ハイパフォーマンス
コンピューティング 52－7
（1994．7．22）

データフローに基づくスプレッドシート処理の並列化

南 宣明* 横田治夫

北陸先端科学技術大学院大学情報科学研究科

あらまし スプレッドシートは計算機の非専門家のユーザーでも手軽に使えるデータ処理ソフトウエアであり、比較的小規模な非定型業務のデータ処理や意思決定支援ツールとして広く使われている。しかしスプレッドシートを大規模なデータ処理に使おうとすると、セルの数に制限があったりデータ量や関数定義を多くすると低速になるという欠点に直面する。本研究ではスプレッドシート処理を並列化することにより、処理の高速化と大規模化を目指す。スプレッドシートのデータのマルチプロセッサへのマッピング, セルデータのデータ構造, データフローによる処理の制御・同期化について検討を行う。さらに、nCUBE2 上に実験システムを構築しその性能評価を行い、並列効果を確認する。

和文キーワード データフロー, スプレッドシート, 並列効果, メッセージパッシング

Parallel Spreadsheet Processing Based on Dataflows

Yoshiaki MINAMI, Haruo YOKOTA

School of Information Science
Japan Advanced Institute of Science and Technology,Hokuriku

Abstract Spreadsheets are data processing software which can be easily used by non-expert users for processing relatively small scaled non-fixed forms or for implementing decision support tools. When the spreadsheets are used for processing large scaled data, there exists some drawbacks : limit on quantities of data or formulas to be stored. The processing speed extremely slows down when the number of data or formula definitions become large. This paper considers parallel spreadsheets processing to make it be able to handle the great number of definitions and to improve its processing speed. We study data mapping on multiprocessors, data structure of each spreadsheet cell, process control and synchronization based on dataflows. We implement an experimental system on an nCUBE2, and evaluate the system using some examples.

英文 key words Dataflow,Spreadsheet,Parallel effectiveness, Message-passing

*現在はNTT インテリジェントテクノロジ株式会社に所属。

1 はじめに

スプレッドシートは計算機の非専門家のユーザーでも手軽に使えるデータ処理ソフトウエアであり、比較的小規模な非定型業務のデータ処理や意思決定支援ツールとして広く使われている。しかしスプレッドシートを大規模なデータ処理に使おうとすると、セルの数に制限があったりデータ量や関数定義を多くすると低速になるという欠点に直面する。これを克服するにはスプレッドシート処理の高速化や大規模化が必要になるが、これらを並列処理化によって行おうという研究はこれ迄殆んど行われていない。

現在データ量の多い処理にはリレーショナルデータベースが使われているが、その使用には専門家の介在が必要となる。さらにリレーショナルデータベースのアクセスツールであるQBEのユーザーインタフェイスはスプレッドシートと同じく表形式であり計算機の非専門家のユーザーにも広く活用されているが、QBEではスプレッドシートのようなセル単位でデータ格納や関数の処理を行うといった柔軟な処理はできない。

他方、データベース中にデータとして関数や手続きを格納するオブジェクト指向データベースが研究されているが、その使用に際してはスプレッドシートに比べると必ずしもユーザが手軽に使えるインターフェイスを備えているとは限らない。

これに対しスプレッドシート処理を並列化することにより、大量データに対して充分高速化されれば、ユーザー自身が直接データ処理を行うことができる。さらに、利用者が特に並列処理を意識することなく、自然に並列効果を得ることも可能となる。

以下、本稿ではスプレッドシート処理の大規模化・高速化のための並列処理の手法を検討する。具体的には、今後マルチプロセッサの主流となるであろうメッセージパッシングマシンを対象とした、データのマルチプロセッサへのマッピング, セルのデータ構造, 処理の制御, 同期化手法を提案する。

さらに、これらに基づき非同期のメッセージパッシング型MIMDマシンであるnCUBE2に実験システムを作成し、いくつかの性能評価を行う。

2 並列化の方法

2.1 ワークシート上のセルデータのマルチプロセッサへのマッピング

ワークシートのセルデータをマルチプロセッサにマッピングする方法としてハッシュ関数や値による分散が考えられるが、

- スプレッドシートはレコードキーが存在しない
- セルデータには即値の他に計算式・関数が格納される

ため、適当なハッシュ関数の適用や順序付けが困難であることから、スプレッドシート処理には不向きである。そこで我々はワークシート上のセルデータをラウンドロビンでマルチプロセッサにマッピングする方法を採用する。

ラウンドロビンでマッピングする場合、行単位, 列単位, 個々のセル単位に行う方法が考えられる。一般に、

1. スプレッドシートは列方向より行方向により拡大する傾向があり、行単位でラウンドロビンを行う方が処理の均一な分散が望める
2. スプレッドシートのワークシートに入力される計算式・関数は、列を跨って計算するものの方が行を跨って計算するものより複雑になる傾向があり、行単位のラウンドロビンの方が列単位よりも並列効果が期待できる
3. セル単位のラウンドロビンだと計算式・関数のセルの計算の際のメッセージパッシングの量が増えるため、行単位のラウンドロビンの方が計算速度が期待できる

ため、行単位のラウンドロビンを採用する。図1にそのイメージを示す。

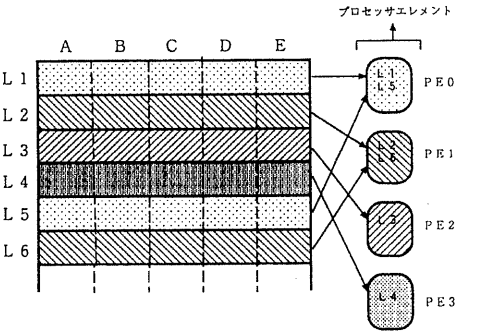

図 1: ワークシート上のセルデータのプロセッサへのマッピング

2.2 各プロセッサ内でのセルのデータ構造

スプレッドシート処理では再計算のための行名・列名によるセルへのアクセス頻度が大きいため、行名・列名によるセルデータへのランダムアクセスを効率良く行う必要がある。そのため各プロセッサ内では、行名, 列名で高速にアクセスする手法を取り入れる必要がある。我々は格納する行名をk=4のB-木の形式で保持し、列名は各々対応する行名のB-木のリーフの下にやはりk=4のB-木の形式で保持することにした。

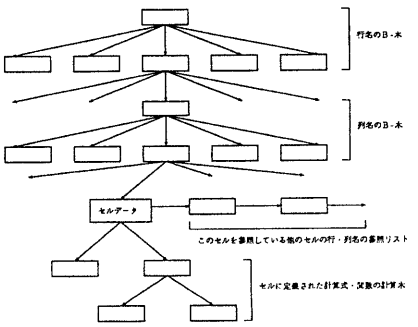

図 2: 全体的なセルデータのデータ構造

実際のセルデータは該当する列名のB-木のリーフの下に置く。セルデータが関数・計算値の場合は、関数・計算式の計算を行うための二分木の形式の計算木に展開してセルデータの下に置く。セルに計算式・関数が定義されていないものは計算式・関数の計算木を保持しない。

以上のような方針による各プロセッサで保持するセルデータの全体的なデータ構造を図2に示す。

2.3 メッセージパッシングとデータフローに基づく計算式・関数処理の並列化・同期化

2.3.1 結果の波及

スプレッドシート処理では、ワークシート上のあるセルのデータを受け取ったなら、そのデータを格納・計算するとともに、そのセルを参照している他のセルにその結果を波及させなければならない。さらに波及を受けたセルを参照しているその先のセルにもその結果を波及させる必要がある。従ってスプレッドシート処理を行うためには以下の機能が必要となる。

1. ワークシートのセルのデータの受け取り・格納
2. 計算式・関数が定義されたセルの計算
3. 他セル間の参照関係のメンテナンス

このため、セルが他のセルから参照されている場合は、セルデータの下に自分を参照しているセルの行名・列名を参照リストとして持つことにする。さらに、スプレッドシート処理をマルチプロセッサ上で行うためには、各プロセッサは以下の機能を持たなければならない。

1. 計算式・関数が定義されたセルが他プロセッサ上のセルを参照しているなら、他のプロセッサ上の参照先のセルの参照リストをメンテナンスするためのメッセージパッシング
2. 他プロセッサ上のセルから参照されているセルの値が変更になった際の、参照元への波及計算のためのメッセージパッシング

これは、そのままプロセッサ間のデータフローにマッピングすることができる。各々のプロセッサが独立して並列に処理することにより、メッセージパッシングを用いたデータフローによる効率の良い並列処理が実現できる。

2.3.2 通信と計算

我々が対象とするnCUBE2は、メモリ容量の制限や通信バッファの確保等の関係から、1プロセッサ上に複数のプロセスを実行させるのは困難である。シングルプロセスで実現する場合、メッセージのreceive後にそのメッセージに関わる全ての計算処理が終了するまで次のメッセージのreceiveを行わないという方法には問題がある。そのため、メッセージのreceiveとメッセージに関する計算処理を交互に行うことにする。ただし、receiveすべきメッセージがない場合は、次のメッセージをwaitすることなく既に受け取ったメッセージに関する計算処理を行う。

これを行うために、個々のプロセッサに計算キューを用意する。メッセージパッシングにより格納・計算するワークシートのセルデータをreceiveしたならセルデータの格納及び参照関係の調整を行った後、計算キューにreceiveしたセルデータのセル名をappendする。同じくメッセージパッシングにより波及計算のメッセージをreceiveしたなら波及データの該当セルデータへの更新を行った後、該当セルデータのセル名を計算キューにappendする。

receiveの処理が終り計算キューにデータが格納されているなら、計算キューからセル名を1つpopし計算を行う。次にそのセルの参照リストを検査し他プロセッサ上のセルから参照されているなら該当プロセッサに波及計算のメッセージをsend、自プロセッサ上のセルから参照されているなら、そのセル名を計算キューにappendする。計算キューが空またはこの一連の処理が終了したならメッセージパッシングによるメッセージのreceiveを行う。その際、receiveすべきメッセージがなければ直ちに計算キューからセル名を1つpopし同様の処理を行う。

上のような波及計算を実現するため、他セルを参照する計算式・関数の定義されたワークシートのセルデータを格納・計算するメッセージをreceiveしたなら、参照先のセルの参照リストを調整しなければならない。すなわち新たに参照先のセルが生じた場合は、参照先のセルの参照リストに該当セルデータのセル名を追加しなければならない。逆に、既に格納されているセルデータと比べて参照先のセルが減った場合、それ迄参照していたセルの参照リストから該当セルデータのセル名を削除しなければならない。この際、参照先のセルが他プロセッサ上のものなら、該当プロセッサに該当するセルの参照リストに追加・削除を行わせるメッセージをsendする。他方、参照リストに追加・削除を行わせるメッセージをreceiveしたプロセッサはそれに対応した処理を行った後、追加メッセージならばメッセージをsendしてきたプロセッサに波及計算を行わせる波及データのメッセージをsendする。このことにより、格納されるセルデータは計算に必要な参照先の値を得ることができる。

2.3.3 プロセッサ間の同期化

同期化という点から見ると、ユーザーがワークシートに入力したセルデータに関する関連プロセッサの処理が全て終ったことを検知できないなと、ユーザーが次にワークシートにセルデータを入力できるタイミングがわからなくなる。最初にワークシート上のセルデータを受け取ったプロセッサは必要に応じ他のプロセッサに波及計算を指示する。波及計算を指示されたプロセッサも必要に応じて他のプロセッサに波及計算を指示する。従って最初にワークシート上のセルデータを受け取ったプロセッサは、そのセルデータを受け取ったことにより直接・間接に波及計算を指示されたプロセッサの処理が全て終了したことを検知する機能を持たなければならない。

これについては、自プロセッサでの計算処理が全て終了し、かつ波及計算を指示した他のプロセッサ全てからそのプロセッサの処理が終了したメッセージを受け取った時点、または他のプロセッサに波及計算を一切指示していない状態になった場合に、自プロセッサにメッセージをsendしたプロセッサ全てに自プロセッサの処理終了のメッセージをsendすることで行う(図3)。

これを行うため、個々のプロセッサに送信先PE(プロセッサエレメント)キューと受信元PEキューを用意する。

個々のプロセッサがセルデータの格納・計算のメッセージまたは波及計算のメッセージを receive したなら、そのメッセージを send したプロセッサ名を受信元 PE キューに append する。また、個々のプロセッサの計算処理で他のプロセッサに波及計算のメッセージを send したなら、send 先のプロセッサ名を送信先 PE キューに append する。

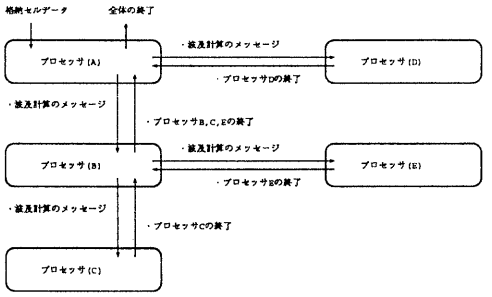

図 3: 波及計算・同期化のメッセージの流れ

他のプロセッサへ波及計算のメッセージを send を行わず送信先 PE キューが空のプロセッサ、つまり波及計算のボトムに当るプロセッサは自プロセッサの計算キューが空になった時点、すなわち自プロセッサでの計算処理が全て終了した時点で受信元 PE キューの全てのプロセッサに自プロセッサの処理終了メッセージを send する。

他のプロセッサへ波及計算のメッセージを send を行ったプロセッサが他プロセッサからプロセッサの処理終了メッセージを receive したなら、送信先 PE キューから該当プロセッサ名を delete する。そのプロセッサの送信先 PE キューが空かつ計算キューになった時点で受信元 PE キューの全てのプロセッサにそのプロセッサの処理終了メッセージを send する。

最終的にワークシートのセルデータの格納・計算メッセージを受け取ったプロセッサがプロセッサの処理終了メッセージを send すれば、そのセルデータの格納・計算に関する関連プロセッサの処理は全て終了となる。

ただしこの方法のみでは、あるプロセッサのセルの参照先が他のプロセッサ経由で自プロセッサにまで及ぶ場合は依存関係のループができてしまう。これを回避するため、各プロセッサでの波及計算指示によるセルの計算が終了するたびに計算キューをチェックし、他に同じプロセッサからの他に波及計算指示がないならそのプロセッサのみに自プロセッサの処理終了メッセージを send する。

これらのことを行うための個々のプロセッサ内の構成を図 4に示す。

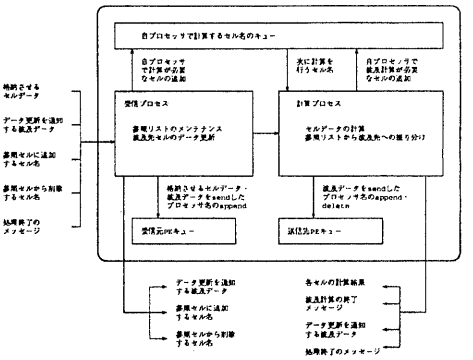

図 4: 個々のプロセッサ内でのプロセス・メッセージ・各キューの関係

3 性能評価

256 プロセッサ構成の非同期のメッセージパッシング型 MIMD コンピュータである nCUBE2 上に実験システムを作成し、表 1 の内容について、計算処理時間に関するいくつかの性能評価を行った。

実験システムのプログラムはユーザーとのコミュニケーションを行う 1 つの HOST プロセッサのプロセスと、実際に並列にスプレッドシートの計算を行う他のプロセッサのプロセスから構成する。nCUBE2 は任意の 2 のベキ乗のプロセッサを利用できる。従って、並列処理部分で使用できるプロセッサ数は 1,3,7,15,31,63,127,255 のいずれかになる。

3.1 同一行内で計算が完結する場合

先ず全ての計算が同一行内で完結するような計算を行わせ、プロセッサスケーラビリティおよびデータスケーラビリティの評価を行った。具体的には各行に図 5 の形式のワークシート (n は行名) を設定し、それを行数・プロセッサ数を変化させながら実行させた。

この処理でのワークシートの行数が 200 行,400 行,800 行の場合の、各プロセッサ数での処理時間の変化を図 6 に、プロセッサ数が 1 の時の処理速度を 1 とした時の各プロセッサ数での相対的な処理時間の変化を図 7に示す。これらのグラフの横軸はプロセッサノード数を、縦軸は処理時間を表す。

表 1: 評価の内容と処理の特徴

No.	評価内容	処理の特徴
1	同一行内で計算が完結する単純な計算の場合	メッセージパッシングが殆んど発生しない
2	行を跨った計算を行う場合	ある行が他の行を 1 行だけ参照する
3	横展開の行列の乗算の場合	メッセージパッシングが 1 行につき複数発生
4	縦展開の行列の乗算の場合	同じ計算でもメッセージパッシングは上の例よりは多い
5	ガウス消去法の場合	行の順序と計算の順序が不一致 メッセージパッシングが多数かつ複雑

A	B	C	D	E	F	G	H	I	J	K	L	M
即値	即値	即値	(An+Bn+Cn)/3	即値	即値	即値	(En+Fn+Gn)/3	即値	即値	即値	(In+Jn+Kn)/3	Dn*Hn*Ln

図 5: 同一行内で計算が完了するワークシート

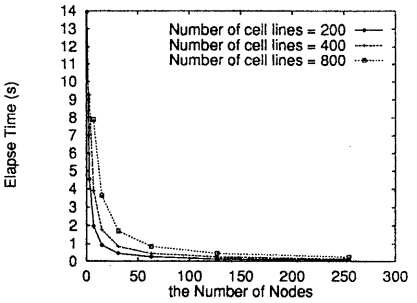

図 6: 同一行内計算のみのワークシートの処理時間の変化

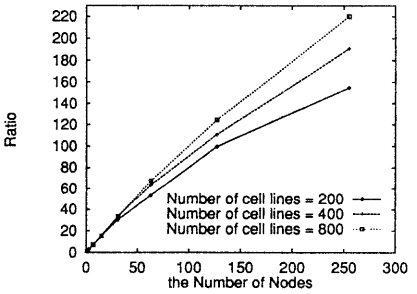

図 7: 同一行内計算のみのワークシートの処理時間の相対的変化

この処理は行を跨る計算が存在しないため、マルチプロセッサ上で処理を行っても波及計算のためのメッセージパッシングは発生しない例である。この処理ではプロセッサ数が増えるに従って処理速度の向上が見られる。この内、データ数が少い内はプロセッサが増えるに従って速度の向上の比率が低減しているが、データ数が多くなるほど処理速度はリニアに近くなり、並列効果も大きくなる。これはデータ数が多い場合、HOST プロセッサと各プロセッサとの間のメッセージパッシングの処理時間に占める割合が相対的に低くなり、並列効果がはっきり 現れたものと見られる。

次に、この処理にてデータ数の増減による処理時間への影響を調べるため、プロセッサ数が 63 の場合での各データ数での処理時間をグラフ化したものを図 8に示す。このグラフの縦軸は処理時間 (秒) を、横軸はデータ数を表す。

図 8で、データ数の増加に伴う実行時間の増加に多少ばらつきがあるのは、行名・列名の格納に B-木を使用しており、木の分割によって検索時間が直線的には増加しないためと見られる。

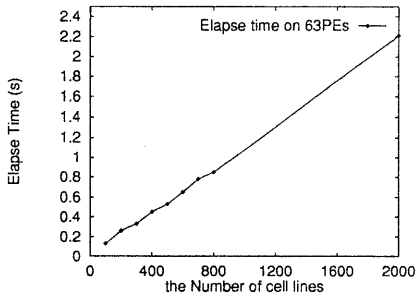

図 8: 63 プロセッサ上での計算処理時間の変化

3.2 行を跨った計算を行う場合

次に、行を跨って計算を行うようなワークシートの処理を行い、プロセッサスケーラビリティおよびデータスケーラビリティの評価を行った。具体的には各行に下記の形式のセルデータ (n は行名) を設定し、それを行数・プロセッサノード数を変化させながら実行させた。

	A	B	C	D
1	即値	即値	即値	(A1+B1)/2 + Cn
2	即値	即値	即値	(A2+B2)/2 + C1
n	即値	即値	即値	(An+Bn)/2 + Cn-1

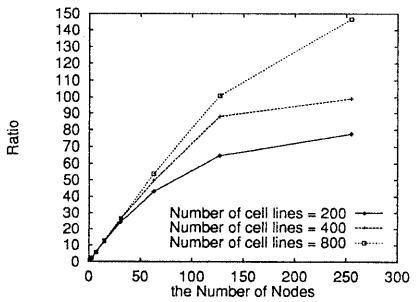

図 9: 行を跨った計算を行うワークシートの処理時間の相対的変化

この処理でのワークシートの行数が 200 行,400 行,800 行の各々の場合でのプロセッサ数が 1 の時の処理時間を

1とした時の各プロセッサ数での相対的な処理時間の変化を図9に示す。

3.1の処理とは異なり、この処理ではマルチプロセッサ上では波及計算のためのメッセージパッシングが発生する。従ってプロセッサ数の増加に伴う並列効果の増加は3.1のものよりは小さくなっている。ただし3.1と同じく、データ数が大きくなるほど並列効果が高くなる傾向が見られる。

3.3 縦展開の行列の乗算の場合

ワークシート上のある行のセルデータが複数の他の行を参照して計算を行う例として、データ・計算式を縦方向に展開した形式による正方行列同士の乗算を次数・プロセッサ数を変化させながら実行させた。

	A	B	C	
0	即値	即値	即値	
1	即値	即値	即値	乗算の左辺
2	即値	即値	即値	
3	即値	即値	即値	
4	即値	即値	即値	乗算の右辺
5	即値	即値	即値	
6	A0*A3+B0*A4+C0*A5	A0*B3+B0*B4+C0*B5	A0*C3+B0*C4+C0*C5	
7	A1*A3+B1*A4+C1*A5	A1*B3+B1*B4+C1*B5	A1*C3+B1*C4+C1*C5	乗算の結果
8	A2*A3+B2*A4+C2*A5	A2*B3+B2*B4+C2*B5	A2*C3+B2*C4+C2*C5	

図 10: 3 × 3 の行列の縦方向の乗算のマッピング

行列の乗算を縦方向に展開した場合のワークシート上のセルデータのマッピングを 3 × 3 の正方行列の乗算を例に図10に示す。

この処理でのワークシート上の正方行列が10次,15次および20次の場合の、各プロセッサ数での処理時間の変化を図11に、プロセッサ数が1の時の処理速度を1とした時の各プロセッサ数での相対的な処理時間の変化を図12に示した。

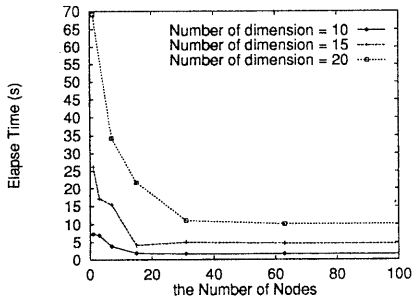

図 11: 行列の縦展開での乗算の処理時間の変化

この内10次のグラフを見ると、プロセッサ数を3から7及び7から15に増加させた場合に最も高い速度向上比を示している。これに対し、プロセッサ数を1から3及び15から31にした場合は行列要素の計算の負荷分散の効果に対してメッセージパッシングの増加による抑制が働き、速度の向上は鈍っている。31プロセッサ以降は実質的に動作するプロセッサ数が同じであるので、計算速度は不変である。

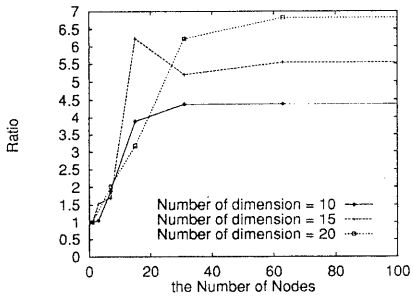

図 12: 行列の縦展開での乗算の処理時間の相対的変化

15次の場合は15プロセッサで最高の処理速度になり、31プロセッサでは却って処理速度が低下している。15プロセッサの場合は、1プロセッサ毎に格納されるセルデータが即値の行が2行、計算を行う行が1行と平順化され、かつ計算のためのメッセージパッシングの量も全プロセッサを通じて同一なため、全体的な処理のバランスが取れている。31プロセッサの場合はプロセッサが即値の行が1行かつ計算の行が1行のものが14プロセッサ、即値の行が1行のみのものが16プロセッサ、計算の行が1行のみのものが1プロセッサとアンバランスになる。その結果メッセージパッシングによるオーバーヘッドが個々のプロセッサの計算量の低減効果を上回ったと見られる。

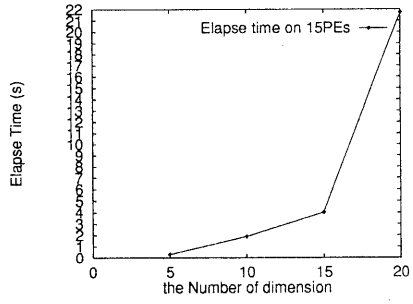

図 13: 15プロセッサ上での処理時間の変化

次に、この処理にてデータ数の増減による処理時間への影響を調べるため、プロセッサ数が15の場合での正方行列の各次数での処理時間をグラフ化したものを図13に示す。

このグラフでは正方行列の次数を 15 から 20 にした場合の性能低下の度合が次数が5～15 の場合の性能低下の度合より急になっていることを示している。これは次数が増加することにより行列要素の計算で参照するセルデータのメッセージパッシングの増加に由来すると見られる。

3.4 横展開の行列の乗算の場合

次にワークシート上の行列のデータ・計算式を横方向に展開した形式による正方行列同士の乗算を次数・プロセッサ数を変化させながら実行させ、計算処理時間を計測した。

行列の乗算を横方向に展開した場合のワークシート上のセルデータのマッピングを 3 × 3 の正方行列の乗算を例に図 14 に示す。

この処理でのワークシート上の正方行列が 10 次,15 次,20 次の各々の場合での、各プロセッサ数での処理時間の変化を図 15に、プロセッサ数が 1 の時の処理時間を 1 とした時の各プロセッサ数での相対的な処理時間の変化を図 16に示す。

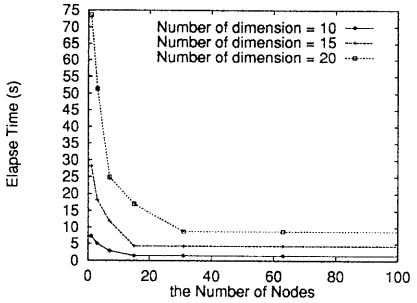

図 15: 行列の横展開での乗算の処理時間の変化

図 15を見ると、マルチプロセッサ上では 3.3の行列の横展開での処理時間に比べ処理時間が速くなっている。さらに図 16を見ると 3.3に比べ、プロセッサ数を増加させた際の処理時間の改善が一様に大きくなっている。これは行列要素の計算の際に参照するセルデータが同一プロセッサに存在する比率が高く、処理に占めるメッセージパッシングの割合が低いことから自明である。プロセッサ数が次数以上の場合は実質的に動作するプロセッサ数が同じであるので、処理時間は不変である。

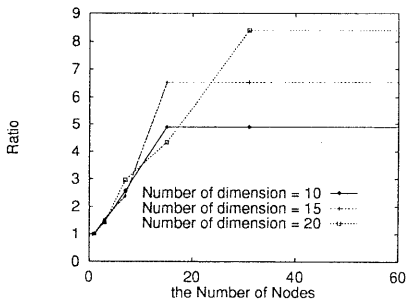

図 16: 行列の横展開での乗算の処理時間の相対的変化

3.5 ガウス消去法の場合

最後にワークシート上で連立方程式をガウス消去法を用いて解く処理を次数・プロセッサ数を変化させながら実行させ、計算処理時間を計測した。例として下の 3 元の連立方程式をワークシート上のセルデータにマッピングしたものを図 17に示す。

$$\begin{aligned} 2X + 5Y + 7Z &= 23 \\ 4X + 13Y + 20Z &= 58 \\ 8X + 29Y + 50Z &= 132 \end{aligned}$$

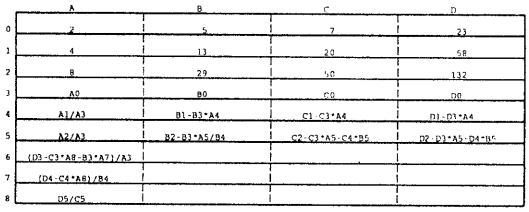

図 17: 3 元の連立方程式のガウス消去法のマッピング

図 17のワークシートでリージョン A0～D2 が元の連立方程式の部分であり、X の解はセル A6 の、Y の解はセル A7 の、Z の解はセル A8 の計算式で求められる。

この処理での 4 元,5 元,6 元の連立方程式の各々場合での、各プロセッサ数での処理時間の変化を図 18に、プロセッサ数が 1 の時の処理時間を 1 とした時の各プロセッサノード数での相対的な処理時間の変化を図 19に示す。

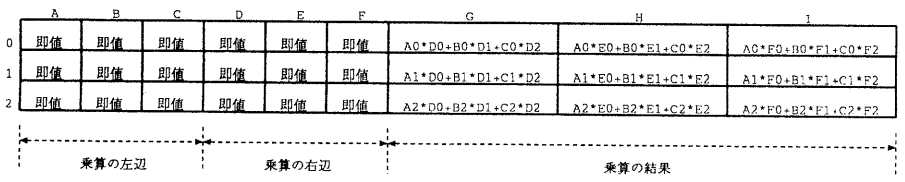

図 14： 3×3の行列の横方向の乗算のマッピング

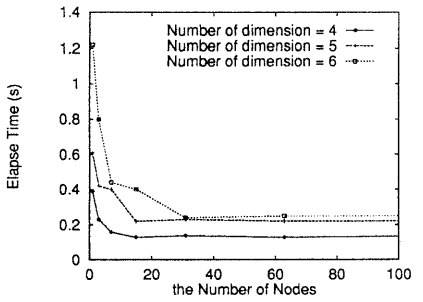

図 18: ガウス消去法の処理時間の変化

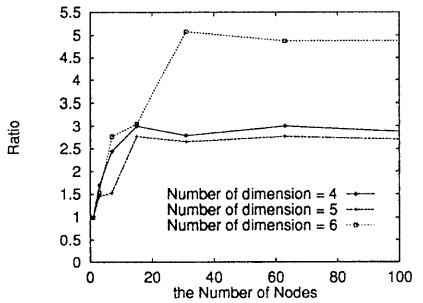

図 19: ガウス消去法の処理時間の相対的変化

図19を見るとプロセッサ数の増加にともなう処理速度の向上は一様ではない。これはガウス消去法では各行・各列の計算量が一様ではないため、行のプロセッサのマッピングによってプロセッサ毎の計算の負荷やメッセージパッシングの状況が不均一になるためと見られる。

全般的には元の数が多いほど並列効果が高くなっている。プロセッサ数がワークシートの行数以上の場合の処理時間の差異は高々0.01秒であり、実質的に計算速度は同一である。

3.6 全体的な評価

以上行ってきた性能評価をまとめると、以下のことが言える。

大規模データの処理という点から見ると、データ数が多いほどプロセッサ数を増やすことによる並列効果が大きくなり、この方法が大規模データ処理に適していることがわかった。

絶対的な処理速度という点から見れば、データ数が同じであればプロセッサの増加により並列度を上げることで処理速度は改善される。しかし、問題によってはプロセッサ間のメッセージパッシングが増大するため速度上昇比は並列度が上がるほど低くなる場合がある。

そのため並列度の効果をさらに高めるには、

1. メッセージパッシングの際には、いくつかのメッセージをまとめて send するようにし、全体的なメッセージパッシングの回数を減らす
2. メッセージの衝突を避けるため、各プロセッサのメッセージパッシングのタイミングが重ならないようにする

等の対策が必要となる。

4 おわりに

本研究ではスプレッドシート処理の高速化・大規模化という目的に対し、データフローに基づく並列化による実現方法とそれに適応したデータ構造を提案した。スプレッドシートのセル間の依存関係をそのままデータフローにマッピングすることにより各セルの処理をマルチプロセッサ上で並列に行うことができる。このことにより、使用するプロセッサ数を増やすことによる処理時間の短縮、扱うことが可能なデータ量の拡大が可能となる。

実際にこの方法による実験システムを非同期のメッセージパッシング型 MIMD マシンである nCUBE2 上に作成し、いくつかの性能評価を行った。

その結果、メッセージパッシングの増加に起因するオーバヘッドによる並列効果の低減が見られる場合もあったが、全体的にはデータの規模が大きくなるほど並列効果が増大し、この方法がスプレッドシート処理の大規模化・高速化に対して有効であることを確認した。さらに、セルデータの格納順序は一切考慮しなくても処理可能なため、複数ディスクへの格納/読み出しが容易に行えると思われる。

今後の課題としては、

- メッセージパッシングの頻度の削減・タイミングの最適化
- 複数ディスクを対象とした場合の最適な格納方式

等の検討が大切であると思われる。

参考文献

[1] 南 宜明, スプレッドシート処理の並列化, 修士論文, 北陸先端科学技術大学院大学情報科学研究科,1994.

[2] Lee Ben, Hybrid scheme for processing data structures in data-flow enviroment, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, Vol.3, No.1, pp83–96, 1992.

[3] 渋谷正弘, 田中 譲, スプレッドシートを介した論理型プログラミング, 情報処理学会論文誌, Vol.30, No.6, pp709–718, 1989.

[4] 金井直樹, 福永光一, 横井伸司, インテリジェント・スプレッド・シートとオブジェクト指向パラダイム, WOOC'87,1987.