計算機アーキテクチャ　128－4
ハイパフォーマンス
コンピューティング　70－4
（１９９８．３．５）

通信ブロックの軽減を考慮した大規模行列における分散処理システムの 設計

松村 博光　大鎌 広　藤原 祥隆
北見工業大学

WS クラスタで大規模行列計算を高速に行なうための分散処理システムを提案している。分散処理システムの形態はマルチサーバ・１クライアント型であり、クライアント・サーバ間、及びサーバ・サーバ間において行列データの通信を行なっている。このような分散処理を行なう場合、通信の待ち時間 (ブロック) による処理効率の低下が問題となる。本稿では、送信の子プロセス化やメッセージのキャッシュを用いることにより、通信ブロックの影響を抑えた分散処理システムの設計を示している。

Construction of a distributed computing system for large-scale matrices intended for reduction of communication blocks

Hiromitsu MATSUMURA　Hiroshi OHKAMA　Yoshitaka FUJIWARA
Kitami Institute of Technology

A distributed computing system is suggested for high performance computation for large-scale matrix using networked workstation clusters. The proposed distributed computing system is multi-server 1-client model. The servers are communicating matrix data to the client and the other servers.

A problem in distributed computations is performance deterioration by waiting communication(block). In this paper, the distributed computing system that reduces the influence of communication block by using a child process to send the data and caching the message is proposed.

1　はじめに

ネットワークに接続された複数の計算機で分散処理を行なうためのソフトウェアとして、PVM[1] や MPI[2] が知られている。これらは主に計算機間の通信部分のインタフェイスを提供するものであり、異機種間でのデータ型変換を伴う通信が可能であるなど、各種のプログラムスタイルに適応できるようになっている。

しかしこれらを用いてプログラムを作成する際、本来のアルゴリズムとは関係ない通信部分の記述まで行なう必要があり、プログラマの負担が大きい。

本研究で提案している分散処理システムでは、同機種による WS クラスタを用い、用途も行列計算に限定している。これによって通信時のデータ変換による通信速度の低下を抑え、通信部分の記述をできるだけ行なわずに済むようなユーザインタフェイスを実現している。

またこのような分散処理を行なう場合、通信の待ち状態 (ブロック) により、処理効率が下がることが１つの問題となっている [1]。

そこで、通信ブロックの影響を軽減するような分散処理システムの設計を行なう。

2 システム概要

本研究における分散処理システムの形態は、マルチサーバ・1クライアントとなっている (図1)。システムは行列計算サーバプログラムと C++による分散行列クラスライブラリで構成されている。

ユーザはまず分散行列クラスライブラリを用いて行列計算のプログラムを作成する (これをユーザプログラムとする)。このユーザプログラムが実行される計算機を行列計算クライアントと呼ぶ。それに対して、行列計算サーバプログラムを動作させる計算機を行列計算サーバと呼ぶ。

行列計算クライアントでユーザプログラムを実行すると、ユーザプログラムはライブラリによって行列計算サーバへの命令列に変換され、各行列計算サーバに送信される。この時送られる命令には大きく分けて

行列データ通信命令 行列のデータを送信する

計算命令 加算や乗算など、計算を行なうよう指示する

特殊命令 計算の終了などの指示を出す。

がある。

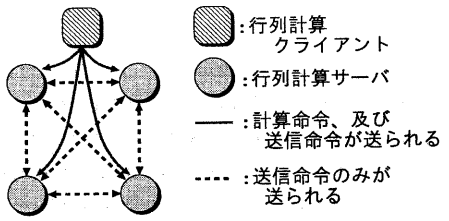

図 1: 分散処理システムの構成

行列計算サーバでは、ユーザプログラムが実行される前に行列計算サーバプログラムを起動しておく。後はクライアントからの命令を受信して、実際に行列計算を行なっていく。サーバプログラムの処理は基本的に以下のようになっている。

1. 行列計算クライアント、及び他のサーバからの命令を監視する
2. 命令受信
3. 命令デコード
4. 命令が終了命令なら処理を終了
5. 命令実行 (行列計算、行列データ送信)
6. 1. に戻る

1. にあるように、命令を発行するのは行列計算クライアントだけでなく、サーバ側でも行なう。ただし、サーバ側が発行する命令は行列データ送信命令だけであり、計算命令や特殊命令は発行しない。

計算機間の通信には TCP を使用している。このため実際に通信を行なう前にコネクションを確立する必要がある。

そこで行列計算サーバプログラムでは、起動時にサーバに使用する計算機の台数とホスト名の記述された設定ファイルを読み込む。ファイルから得られた情報を使って、行列計算サーバ間のコネクションを確立する。サーバ間のコネクションが全て確立したら、ユーザプログラムの実行を待つ。

ユーザプログラムでも実行時に先ほどの設定ファイルを読み込み、各行列計算サーバに対してコネクション確立要求を出す。ここでクライアント-サーバ間のコネクションが確立される。

またこのような設定ファイルを用いることで、後から計算機の台数を変更したり、使用する計算を変更することが容易になるといった利点もある。

このように、1台の行列計算クライアントが各行列計算サーバに行列データや計算命令を送信し、それを受信した行列計算サーバが命令を実行することで、分散処理が成り立っている。

2.1 システム使用の例

ここで、実際にユーザが行列計算の分散処理を行なうために必要な手順を示す。

1. ユーザプログラムを記述する。
   以下にユーザプログラムの簡単な例を示す。

```
#include "DMatrix.h"
void main(void)
{
    DMatrix a("data1.dat"),
            b("data2.dat"), c;
    c = a * b;
    c.Print("kekka.dat");
}
```

2. ユーザプログラムをコンパイルし、分散行列クラスライブラリとリンクして実行ファイルを作成する。

3. 行列計算サーバの設定ファイルを記述する。例えば4台で分散処理する場合、次のようになる。

```
4 # 使用する計算機の台数
# 以下にホスト名を記述する
pro101
pro102
pro103
pro104
```

4. 各行列計算サーバで行列計算サーバプログラムを起動する。

5. 2. で作成した実行ファイルを実行する。

このように、すでに分散行列クラスライブラリに実装されている演算を用いてユーザプログラムを作成することは容易である。

3 行列データの管理

分散処理システムでの行列データの管理は、ユーザプログラム内で使用される行列データを行列計算サーバ数で分割した行列 (以後分割された行列のことを小行列と呼ぶ) を単位に行なわれる。

行列計算クライアントでは、ユーザプログラム内で分散行列クラスのオブジェクトが生成されると直ちにその行列データを分割して小行列を作成する。この際、各小行列に対して全プログラム実行時間においてシステム全体でユニークな ID (行列 ID) が与えられる。

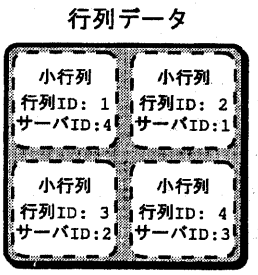

図 2: 小行列による行列データ管理

この行列 ID を用いることにより、行列計算クライアント、サーバ双方でデータ間の依存関係を明確にでき、データの格納場所を統一的に管理することが可能となっている。

3.1 行列計算クライアントでの行列データ管理

クライアント側では、小行列を作成するとすぐに各行列計算サーバに送信する。送信後はユーザプログラム内の行列データと小行列の対応関係を記録して、行列データは解放してしまう。つまり行列データは基本的にサーバ側で保持され、クライアントは必要になったときのみサーバから受信する。こうすることでクライアント・サーバ間の通信を減らすことができ、結果としてクライアントの負荷を軽減できる。

この管理法を実装するために、分散行列クラス (DMatrix) と小行列クラス (UniMatrix) という2つのクラスを作成した。

ユーザプログラム内において、行列は全て分散行列クラスのオブジェクトとして記述するようにしてある。分散行列クラスのオブジェクトは、それぞれ内部に行列計算サーバ数と等しい数の小行列クラスのオブジェクトを持つようにする。小行列クラスのオブジェクトは、行列 ID とどの行列計算サーバに格納されているかを示す行列計算サーバ ID を保持する。なお、ユーザプログラムを作成するときに小行列クラスを意識する必要は全くない。

また、ユーザプログラム内では分散行列クラスのオブジェクトを用いて計算を行なっているが、行列計算サーバでは小行列単位での演算しか行なえないので、何らかの変換が必要になる。

そこで分散行列クラスの演算関数内では、小行列クラスのオブジェクト同士の演算を記述する。例として2項演算の加算を示す。

```
DMatrix
operator+(DMatrix &a, DMatrix &b)
{
    // 計算結果を格納するオブジェクト
    DMatrix r(a.m, a.n);
    int i;
    // DMatrix::Dnum は行列計算サーバ数
    for (i=0; i<DMatrix::Dnum; i++)
    {
        // Element が
        //    小行列クラスのオブジェクト
        r.Element[i]=
            a.Element[i]+b.Element[i];
```

```
    }
    return r;
}
```

このように計算関数内では、被演算子となる分散行列クラスのオブジェクト a,b が持つ小行列クラスのオブジェクト a.Element,b.Element 同士の演算が記述されている。

後は関数内に記述した小行列同士の演算を、各行列計算サーバに命令として送信する必要があるが、それは小行列クラスの演算命令内で記述する。同じく 2 項演算の加算の例を示す。

```
UniMatrix
operator+(UniMatrix &a, UniMatrix &b)
{
    // 計算結果を格納するオブジェクト
    UniMatrix r(a.GetServID());
    // 命令用構造体を宣言
    Instruction I;
    // 行列計算サーバに送信する命令を設定
    I.Inst            =Add;
    I.Ope[0].addr     =a.GetServID();
    I.Ope[0].MatrixID=a.GetMatrixID();
    I.Ope[1].addr     =b.GetServID();
    I.Ope[1].MatrixID=b.GetMatrixID();
    I.storeID         =r.GetMatrixID();
    // 命令送信
    SendInst(a.GetServID(), I);
    return r;
}
```

これによって、a.Element[1]+b.Element[1] を実行すると、小行列 a.Element[1] を格納している行列計算サーバに対して加算命令が送信されることになる。

計算命令を送信する際、サーバ側に必要な行列データが全て揃っていないことも考えられる。そのため、命令送信関数では被演算子が送信先のサーバに揃っているかを確認し、足りない場合はその行列データを持っているサーバに対して行列データの送信命令を自動的に発行するようにしている。

このような階層的な管理方法を用いることで、各クラスでの関数内の記述を簡潔に保つことが可能になり、容易に機能を追加、拡張できる。

3.2 行列計算サーバでの行列データ管理

行列計算サーバで扱う行列データは小行列だけである。つまり、サーバ側では小行列とユーザプログラム内の行列データとの対応を、全く意識しない。ただしクライアントとは違い、行列 ID だけでなく実際のデータも格納する。

ここで、クライアントからの指示はすべて行列 ID を用いて行なわれるため、小行列のデータを行列 ID で検索できるような形で格納しなければならない。

そこで小行列を格納するためのリスト (行列リスト) を作成した。行列リストでは、行列 ID をキーとして小行列データの挿入、削除、検索が可能になっている。

このように、行列計算サーバ側での行列データの管理はクライアントと比較して非常に単純にしている。このような設計を行なったのは、サーバの主な処理が行列計算と行列データ通信であり、行列データの管理を複雑化するとこれらの処理の高速化の弊害になるためである。

4 通信ブロックの軽減

通信ブロックには、

- 命令受信時に相手の命令送信を待つ命令受信待ち時間
- 命令送信時に相手の命令受信を待つ命令送信待ち時間
- データ受信時に相手のデータ送信を待つデータ受信待ち時間
- データ送信時に相手のデータ受信を待つデータ送信待ち時間

といったものが考えられる。本システムの性質上、通信ブロックが問題となるのは主に行列計算サーバ側である。そこで、通信ブロックの軽減のために行列計算サーバプログラムで用いた方法について説明する。

4.1 送信処理の子プロセス化

TCP で通信する場合、通信が終了するまで処理がブロックされてしまう。またノンブロッキングな通信を使うと、1 度にすべてのデータが通信されないのでデータが細切れになってしまう。すべてのデータが通信されるまで待っていたのでは

ブロックするのと同じなので、計算プログラムを細切れになったデータに合わせて組む必要があり、処理が複雑になってしまう。

これを解消するため、データ送信を行なう場合は子プロセスを生成し、そちらで送信処理を行なうようにした。これによって送信がブロックしても親プロセス側で受信や計算を行なうことができるので、通信ブロックの影響を抑えることができる。

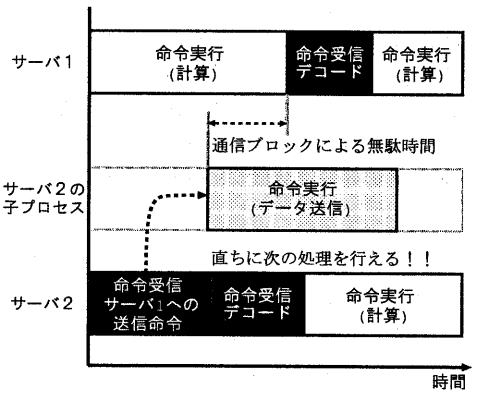

図 3: 送信処理の子プロセス化

4.2 メッセージキャッシング

行列計算サーバでクライアントから受信した命令を実行するときに、常に実行に必要な行列データが揃っているとは限らない。この場合、他のサーバから必要な行列データを受信することになるが、受信の待ち時間が生じてしまう。命令実行は通常逐次処理されるので、この待ち時間には他の処理を行なうことができず、処理効率が低下してしまう。

この問題の解決のために、データフロー型計算システムのようにプログラムの記述順でなくデータの依存関係により実際の実行順序を決定することを提案する。この提案方式をメッセージキャッシングと呼ぶことにする。図示すると図 4のようになる。

1. 受信命令が計算命令の場合
   命令実行に必要なデータが全て揃っている場合は直ちに実行し、揃っていない場合には命令を未受信の行列データの ID と共に命令キャッシュと呼ばれる場所に格納し、次の命令の受信に移る。
2. 受信命令がデータ受信の場合
   受信データを行列リストに格納し、命令キャッシュ内にそのデータを必要とする命令がないかを ID によって検索する。あった場合はその命令を実行し、なければ次の命令の受信に移る。
3. 受信命令が終了命令の場合
   行列計算サーバプログラムを終了する。

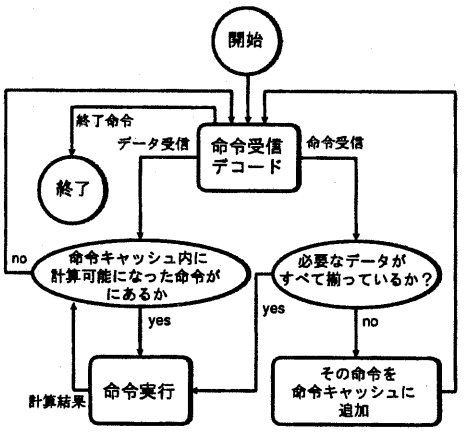

図 4: メッセージキャッシングのアルゴリズム

さらに命令キャッシュを検索するときには、計算命令よりも送信命令を優先している。これは、例えば計算命令と送信命令が実行可能になった場合、計算命令を先に実行すると計算が終わるまで送信命令を実行できないが、送信命令を先に実行すれば送信処理は子プロセスで行なうので、すぐに親プロセス側で計算命令を実行できるからである。

5 性能評価

今回は行列計算クライアント、サーバとも全て SGI 社の O2(R5000 180MHz,2 次キャッシュなし,64MByte) を使用し、2000 × 2000 の行列同士の乗算を行なったときの計算時間 (経過時間) を計測した。なお、各計算機は LAN(100BASE-TX) で接続されている。計算時間の単位は全て秒である。

このように分散処理を行わない場合と比較して、サーバ数 4 台の場合で約 4.6 倍、16 台の場合で約 10 倍という結果が出た。

サーバ数	1回目	2回目	3回目	平均
1台	1878	1948	1886	1904
4台	422	439	379	413
16台	198	192	181	190

表 1: 計算時間

次に今回の目的である通信ブロックの影響を調べるために、行列計算サーバにおける通信待ち時間を計測した。サーバ数 16 台で 2000 × 2000 の行列同士の乗算を行なった場合、通信待ち時間が約 125 秒であった。つまり計算時間の半分以上が通信待ちになっていることになる。16 台で分散処理を行なう場合、1 台の行列計算サーバは 21 回命令を受信することになっている。そこでどの部分でブロックが発生しているかを調べるために各命令の受信待ち時間を計測したのが図 5である。

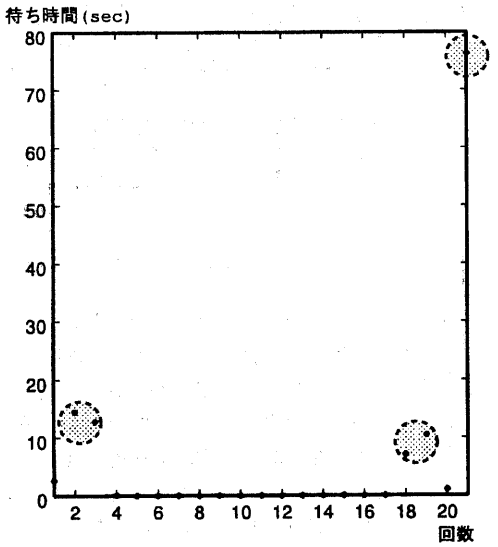

図 5: 通信待ち時間

これを見て分かるように、ほとんどの場合において通信待ち時間は発生しておらず、特定の 3 個所の待ち時間によって処理効率が落ちている。

最初の 2、3回目の部分は、クライアントからの行列データ受信に伴う待ち時間である。1 台のクライアントが全てのサーバに対して順次行列データを送信していくため、サーバ側ではどうしても待ち時間が発生してしまう。

最後の部分は計算結果をクライアントに送信した後に、計算終了命令を受信するまでの待ち時間である。この場合も同様に、1 台のクライアントが全てのサーバから順次行列データを受信しなければならないので、待ち時間が発生してしまう。

18、19回目の部分の待ち時間は、行列計算中には新たな命令を受信できないという行列計算サーバプログラムの制約から発生したものである。つまり命令の受信順によってはこの部分の待ち時間はほとんど発生しないこともあり、処理効率に差が出ることになる。表 1を見ると分散処理を行なった場合には計算時間のばらつきが大きいことが分かる。これも同じ原因によるものであると判断される。この部分は今後改良の余地があるだろう。

6 まとめ

今回の結果から、命令がキャッシュ中に多数あるときの通信ブロックの影響を抑えることには成功したが、受信側が計算中で受信待ちになっていないために発生する通信ブロックの影響を完全に抑えることができず、結果として計算命令実行時に必要なデータが揃わずデータ受信待ちになってしまうことがあった。

また、計算の開始と終了において行列計算クライアントの通信部分にシステムのボトルネックがあることが分かった。

今後は今回判明した問題点を解決し、サポート演算の追加、疎行列の対応などが課題である。

参考文献

[1] Al Geist, etc:"PVM 3 USER'S GUIDE AND REFERENCE MANUAL", 1993.

[2] Message Passing Interface Forum:"MPI: A Message-Passing Interface Standard", 1995.

[3] 松村 博光, 大鎌 広, 藤原 祥隆:"分散型大規模行列計算における通信ブロックの軽減", 電気関係学会北海道支部連合大会講演論文集, p.353(1997-10)