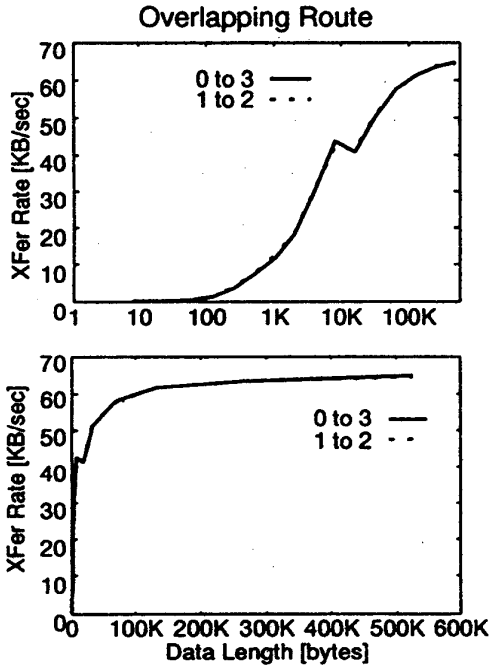


図 5: 通信路が重複する場合の転送速度

3.4 非同期通信

Paragon は非同期通信ライブラリを備えている。

同期送信では、少なくとも受信ノード内のシステムバッファにデータが完全に転送されるまで、また同期受信ではユーザの用意するバッファに完全にデータが書き込まれるまで、通信を発行したプログラムの実行はブロックされる。これに対し非同期送受信では、通信発行後直ちに制御がアプリケーションに復帰する。すなわち、非同期通信ライブラリを利用することで文字通りデータ転送時間を演算時間とオーバラップさせることができる。非同期通信のパフォーマンスについては、次節のアプリケーション実行例で合わせて述べる。

4 アプリケーション実行例

4.1 行列積

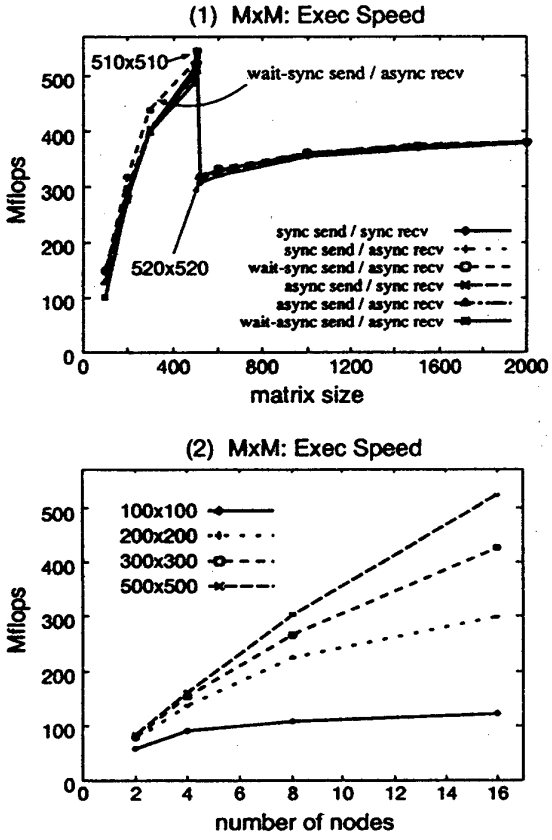


図 6: 実行速度の変化

以下に示す行列積の計算を行なった。

```
DO JJ = 1, N/WIDTH
[(1) 一つ後の繰り返しでの計算のためのデータ転送]
 DO I = 1, N
 DO J = (JJ-1)*WIDTH, JJ*WIDTH
 DO K = 1, N
   C(I, J) = C(I, J) + A(I, K) * B(K, J)
 ENDDO
 ENDDO
 ENDDO
[(2) 次の繰り返しのためのデータ受信確認]
 ENDDO
```

アルゴリズムはいわゆる内積形式で、B, C をそれぞれ行方向および列方向に短冊状に分割して各ノードに配置するのに合わせ、ループ分割・交換によるブロック化を行なっている。上記の WIDTH は B,C の分割幅を示す。これ以上のアンローリングなどの最適化は行なっていない。計算は全て倍精度で行なった。またデータ送受信は、次のように六通りの組合せで行なった。ただし一つのプログラム中では全ノードの振舞いは同一とした。

NII-Electronic Library Service

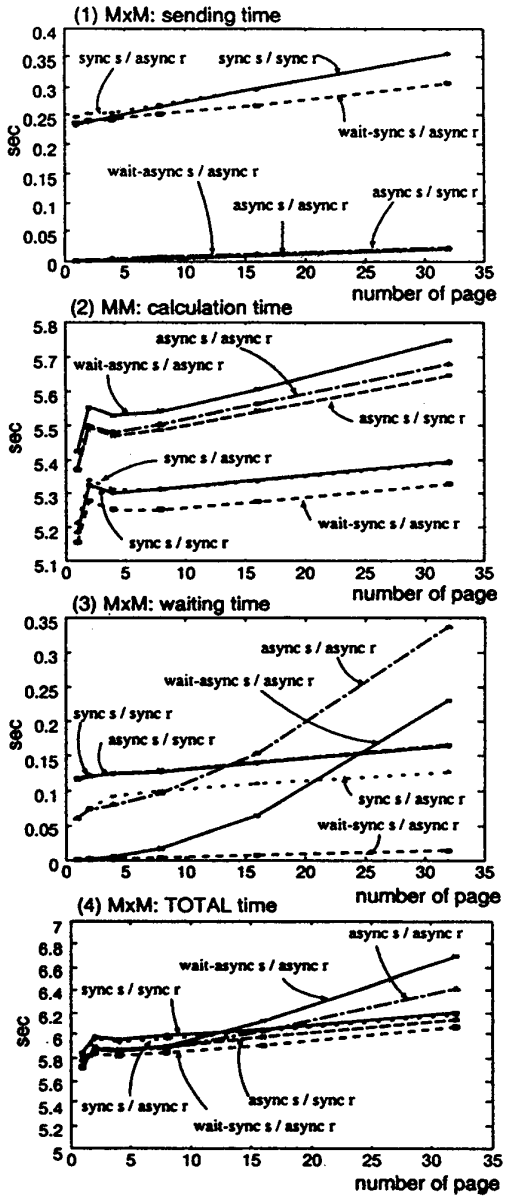


図 7: ノードごとのページ数と実行時間の対比

- 送信・受信ともに同期あるいは非同期
- 送信のみ同期、あるいは受信のみ同期
- 非同期受信の場合に、受信命令発行確認後に送信

送信命令および非同期受信命令の発行は上記プログラムの (1) の位置で行ない、同期受信および（非同期受信時の）データ受信確認は (2) の位置で行なう。送受信ともに (2) の位置で行なう方法もあるが、演算とのオーバラップの効果が生じないためここでは考慮に入れなかった。

図 6 (1) は行列のサイズ別の実行結果である。16 ノードで実行した。通信方式による大きな差は見られないが、“同期送信・非同期受信・受信確認有り” が常に最速である。なおこの図から、明瞭なパフォーマンスの不連続性が見受けられる。これはキャッシュの影響であろう。実際、500× 500 の行列積の最内側ループは 4KB の

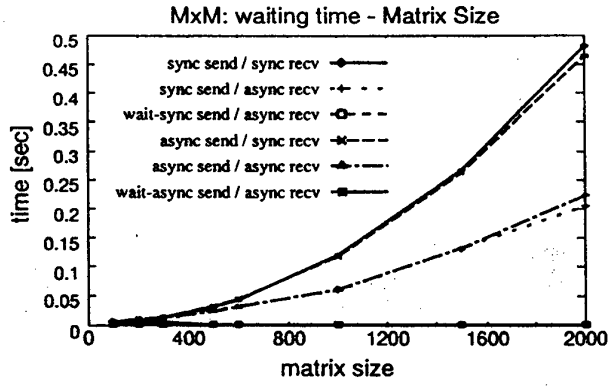


図 8: 行列サイズと受信待ち時間の比較

ベクトルどうしの内積で、各プロセッサのもつ 16KB データキャッシュ (on chip) が効果的に働くであろう大きさである（二次キャッシュ等は装備されていない）。

図 6 (2) は “同期送信・非同期受信・受信確認あり” でノード数を変化させた場合の実行結果である。

4.1.1 行列積にみるデータ転送の影響

前述の行列積プログラムを利用して転送性能の実測を行なった。図 7は行列の分割の度合と実行時間の比較を示すグラフで、一つのノードが保持するページ数（各ノードの保持するデータの総量は等しい）を横軸に、プログラムの各部分の実行で費やされた時間を縦軸にとっている。行列の次元数は 1000、通信方式は前述した六通りで行なった。以下、次の表現を用いる。

送信（待ち）時間 同期 / 非同期送信ルーチンを呼んだ後、制御が戻るまでの時間

受信（待ち）時間 同期受信ルーチンを呼んだ後、制御が戻るまでの時間。あるいは、発行済みの非同期受信命令に対する受信完了確認ルーチンを呼んだ後、制御が戻るまでの時間

データは全て各ノード上の実測値の算術平均である。非同期受信の待ち時間は、ノードごとのばらつきが多い。

一般に、送受信の方式によらず、ブロック長が小さいほど（転送回数が増加するほど）オーバヘッドの積み重ねにより各部で費やされる時間は増加している。データ転送単位の大きさの減少にともなう転送時間の増加も影響しているであろう。

図 7から分かる事柄を以下に挙げる。

1. 非同期転送のオーバラップは演算に影響がある

図 7 (1), (2) によると、非同期送信を行なうと、同期送信の場合と比較して送信待ち時間は抑えられるものの計算時間は伸びている。これは、演算中にデータ転送が行なわれることによりノード内のバスおよびメモリで競合が生じるためであろう。一方最も計算時間が短いのは、非同期受信命令発行を待った後に同期送信を行なう（計算は送信終了後に実行を開始する）方法である。受信命令を待ってから送出するか否かによる違いは、受

NII-Electronic Library Service

Information Processing Society of Japan

信バッファに直接データを書き込むことが保証されるか否かである。つまり、システムバッファに受信データがある状態では演算速度が低下するという現象が伺える。

2. 受信命令発行を待つ方が計算実行が速い

図7（3）によると、受信待ち時間が最も短いのは非同期受信命令の発行を待ってから同期受信を行なう方法である。この場合、受信側のアプリケーションバッファにデータが直接書き込まれるため、受信確認時点であらためてシステムバッファからデータをコピーする必要がない。また同期送信であることにより、自分のノードで演算を行なう間にほぼ確実に全てのデータが自分のバッファ上に揃い、演算と通信のオーバラップが実際はほとんど行なわずにすんでいるためと考えられる。これにより 2% の性能向上が得られている。

3. 送受信とも非同期の場合は性能低下が大きい

送受信ともに非同期にすると、ブロック長を短くして転送回数を増加させた時に、転送回数に比例する以上に受信待ち時間が増加している。

4. 行列積では、データ転送時間は実は無視し得る

図7（4）に示される通り、演算時間以外に要する時間が総実行時間に占める割合は $N = 1000$ において一割に満たない。しかも行列積では、行列の次元数 N に対してデータ転送量および演算時間はそれぞれ N^2, N^3 に比例するため、通信時間はほとんど無視し得る。

表 1: 受信待ちに入るまでの時間と受信待ち時間の対比

方式		通常実行		演算せず		演算量増加	
送信	受信	c	w	c	w	c	w
同	同	4.87	0.11	–	0.13	47.3	0.11
同	非	4.89	0.054	–	0.14	47.4	0.054
同†	非	4.83	0.00043	–	0.014	47.3	0.00056
非	同	5.03	0.11	–	0.38	47.5	0.11
非	非	5.03	0.055	–	0.37	47.5	0.054
非†	非	5.08	0.00062	–	0.22	47.5	0.00073

c: 計算時間 [sec], w: 受信待ち時間 [sec], 同: 同期, 非: 非同期
†: 受信発行を確認して送信

4.1.2 受信待ち状態の挙動についての推測

図8は、データ受信待ち時間が行列の大きさの変化にともなって変化する様子を示す。受信待ち時間は送受信したデータの大きさにほぼ比例している。

ここで、受信待ちに入る段階でどの程度のデータが転送されているのか調べるため、1000 次元の行列積プログラムの演算部分に手を加えて演算量（送信発行から受信確認までの時間）を変化させ、演算中に“十分に転送が完了する”および“転送がほとんど完了しない”という状況を作って実測した（表1）。同期送受信の場合には、待ち時間は演算時間を伸ばしてもほぼ一定である。すなわち、表1 の第一行目（同, 同の行）より、0.13 - 0.11 = 0.02 sec がデータ転送時間で、システムバッファからアプリケーションバッファまでのコピーの時間が 0.11 sec であると解釈できる。こう考えると、同期送信・非同期受信（表1、二行目）では平均して半分程度のデータがシステムバッファに転送され、残りがアプリケーションバッファに直接書き込まれていると捉えることができる。図8で非同期受信の待ち時間のノード間平均が（送信するデータの大きさによらず常に）同期受信の場合のおよそ半分であることも説明できる。

また、受信発行を待って転送する場合（表1、三行目）は、受信までの時間を充分とれば全てアプリケーションバッファに書き込まれるためにほとんど待ち時間は生じないし、即座に受信待ちに入る場合は実際にデータが届くのを 0.14 sec 待っているという状況が推測できる。

非同期送信では、即座に受信待ちにした場合は同期送信の三倍の待ち時間で、充分な時間をおけば同期送信と同様な待ち時間に落ち着いている。受信待ち時間が伸びるのは、同期転送と非同期転送を OS が区別してスケジューリングするなどの影響であると予測される。

4.2 LU 分解

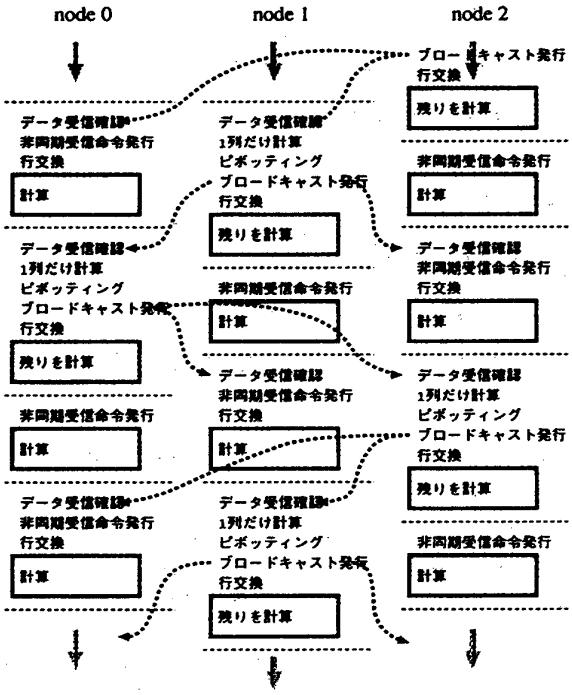


図 9: 並列 LU 分解の様子 (ノード数 3)

非対称密行列向けの LU 分解の実測を行なった。分解のアルゴリズムはいわゆる外積形式ガウス法を採用した。行列は列方向に分割して各ノードへサイクリックに割り当て、配列データに対する書き込みは全てローカルに保持するデータに対してなされるようにした。行方向のデータを全く見ない通常の部分ピボッティングを行ない、行交換はその都度行なった。データ転送は、分解列

NII-Electronic Library Service

Information Processing Society of Japan

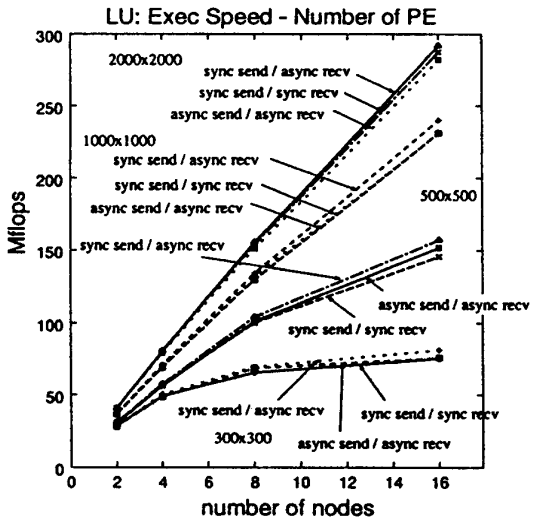


図 10: ノード数と LU 分解実行速度

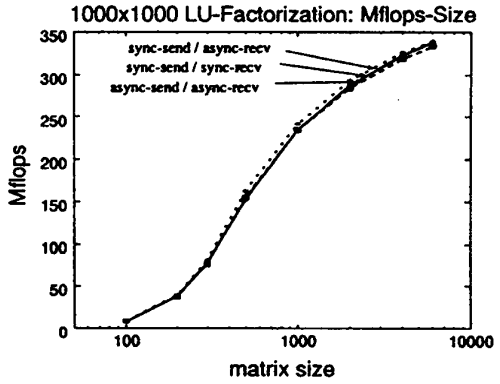


図 11: 行列サイズと LU 分解実行速度

を保持しているノードがその他のノードにブロードキャストすることで行なわれる。並列実行の様子を図 9に、実行結果を図 10に示す。なお、演算は全て倍精度実数で行なった。使用したデータは以下の通りである。

$$a_{ij} = \sqrt{\frac{2}{n+1}} \sin \frac{\pi ij}{n+1}$$

ノード間でのデータ交換は、送受信とも同期・送信のみ同期・送受信とも非同期の三通りで行なった。図 10によると、行列積の場合と同様、同期送信・非同期受信のパターンが最も高速に実行されている。

非同期通信命令は、演算と通信をオーバラップさせるのに有効であるが、演算速度に影響を及ぼす場合に注意しなくてはならない。

5 まとめ

疎結合並列計算機 Intel Paragon について、その通信性能を中心に、実測に基づいた評価を行なった。

非同期転送命令を用いることにより演算と通信の文字通りのオーバラップが可能であるが、非同期データ送信は演算にまで影響を及ぼす。このオーバヘッドは行列積のような浮動小数点演算中心の単純なアプリケーションでは演算量の増加にともない無視できるようになるが、6000 次元の LU 分解でも依然として 2 % 程度の差を生じさせる。よりデータ転送回数の多いアプリケーションや負荷分散が完全には行なえない場合、あるいはノード間結合網のトラフィックが均質でない場合などでは、安易な非同期転送はおそらくより大きく総実行時間に影響を及ぼすであろう。

これと比較すると効果は微小であるが、データ受信命令の発行を待って送信を開始し、データ転送時のシステムバッファの介在を避けることで、パフォーマンスを向上させることができる。

本稿では非常に単純な場合についての性能についてのみ触れた。多くのアプリケーションでは、複雑なデータ配置による非対称な通信パターンや動的な要素を多分に含んだ演算パターンが存在する。また構成の大きな並列計算機では、たとえ均質・対称的な演算が行なわれる場合でも、ノード間結合網のトラフィックにばらつきが生じる。こうした状況を想定した（定量的な）評価が望まれる。これらについては今後の課題としたい。

また、ノード間通信速度や行列積の演算速度の測定の結果、パフォーマンスに明白な不連続性が見い出された。これはキャッシュの影響が大きいと思われるが、細かい挙動についてはまだ不明である。今後調査するとともに、キャッシュの影響を考慮した並列プログラミングに対する考察が必要であろう。

謝辞

本稿をまとめるにあたり御助言御指導頂いた、広島市立大学弘中哲夫助教授、京都大学上原哲太郎助手に感謝致します。

参考文献

[1] Jack J. Dongarra: “Performance of Various Computers Using Standard Linear Equations Software”, Rep. CS-89-85, Comput. Sci. Dep. Univ. Tennessee, 1995.

[2] Intel SSD: “Paragon System User’s Guide”, 1995.

[3] Intel SSD: “Paragon System C Calls Reference Manual”, 1995.

[4] 津田孝夫: “数値処理プログラミング”, 岩波書店, 1988.

[5] 小国力 編著, 村田健郎, 三好俊郎, ドンガラ,J.J., 長谷川秀彦 著: “行列計算ソフトウェア”, 丸善, 1991.

[6] Rob H. Bisseling et al. : ”Parallel LU Decomposition on a Transputer Network”, Parallel Computing Shell Conference Proceedings, 1988.

NII-Electronic Library Service