卒業研究論文

並列コンピュータの試作

2002年2月5日

高知工科大学
知能機械システム工学科
磯村研究室

伊敷 太郎

目　　次

「並列コンピュータの試作」

―緒言―

昨今のコンピュータに対する要求は非常に大きなものとなっている。日進月歩の勢いで高性能化が進んだコンピュータは高速な計算を可能としたが、今まで出来なかった計算が可能となったことで、より高精度で複雑な計算の要求が起こってきた。

ここでは計算時間を短くする手段として、一つの仕事を複数のプロセッサに分配して計算を進める並列化に着目した。

第一章　並列コンピュータの狙い

多量の計算を行うには処理を高速に行うことが肝要である。機械、工業分野の現場でのシミュレーションでは締め切りという時間的制約があるため、秒、分などの短時間で計算をこなす必要がある。高速な計算にはスーパーコンピュータが最も適しているが、非常に高価であり、全ての設計現場に採用することは困難であると考えられる。ここでは視点を変え、安価なワークステーションやパーソナルコンピュータを複数利用した、並列化による計算環境を試作したい。

並列化は複数のプロセッサに仕事を分担させるため、計算時間は短くなっているが、処理時間には変わりがないことが特徴となっている。しかし分割によるオーバーヘッドが加わるため、計算時間の総和では一つのプロセッサで順次処理を行った計算時間を上回る。よって並列化による性能の向上は最大でプロセッサの個数倍となる。図（図1．1）から並列化では仕事量とプロセッサの数が多い程効果を上げることが解る。頻繁にプロセッサ間で情報交換が行われる場合には、オーバーヘッドが大きくなり計算の高速化に繋がらない点が短所として挙げられる。しかし、並列化に配慮した計算では、複数のプロセッサに作業を分配することで、個々のプロセッサの負荷軽減や計算の高速処理に威力を発揮すると予想される。

並列計算ではプログラムの分割ができ、並列に実行出来る部分をよくプロセスと呼ぶ。つまり、並列計算は複数に分割したプロセスをプロセッサに振り分け、並列に処理を行って計算時間を短縮することと言える。プロセスとプロセッサの数が一致することが理想的だが、プロセスがプロセッサの数より多い場合、プロセッサは複数のプロセスを順に実行していくことになる。この時、処理はプロセッサの中でプロセスの順に処理されていく。

プログラム内の作業には分割出来るものと出来ないものがある。エンジンのシリンダの計算等ではシリンダごとの計算が別々に記述されるため、各計算をプロセッサに配分し並列計算が可能である。しかし、ある処理 A の結果に従って処理 B が実行される場合の並列化は困難である。

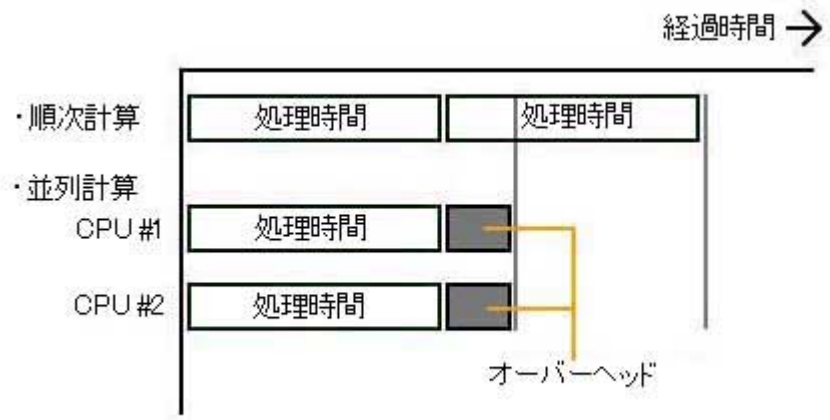

図1．1　順次計算と並列計算の経過時間の違い

並列コンピュータには幾つかの種類が挙げられるが、本研究ではメッセージ通信マルチプロセッサ方式を採用した（図1．2）。このシステムは完全なコンピュータで構成され、他のコンピュータからメモリ領域に直接アクセスすることが出来ず、プロセッサは自らのメモリ領域のみを参照出来るところに特徴がある。一つ一つのコンピュータが独立して動作できる、単体で完結したコンピュータ（ワークステーション等）で構成される場合はマルチコンピュータとも呼ばれ、本研究の並列環境もこの形である。

並列コンピュータが稼働するには情報の共有が不可欠であるが、このシステムでは構造上、コンピュータが他の場所にあるメモリ領域を参照することが出来ないため、情報の受け渡しはメッセージを送る形でプロセッサ間を行き来する。このメッセージは一つのプロセッサから別のプロセッサにプログラムの指定に従ってデータを運ぶ。つまりマルチプロセッサ方式では、データや結果の受け渡しはメッセージでのみプロセス間の通信が行われる。

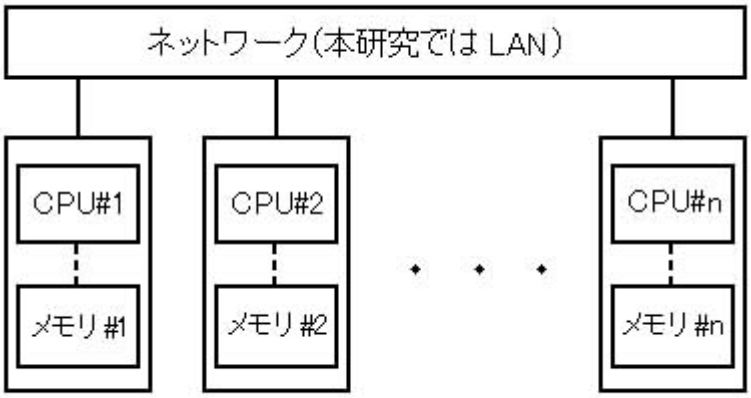

図1．2　メッセージ通信マルチプロセッサシステム

計算時間の短縮にはもう一つの方法がある。それはベクトル化と呼ばれるもので計算を配列で扱い、連続する数の計算を高速な命令で一括して処理を行う方法である。ベクトル化では計算方法が違うため処理時間が短くなり、その結果計算時間の短縮につながっている（図1．3）。一般にスカラー計算を効果的にベクトル化した場合、効果は10倍以上にのぼるとされる。

つまり、問題を並列化で分割し、分割した計算をベクトル化でさらに効率化して計算を行うことが、最も高速に並列計算を行う方法となる。

図1．3　順次計算とベクトル化による処理時間の違い

本研究では各コンピュータの結合を LAN で行うこととした。理由としては高知工科大学のインフラとして身近にあり、社会的にも最も普及した接続方法であると考えたからである。LAN での結合に用いるスイッチングハブは磯村研究室で使用していたものを本研究に利用した。

本研究は試作したマルチプロセッサでプログラムを並列動作させる計算環境を構築することを目的とし、テーマを「並列コンピュータの試作」とした。

第二章　ハードウェア

ここでは並列システムに用いるコンピュータを、下記のパーツ単位で購入し、2台組み立てた。これはコンピュータを実際に組み立て、コンピュータの振る舞いとハードウェアがどのように関わるのかを確認することで、コンピュータやプロセッサを内蔵する機器の設計思想が判るのではないか、と考えたためである。システムを2台のコンピュータで構成した理由は、2台構成が最も基本的なマルチプロセッサシステムとなるからである。

・プロセッサ	PentiumⅢ　1．0GHz	
・メモリカード	256Mbyte　SDR SD-RAM	
・ハードディスク30Gbyte	7200rpm	
・マザーボード	Socket370ボード　i815チップセット搭載	
・LAN カード	10Base T／100Base TX	
・フロッピードライブ	3．5インチフロッピードライブ	
・CD-ROM ドライブ	IDE　40倍速	
・CD-RW ドライブ	IDE　読み込み32倍速／書き込み12倍速／書き換え10倍速	
・ケース	ATX ミドルタワー　300W 電源	
・ディスプレイ	17インチ	
・ビデオカード	ビデオメモリ8Mbyte 搭載　AGP	
・サウンドカード	Sound Blaster　Live Digital Audio2	
・キーボード	日本語109キーボード	
・マウス	3ボタンマウス	
・ケースファン	80mm　センサー付き　3000rpm	
・ケーブル	ATA66／100フラットケーブル	
	電源分岐ケーブル	
	CD オーディオケーブル	各2セット
・スイッチングハブ	8ポート	1台

ここでは、コンピュータの組み立てを追って解説する。初めにマザーボードにメモリカードとプロセッサを、それぞれ専用に用意されたスロットに取り付ける。どちらもスロットの形状に従って、正しい向きに装着した。次に CPU ファンの取り付けとなる。ファンのプロセッサのコアに当たる部分に、プロセッサパッケージ付属の導熱グリスを塗り、CPU スロット両横に出ている爪にファンの留め具を引っかけ固定する。ファンの取り付けでプロセッサの損傷をさけるため、コアの両側に1、2mm 厚のパンフレットを切って作ったスペーサーを敷き、コアとファンが接触しないように注意して取り付けた。CPU ファンの電源のコードをマザーボードのコネクタに差し込み、プロセッサまわりの作業が終了した。

プロセッサとメモリカードをマザーボードに取り付けた段階で、ケースのシャシーにマザーボードを固定する。マザーボードに空いている固定用の穴に合わせて、基板を浮かすスペーサーをシャシーに取り付け、そこにマザーボードをネジ留めする。

購入したケースには、ケース用ファンの取り付け場所が前後に二つずつあった。購入したケース用ファンは二つであったため、冷却には空気の流れを作ることが必要と考え、前後に一つずつ装着し、それぞれの電源コードをマザーボードのコネクタと接続した。

ここでケースの前面に集中する電源、リセット各スイッチ、電源、ディスクアクセス確認用の LED、エラー時に作動するスピーカの電源を確保する。スイッチ、LED、スピーカのコードは一まとめになっていたので、マザーボードのマニュアルを参考にして、極性に注意しつつマザーボードにコネクタを差し込んで、スイッチ類の接続を終了した。

次にドライブ類をケースに組み付けた。取り付けるドライブは5インチ用ドライブベイに CD-ROM と CD-RW、3．5インチ用ベイにはフロッピーにハードディスクと、計4つのドライブを取り付けた。この作業はドライブの両側面をシャシーに固定するため、ケースの側板を外して作業を行った。まず、ベイに付いている覆いを外して、フロッピードライブから取り付けることにした。両側面を二カ所ずつ、シャシーにネジで固定する。次に CD-ROM、CD-R ドライブを同様に5インチベイにネジ留めした。最後にハードディスクドライブを装着した。ハードディスクは内部で、記憶部分が高速で回転するため、コンピュータの起動中に熱を持ちやすい。今回使用しているケースではフロッピードライブの下に3．5インチドライブ二つ分の空きがあったため、一つスペースを空けてハードディスク取り付けることとした。

CD-ROM ドライブとサウンドカードを繋ぐ必要があるため、ドライブ類とマザーボードの結線の前にカード類を差し込むことにした。取り付けるカードは3枚、前述のサウンドカード、それにビデオカードと LAN カードである。ビデオカードはケース背面から見て、最上段が定位置となっているので、そこへ取り付けた。LAN カードは、ビデオカードから一つスロットを空けて取り付け、サウンドカードの位置は最下段とした。ビデオカードのすぐ下を空けた理由は、ビデオカード用のスロットと、すぐ下の拡張カード用スロットは内部で配線を共有しており、IRQ（OS が拡張カードに割り振る住所のようなもの）の割り振りが上手くいかない場合があるからである。サウンドカードは比較的上の方に集中するプロセッサや、CPU ファンをはじめとするノイズ源から少しでも離した方が良いと考え、一番下に配置した。

カード類の取り付けが終わったところで、保留していたドライブ、カード、マザーボードの結線をする。まずフロッピードライブをマザーボードと接続した。フロッピー用のケーブルには1番ピンを差し込む位置が記されており、この目印を目安にマザーボード、ドライブのコネクタとの位置を合わせ、接続した。

次にマザーボードと CD-ROM、CD-RW、ハードディスクの接続を行う。このマザーボードにはプライマリと、セカンダリの二つのコネクタが、フロッピー以外のドライブ用に備えてあり、それぞれ2つずつドライブを取り付けることが出来る。今回はハードディスクをプライマリコネクタに、その他のドライブをセカンダリコネクタに取り付けた。フロッピー以外のドライブ用のケーブルには、1番ピンの目印はなかったがマザーボード、ドライブ、ケーブルのそれぞれのコネクタには、一本端子のない部分があり、そこを目安に接続した。今回は CD-ROM、CD-RW ドライブ両方ともセカンダリコネクタに繋げたため、二つを区別する必要があった。一つのコネクタに二つのドライブを繋ぐ時には、マスターとスレーブという設定を、それぞれのドライブに施すことになっている。ここでは各ドライブの表記に従い、CD-RW ドライブをマスター、CD-ROM ドライブをスレーブに設定してマザーボードと接続した。単独でプライマリコネクタに繋いだハードディスクはマスター設定とした。CD-ROM ドライブとサウンドカードの CD-IN コネクタをオーディオケーブルで接続して、データ用のケーブル接続が終了となる。

最後に電源ケーブルを接続して、マザーボード及び各種ドライブに電気を供給する。マザーボードには専用の大型のコネクタがあり、ここと電源とを接続する。ドライブ用には大小二種類のコネクタがあり、

この構成では小さい方をフロッピー用に、大きい方をその他のドライブ類に使用した。これでケース内部の作業は終わり、再び側板を取り付けて本体組み立ての作業は完了する。

本体とディスプレイを専用ケーブルで繋ぎ、キーボード、マウスをケース背面の所定の位置に繋いでコンピュータの機械部分は完成となる。

第三章　ソフトウェア

ソフトウェア（試作コンピュータ2台にそれぞれインストール）

- ・Microsoft　WindowsNT　Workstation　4.0
- ・Microsoft　Visual　C++　4.0
- ・Argonne National Laboratory　MPICH　Ver.1.2.3

本研究で扱うメッセージ通信マルチプロセッサ方式では、メッセージのやり取りが必須となる。ここではMPI（Message Passing Interface）というメッセージ通信規格を採用することにした。MPIはMPI－Standardとしてまとめらており、１９９４年５月５日に基本仕様であるMPI-1.1がリリースされ、以下の拡張を行ったMPI-2が１９９７年４月２５日に策定されている。

- ・Miscellaneous Topics
- ・Dynamic Process Management
- ・One-Sided Communication
- ・Extended Collective Operations
- ・External Interfaces
- ・Parallel I/O
- ・Language Bindings (C++ and Fortran-90)

本システムで並列計算を行うMPIライブラリには、Argonne National Laboratoryにより推奨パッケージとして作られたMPICHを選択し、組み込んだ。MPICHは移植が容易に出来るよう設計されており、様々な環境にMPIを普及させるべく開発された。世界で最も使われているライブラリの一つであり、LAN環境でも使えることから本研究に採用した。

MPICHはMPICH Home Page及びMCS(Mathematics and computer Science Division)のFTPサイトにて無償で配布されている。MPICHにはUNIX、Windowsの NTと2000が対応OSとして挙がっており、それぞれに専用のパッケージが公開されている。使用出来るプログラム言語はC言語、C++言語、FORTRAN言語の三つがサポートされており、最新版はVersion 1.2.3となっている。

OSはUNIXの操作には慣れていないため、Microsoft社のWindowsNT Workstation 4.0を選択した。

計算を記述するプログラム言語はMPIがサポートする中からMicrosoft社のVisual C++ 4.0を選んだ。

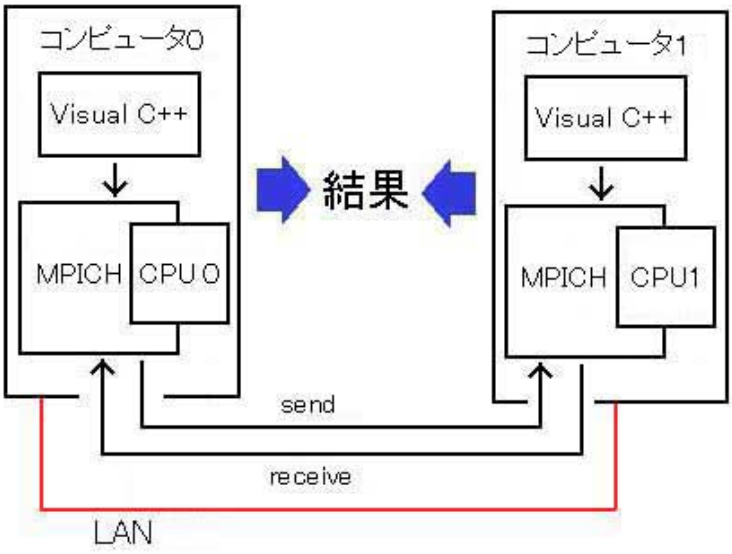

図3．1　MPICH による2台のコンピュータでの並列計算の模式図

試作並列コンピュータテスト用のプログラムは磯村研究室で開発した Send/Receive Test Program（0．0～9．0の実数から、それぞれの数の二乗、三乗の値と、二乗と三乗の和を求めるプログラム）を使わせていただいた。このプログラムは C++言語で書かれ、3つのプロセスに分けられている。

本研究では便宜上プロセッサとプロセスに名前を付けた。単独でプロセスを処理しているコンピュータのプロセッサを CPU0、そこで処理されるプロセスを Kanamaru と名付けた。二つの処理を担っている方のプロセッサは CPU1とし、そこで処理されるプロセス名をそれぞれ Iishiki−1、Ishiki−2とした。

テストプログラムは0．0～9．0の実数を CPU0から CPU1に送る。CPU0から受け取った実数を Ishiki−1で二乗、Ishiki−2では三乗の計算をしてから、それぞれ CPU0に送り返す。Kanamaru は CPU1から帰ってきた計算結果を足し合わせて表示する。プロセスの管理は myrank という変数が担っており、myrank=0 の時に Kanamaru、myrank=1 の時に Iishiki−1、myrank=2 の時には Ishiki−2のプロセスが処理される。

このプログラムはプロセスごとに、Kanamaru の処理の内容を send_recv_0.lis、Ishiki−1の処理時には send_recv_1.lis、Ishiki−2の処理では send_recv_2.lis のリストに出力するようになっている。これらの内容は各プロセスごとの処理の結果であり、プログラム実行後にコマンドプロンプトに表示される内容と同一である。

一つあるいはそれ以上のプロセッサで別のプロセスを実行出来るようにするためには、それぞれのプロセッサが、どのプロセスを実行するのかを選択する文がプログラムに必要となる。本並列プログラムでは、MPI_send、MPI_recv のルーチンで、各プロセッサに情報を受け渡しさせている。以下にプログラムの流れとプロセッサの関係を図に示す（図3．2）。

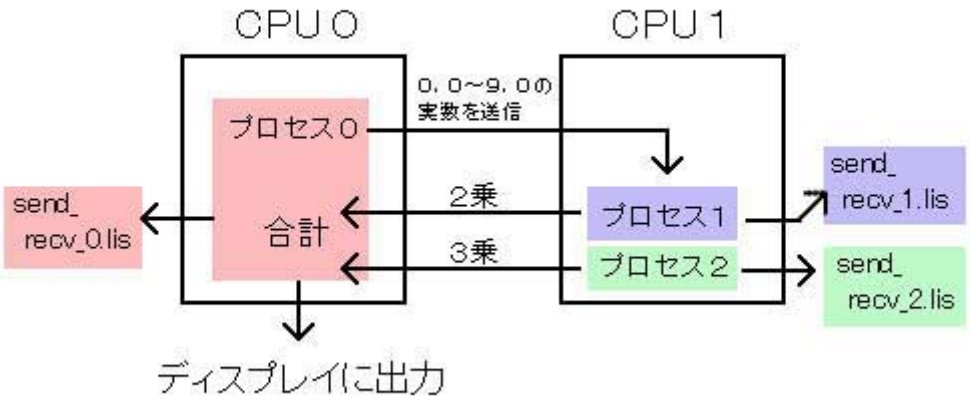

図3．2　Send/Receive Test Program とプロセッサの関係

第四章　　実行環境の構築

第二章で組み立てたコンピュータは、まだソフトウェアが一切組み込まれていない状態なので、必要なソフトをインストールする。まず OS である WindowsNT　Workstation　4.0 はパッケージ付属のブートディスクを読み込ませた後、ウィザードに従い CD-ROM を挿入し、インストールを終えた。

続いてサウンドカード、LAN カードのドライバを組み込んだ。サウンドカードはウィザードに従って正常にインストール出来たが、LAN カードはドライバの組み込みが上手くいかなかったため、一度ドライバを削除してから、LAN カード添付のフロッピーディスクで改めてインストールを行った。

MPICH は MCS の FTP サイト ftp.mcs.anl.gov/pub/moi/nt より mpich.nt.1.2.3.exe、mpich.nt.1.2.3.tar.gz をダウンロードした。exe 形式ファイルで各プロセッサにインストール後、gz 圧縮ファイルを解凍したものの中から、インクルードファイル等、必要なファイルをコピーした。

最後に Visual C++ 4.0 のインストールを行った。初期設定を行い、並列計算に必要なファイルを Visual C++に取り込むため、インクルードファイルの置かれているディレクトリのパス指定を行ってコンパイルの準備が完了した。

並列プログラムを Visual C++でコンパイル、ビルドして、実行ファイルの作成が完了すると、MPICH による並列処理が可能となる。並列計算は MPICH パッケージの中の mpirun.exe で、コマンドプロンプトを通して行われる。mpirun は実行時に、実行ファイルの位置をコンフィギュレーションファイルで指定する（図4．1）。コンフィギュレーションファイルは、2台のプロセッサのコンピュータ名と実行ファイルのパスを記述したもので、テキスト形式のファイルである。

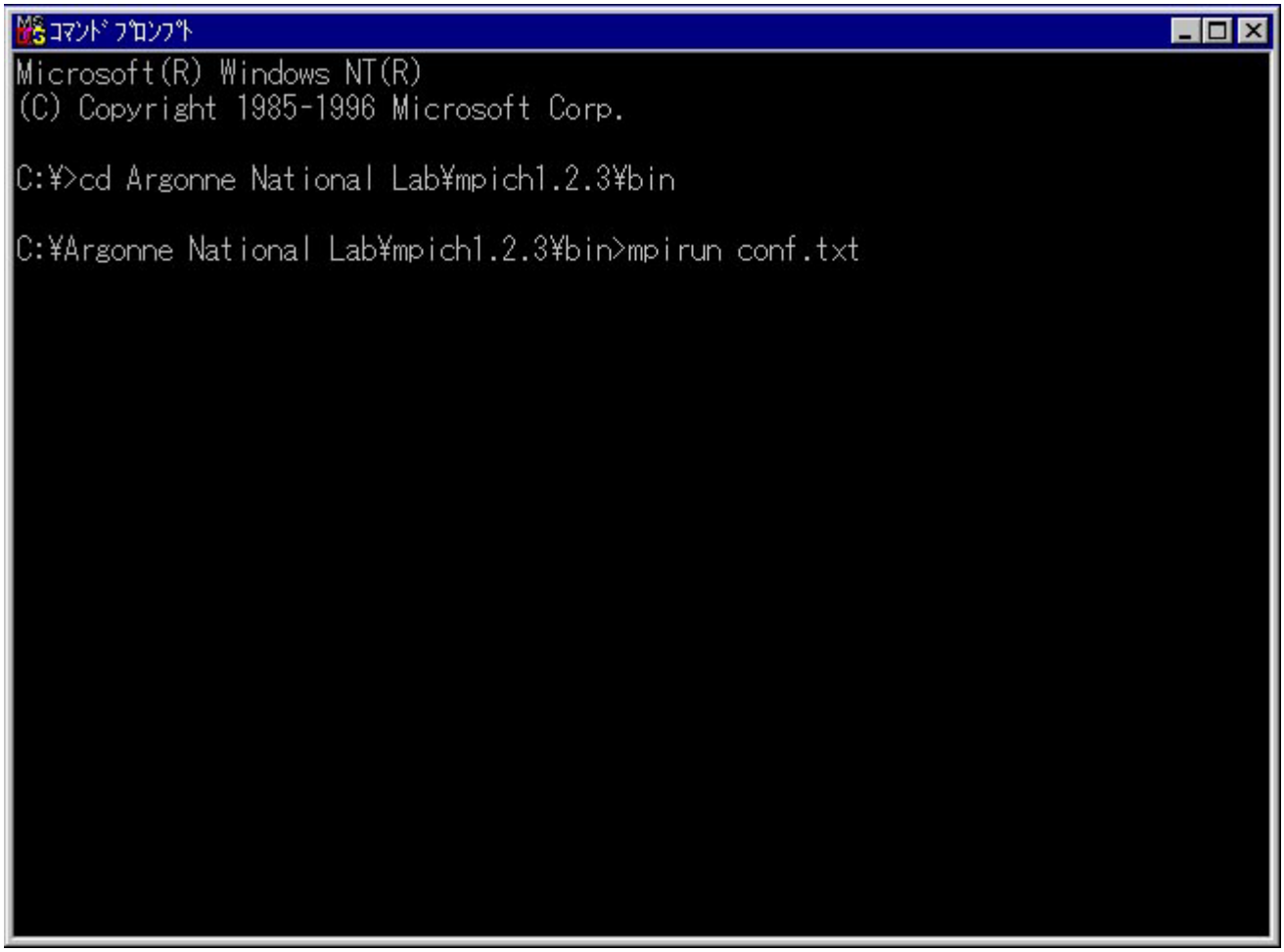

図4．1　並列計算実行直前のコマンドプロンプト画面

第五章　　実行結果

上記の並列プログラムが実行出来たことで、並列計算環境の獲得が確認できた。以下に示すのは myrank =0 の出力結果である send_recv_0.lis 、同じく myrank =1、myrank =2 で出力される send_recv_1.lis と send_recv_2.lis の内容であり、各プロセッサで行われた処理である。なお、左端の数字は並列システム全体での処理順序を表している。

CPU0(myrank=0)のプリントアウト（send_recv_0.lis の内容）

```
0 nproc=3
1 Kanamaru myrank=0   nproc=3   ysend1[0]=0.000 ptr_ysend1[0]=1244940 *ptr_ysend1[0]=0.000
1 Kanamaru myrank=0   nproc=3   ysend1[1]=1.000 ptr_ysend1[1]=1244948 *ptr_ysend1[1]=1.000
1 Kanamaru myrank=0   nproc=3   ysend1[2]=2.000 ptr_ysend1[2]=1244956 *ptr_ysend1[2]=2.000
1 Kanamaru myrank=0   nproc=3   ysend1[3]=3.000 ptr_ysend1[3]=1244964 *ptr_ysend1[3]=3.000
1 Kanamaru myrank=0   nproc=3   ysend1[4]=4.000 ptr_ysend1[4]=1244972 *ptr_ysend1[4]=4.000
1 Kanamaru myrank=0   nproc=3   ysend1[5]=5.000 ptr_ysend1[5]=1244980 *ptr_ysend1[5]=5.000
1 Kanamaru myrank=0   nproc=3   ysend1[6]=6.000 ptr_ysend1[6]=1244988 *ptr_ysend1[6]=6.000
1 Kanamaru myrank=0   nproc=3   ysend1[7]=7.000 ptr_ysend1[7]=1244996 *ptr_ysend1[7]=7.000
1 Kanamaru myrank=0   nproc=3   ysend1[8]=8.000 ptr_ysend1[8]=1245004 *ptr_ysend1[8]=8.000
1 Kanamaru myrank=0   nproc=3   ysend1[9]=9.000 ptr_ysend1[9]=1245012 *ptr_ysend1[9]=9.000
2 Kanamaru myrank=0   nproc=3   ysend2[0]=0.000 ptr_ysend2[0]=1244700 *ptr_ysend2[0]=0.000
2 Kanamaru myrank=0   nproc=3   ysend2[1]=1.000 ptr_ysend2[1]=1244708 *ptr_ysend2[1]=1.000
2 Kanamaru myrank=0   nproc=3   ysend2[2]=2.000 ptr_ysend2[2]=1244716 *ptr_ysend2[2]=2.000
2 Kanamaru myrank=0   nproc=3   ysend2[3]=3.000 ptr_ysend2[3]=1244724 *ptr_ysend2[3]=3.000
```

```
2 Kanamaru myrank=0   nproc=3     ysend2[4]=4.000 ptr_ysend2[4]=1244732 *ptr_ysend2[4]=4.000
2 Kanamaru myrank=0   nproc=3     ysend2[5]=5.000 ptr_ysend2[5]=1244740 *ptr_ysend2[5]=5.000
2 Kanamaru myrank=0   nproc=3     ysend2[6]=6.000 ptr_ysend2[6]=1244748 *ptr_ysend2[6]=6.000
2 Kanamaru myrank=0   nproc=3     ysend2[7]=7.000 ptr_ysend2[7]=1244756 *ptr_ysend2[7]=7.000
2 Kanamaru myrank=0   nproc=3     ysend2[8]=8.000 ptr_ysend2[8]=1244764 *ptr_ysend2[8]=8.000
2 Kanamaru myrank=0   nproc=3     ysend2[9]=9.000 ptr_ysend2[9]=1244772 *ptr_ysend2[9]=9.000
7 Ishiki-1 myrank=0   nproc=3 yrecv3[0]=0.000 ptr_yrecv3[0]=1244460 *ptr_yrecv3[0]=0.000
7 Ishiki-1 myrank=0   nproc=3 yrecv3[1]=1.000 ptr_yrecv3[1]=1244468 *ptr_yrecv3[1]=1.000
7 Ishiki-1 myrank=0   nproc=3 yrecv3[2]=4.000 ptr_yrecv3[2]=1244476 *ptr_yrecv3[2]=4.000
7 Ishiki-1 myrank=0   nproc=3 yrecv3[3]=9.000 ptr_yrecv3[3]=1244484 *ptr_yrecv3[3]=9.000
7 Ishiki-1 myrank=0   nproc=3 yrecv3[4]=16.000 ptr_yrecv3[4]=1244492 *ptr_yrecv3[4]=16.000
7 Ishiki-1 myrank=0   nproc=3 yrecv3[5]=25.000 ptr_yrecv3[5]=1244500 *ptr_yrecv3[5]=25.000
7 Ishiki-1 myrank=0   nproc=3 yrecv3[6]=36.000 ptr_yrecv3[6]=1244508 *ptr_yrecv3[6]=36.000
7 Ishiki-1 myrank=0   nproc=3 yrecv3[7]=49.000 ptr_yrecv3[7]=1244516 *ptr_yrecv3[7]=49.000
7 Ishiki-1 myrank=0   nproc=3 yrecv3[8]=64.000 ptr_yrecv3[8]=1244524 *ptr_yrecv3[8]=64.000
7 Ishiki-1 myrank=0   nproc=3 yrecv3[9]=81.000 ptr_yrecv3[9]=1244532 *ptr_yrecv3[9]=81.000
8 Ishiki-2 myrank=0   nproc=3 yrecv4[0]=0.000 ptr_yrecv4[0]=1244220 *ptr_yrecv4[0]=0.000
8 Ishiki-2 myrank=0   nproc=3 yrecv4[1]=1.000 ptr_yrecv4[1]=1244228 *ptr_yrecv4[1]=1.000
8 Ishiki-2 myrank=0   nproc=3 yrecv4[2]=8.000 ptr_yrecv4[2]=1244236 *ptr_yrecv4[2]=8.000
8 Ishiki-2 myrank=0   nproc=3 yrecv4[3]=27.000 ptr_yrecv4[3]=1244244 *ptr_yrecv4[3]=27.000
8 Ishiki-2 myrank=0   nproc=3 yrecv4[4]=64.000 ptr_yrecv4[4]=1244252 *ptr_yrecv4[4]=64.000
8 Ishiki-2 myrank=0   nproc=3 yrecv4[5]=125.000 ptr_yrecv4[5]=1244260 *ptr_yrecv4[5]=125.000
8 Ishiki-2 myrank=0   nproc=3 yrecv4[6]=216.000 ptr_yrecv4[6]=1244268 *ptr_yrecv4[6]=216.000
8 Ishiki-2 myrank=0   nproc=3 yrecv4[7]=343.000 ptr_yrecv4[7]=1244276 *ptr_yrecv4[7]=343.000
8 Ishiki-2 myrank=0   nproc=3 yrecv4[8]=512.000 ptr_yrecv4[8]=1244284 *ptr_yrecv4[8]=512.000
8 Ishiki-2 myrank=0   nproc=3 yrecv4[9]=729.000 ptr_yrecv4[9]=1244292 *ptr_yrecv4[9]=729.000
9 sum[0]=0.000
9 sum[1]=2.000
9 sum[2]=12.000
9 sum[3]=36.000
9 sum[4]=80.000
9 sum[5]=150.000
9 sum[6]=252.000
9 sum[7]=392.000
9 sum[8]=576.000
9 sum[9]=810.000
CPU  Time  890(msec)
```

CPU1(mayrank=1)のプリントアウト(send_recv_1.lis の内容)

```
3 Ishiki-1 myrank=1   nproc=3 yrecv1[0]=0.000 ptr_yrecv1[0]=1244940 *ptr_yrecv1[0]=0.000
3 Ishiki-1 myrank=1   nproc=3 yrecv1[1]=1.000 ptr_yrecv1[1]=1244948 *ptr_yrecv1[1]=1.000
3 Ishiki-1 myrank=1   nproc=3 yrecv1[2]=2.000 ptr_yrecv1[2]=1244956 *ptr_yrecv1[2]=2.000
3 Ishiki-1 myrank=1   nproc=3 yrecv1[3]=3.000 ptr_yrecv1[3]=1244964 *ptr_yrecv1[3]=3.000
3 Ishiki-1 myrank=1   nproc=3 yrecv1[4]=4.000 ptr_yrecv1[4]=1244972 *ptr_yrecv1[4]=4.000
3 Ishiki-1 myrank=1   nproc=3 yrecv1[5]=5.000 ptr_yrecv1[5]=1244980 *ptr_yrecv1[5]=5.000
3 Ishiki-1 myrank=1   nproc=3 yrecv1[6]=6.000 ptr_yrecv1[6]=1244988 *ptr_yrecv1[6]=6.000
3 Ishiki-1 myrank=1   nproc=3 yrecv1[7]=7.000 ptr_yrecv1[7]=1244996 *ptr_yrecv1[7]=7.000
3 Ishiki-1 myrank=1   nproc=3 yrecv1[8]=8.000 ptr_yrecv1[8]=1245004 *ptr_yrecv1[8]=8.000
3 Ishiki-1 myrank=1   nproc=3 yrecv1[9]=9.000 ptr_yrecv1[9]=1245012 *ptr_yrecv1[9]=9.000
5 Ishiki-1 myrank=1   nproc=3     ysend3[0]=0.000 ptr_ysend3[0]=1244460 *ptr_ysend3[0]=0.000
5 Ishiki-1 myrank=1   nproc=3     ysend3[1]=1.000 ptr_ysend3[1]=1244468 *ptr_ysend3[1]=1.000
5 Ishiki-1 myrank=1   nproc=3     ysend3[2]=4.000 ptr_ysend3[2]=1244476 *ptr_ysend3[2]=4.000
5 Ishiki-1 myrank=1   nproc=3     ysend3[3]=9.000 ptr_ysend3[3]=1244484 *ptr_ysend3[3]=9.000
5 Ishiki-1 myrank=1   nproc=3     ysend3[4]=16.000 ptr_ysend3[4]=1244492 *ptr_ysend3[4]=16.000
```

```
5 Ishiki-1 myrank=1   nproc=3    ysend3[5]=25.000 ptr_ysend3[5]=1244500 *ptr_ysend3[5]=25.000
5 Ishiki-1 myrank=1   nproc=3    ysend3[6]=36.000 ptr_ysend3[6]=1244508 *ptr_ysend3[6]=36.000
5 Ishiki-1 myrank=1   nproc=3    ysend3[7]=49.000 ptr_ysend3[7]=1244516 *ptr_ysend3[7]=49.000
5 Ishiki-1 myrank=1   nproc=3    ysend3[8]=64.000 ptr_ysend3[8]=1244524 *ptr_ysend3[8]=64.000
5 Ishiki-1 myrank=1   nproc=3    ysend3[9]=81.000 ptr_ysend3[9]=1244532 *ptr_ysend3[9]=81.000
CPU  Time  437(msec)
```

CPU1(myrank=2)のプリントアウト(send_recv_2.lis の内容)

```
4 Ishiki-2 myrank=2   nproc=3 yrecv2[0]=0.000 ptr_yrecv2[0]=1244700 *ptr_yrecv2[0]=0.000
4 Ishiki-2 myrank=2   nproc=3 yrecv2[1]=1.000 ptr_yrecv2[1]=1244708 *ptr_yrecv2[1]=1.000
4 Ishiki-2 myrank=2   nproc=3 yrecv2[2]=2.000 ptr_yrecv2[2]=1244716 *ptr_yrecv2[2]=2.000
4 Ishiki-2 myrank=2   nproc=3 yrecv2[3]=3.000 ptr_yrecv2[3]=1244724 *ptr_yrecv2[3]=3.000
4 Ishiki-2 myrank=2   nproc=3 yrecv2[4]=4.000 ptr_yrecv2[4]=1244732 *ptr_yrecv2[4]=4.000
4 Ishiki-2 myrank=2   nproc=3 yrecv2[5]=5.000 ptr_yrecv2[5]=1244740 *ptr_yrecv2[5]=5.000
4 Ishiki-2 myrank=2   nproc=3 yrecv2[6]=6.000 ptr_yrecv2[6]=1244748 *ptr_yrecv2[6]=6.000
4 Ishiki-2 myrank=2   nproc=3 yrecv2[7]=7.000 ptr_yrecv2[7]=1244756 *ptr_yrecv2[7]=7.000
4 Ishiki-2 myrank=2   nproc=3 yrecv2[8]=8.000 ptr_yrecv2[8]=1244764 *ptr_yrecv2[8]=8.000
4 Ishiki-2 myrank=2   nproc=3 yrecv2[9]=9.000 ptr_yrecv2[9]=1244772 *ptr_yrecv2[9]=9.000
6 Ishiki-2 myrank=2   nproc=3    ysend4[0]=0.000 ptr_ysend4[0]=1244220 *ptr_ysend4[0]=0.000
6 Ishiki-2 myrank=2   nproc=3    ysend4[1]=1.000 ptr_ysend4[1]=1244228 *ptr_ysend4[1]=1.000
6 Ishiki-2 myrank=2   nproc=3    ysend4[2]=8.000 ptr_ysend4[2]=1244236 *ptr_ysend4[2]=8.000
6 Ishiki-2 myrank=2   nproc=3    ysend4[3]=27.000 ptr_ysend4[3]=1244244 *ptr_ysend4[3]=27.000
6 Ishiki-2 myrank=2   nproc=3    ysend4[4]=64.000 ptr_ysend4[4]=1244252 *ptr_ysend4[4]=64.000
6 Ishiki-2 myrank=2   nproc=3    ysend4[5]=125.000 ptr_ysend4[5]=1244260 *ptr_ysend4[5]=125.000
6 Ishiki-2 myrank=2   nproc=3    ysend4[6]=216.000 ptr_ysend4[6]=1244268 *ptr_ysend4[6]=216.000
6 Ishiki-2 myrank=2   nproc=3    ysend4[7]=343.000 ptr_ysend4[7]=1244276 *ptr_ysend4[7]=343.000
6 Ishiki-2 myrank=2   nproc=3    ysend4[8]=512.000 ptr_ysend4[8]=1244284 *ptr_ysend4[8]=512.000
6 Ishiki-2 myrank=2   nproc=3    ysend4[9]=729.000 ptr_ysend4[9]=1244292 *ptr_ysend4[9]=729.000
CPU  Time  656(msec)
```

行っている計算は第三章の図(図3．2)にあるように、それぞれ myrank=0 が二乗と三乗の和、myrank=1 が二乗、myrank=2 が三乗の計算を、それぞれ分担している。send_recv_0.lis と send_recv_1.lis、send_recv_2.lis の内容に作業量の差があるように見えるが、これは myrank=0 の時に myrank=1、 myrank=2 にそれぞれ送信と受信の記録を含んでいるためである。また、Kanamaru プロセスの実行結果である send_recv_0.lis では myrank が0にも関わらず、myrank の1、2で行われている Ishiki-1 と Ishiki-2 のプロセスが混在するように見えるが、これは CPU1での計算結果を受信した結果、このように表示されるためで、処理は CPU0内で行われている。

各リストの最終行にある CPU Time はプロセッサの、処理したプロセスにおける稼働時間を表す。CPU Time は CPU0で 890(msec)、CPU1で Ishiki-1が 437(msec)、Ishiki-2が 656(Msec)と、合計の処理時間が 1093(msec)となった。これらの時間から、二つの CPU で作業がほぼ均等に実行されたことが解る。また、Ishiki-1と Ishiki-2の処理時間の違いは、2乗と3乗の計算量の違いによるものと考えられる。処理時間の大半は画面と、リストファイルへの書き込みに要する時間であり、計算に費やす時間は、表示された時間より短くなる。この結果から書き込み作業であっても、並列化することで負荷分散や高速化が図られていると言える。

本研究では0～9の実数を扱ったが、初期値を0～90としてプログラムを並列稼働させてみた。この時の CPU Time は Kanamaru が 2953(msec)、Ishiki-1が 437(msec)、Ishiki-2が 875(msec)とそれぞれ増加し、計算量と処理時間は比例関係にあることが示された。0～9の時に比べて0～90の処理をした時の CPU Time が Ishiki-2で大きくのびているが、これは初期値が10～90まで増えたため、初期値を3乗する時間を要したためと予想される。また、この時コマンドプロンプトの表示が上記のリストのようにプロセスの順には表示されず、途中に他のプロセスの記述が混じって表示された。これは処理の量が増えたため、プロセスの同期が取れなくなったためと考えられる。

―結言―

1．試作したシステムが並列コンピュータとして正常に稼働した。

2．テストプログラムとして実行した

Kanamaru ０．０～９．０ （送る）→ Ishiki-１（二乗）、Ishiki-２（三乗）

（返す）→ Kanamaru（合計）→出力

という並列プログラムが正常に動作した。

今回は LAN における通信の所要時間が短く、CPU の処理時間に比べて無視できる。つまり、オーバーヘッドが無いと見なせるので、一台で計算を行った場合に比べ処理速度は二倍となる。

計算のタイミングをとることに苦労したが、受け側の CPUOでのロードを増やし、待たせることでCPU１、CPU２での結果を受信する前に合計の計算を開始することが回避出来た。

また、ポインタアドレスは返るが、必要なデータが格納されない問題にも苦労した。

本研究では０．０～９．０の数の二乗と三乗を受け渡したが、計算量が少なく並列計算でも処理時間に明確な差は認められなかった。並列コンピュータの性能評価を考えた時には、オイラー関数、即ち素数の個数計算や、エンジンのシミュレーション等、複雑な計算プログラムが必要となるだろう。

今後の課題としては、複雑な問題を並列環境で扱うプログラミングに挑戦したい。

―謝辞―

試作並列プロセッサの動作検証のために磯村研究室で開発した並列計算プログラムを使わせていただき、ソフトウェアの問題解決に指導をいただいた磯村先生に感謝の意を表したい。

また助言を下さり色々と心配をしてくださった都築弘孝先輩、及び研究に使用したコンピュータの片方を組み立て、OS のインストールまで手伝ってくれた金丸和弘君に感謝したい。

―参考書籍及びウェブサイト―

参考書籍

「C 言語入門」 大角盛広 著 西東社

「Parallel Programming」Barry Wilkinson and Michael Allen

参考ウェブサイト

“Argonne National Laboratory“

http://www.anl.gov/

“MCS: Mathematics and Computer Science Division”

http://www-fp.mcs.anl.gov/division/welcome/default.asp

“MPI Forum“

http://www-unix.mcs.anl.gov/mpi/

“MPICH Home Page”

http://www-unix.mcs.anl.gov/mpi/mpich/

“スーパーコンピュータの自動並列化について”

http://www.cc.tohoku.ac.jp/refer/super/PARA/index.html