スーパーコンピューター SX-4 における MPI を用いた並列化分子動力学シミュレーション
(超高速衝突破壊シミュレーションへの適用)

大阪府立大学工学部航空宇宙工学科　中谷 敬子
nakatani@aero.osakafu-u.ac.jp
大阪府立大学大学院工学研究科　栢木 良典
kayanoki@struc.aero.osakafu-u.ac.jp
大阪府立大学工学部航空宇宙工学科　杉山 吉彦
sugiyama@aero.osakafu-u.ac.jp

Key Words : Molecular Dynamics, MPI, Parallel Processing, Space Debris, Hyper Velocity Impact

1　はじめに

昨年度の SX-4 モニター報告会では，並列化を意識していない (複数の CPU を利用することを前提としていない) シミュレーションプログラムに，並列化指示行を挿入することにより，複数の CPU(2 個) を使った並列化計算をする場合の，手順とその高速化の程度について報告した．SX4 に搭載されている MPI/SX は、すでにいくつかのグループが実績をあげている [1][2]．本年度は，これらの報告を参考にしながら，複数の CPU を利用することを意識したモデル，および，シミュレーションプログラムを SX-4 上で実行させる場合に，MPI/SX を用いた並列計算による高速化の結果と，試行錯誤の過程で得たいくつかの注意点を報告する．

以下では，まず，解析対象の概要について簡単にふれた後，SX-4 上での計算について報告する．具体的には，単独の CPU で実行したもの，MPI/SX を用いて，複数の CPU で並列計算したものについて示す．

2　解析対象の概要

解析対象としては，人工衛星と衝突し致命的なダメージを与えるスペースデブリ (宇宙ゴミ)[3] の超高速衝突破壊現象を取り上げる．その破壊のメカニズムを分子動力学シミュレーションによって解析することが狙いである．

2.1　解析モデルと結果の概要

図 1 に示すように，前述の問題を球形のプロジェクタイルと平板のターゲットによりモデル化する．これらはともに，アルミニウムの結晶構造である面心立方構造とする．解析条件を表 1 に示す．衝突のシミュレーションでは，ターゲットの周囲の 2848 個の原子は境界条件として完全に固定し初期温度を絶対零度に設定して，プロジェクタイルに 2 種の初速度 (表 3) を与えて，分子動力学シミュレーションを行なう．

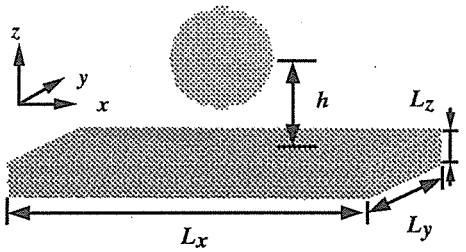


図 1: 超高速衝突破壊の解析モデル

表 1: シミュレーション条件

Initial temperature	0K
Potential	Shifted-Force potential (Morce potential)
Time increment	2.0×10^{-15} s
External stress	0 Pa

表 2: モデルサイズ

Projectile	$R = 2$ nm 1,721 atoms
Target	$L_x = 14.911$ nm $L_y = 14.760$ nm $L_z = 2.980$ nm 32,928 atoms

表 3: 解析ケース (衝突速度)

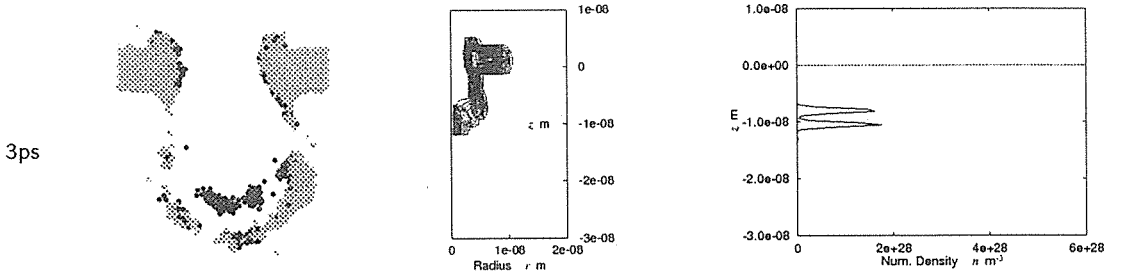
Case	Initial velocity of projectile v_{init} m/s
M1	7×10^3
M2	12×10^3

2.2 解析結果 (デブリ雲形成過程の速度依存性の検討)

図 2, 図 3 に，それぞれ，解析ケース M1，M2 に対する衝突時の変形およびデブリ雲分布の様子を示す．図中 (a) は，衝突直後のスナップショットであり，衝突に伴う変形の様子を表現するために，中央部で，厚さ 9.15Å の断面領域に含まれる原子のみを表示している．また，各構成原子のデブリ雲分布を表現するために，原子の色を，それぞれ，プロジェクタイル構成原子を濃灰色，ターゲット構成原子を薄灰色として示している．(b) は，プロジェクタイル，および，ターゲットの中心を貫く z 軸を中心として，xy 面内方向の距離 r の領域に存在する原子の数密度 $n(r,z)$ を表すコンター図であり，$0.05\times10^{28}1/\mathrm{m}^3$から $4.95\times10^{28}1/\mathrm{m}^3$の範囲を $0.1\times10^{28}1/\mathrm{m}^3$刻みで描かれている．(c) は，デブリ雲の構造を検討するために，z 周りの半径 15Å の領域に存在する原子のみに対して，(b) で表現した原子数密度を調べたものである．

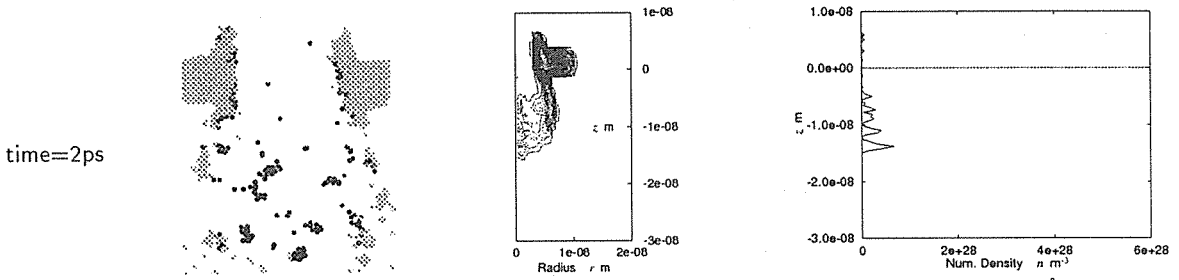
いずれの解析ケースの場合も，ターゲットは破壊され，穴は，塞がらずにスポール破壊が引き起こされ，ターゲットの後方に，デブリ雲が生じた．

各図 (a)，(c) から，デブリ雲の構造は，解析ケース M1 の場合は，地上実験 [2] で確認されている，前方，中央，そして，後方の 3 つの部分からなる構造を持っているが，より高速で衝突する M2 の場合は，各原子はクラスタ構造は作らずに，ばらばらに運動を続けることが分かる．また，変形の時系列データの検討から，衝突速度が速い場合，後方へのデブリ雲は，より大きな構造となり (各図 (b))，衝突直後から明瞭に成長する．一方, プロジェクタイルは，衝突直後から，内部に多くの空隙を持つクラスタ様の構造となっていることから，M2 は，速度の遅い M1 とは異なる破壊様式であることが確認された．



(a) Snapshot (b) Distribution $n(r,z)$ (c) $n(r,z)(0 \le r < 15\text{Å})$

図 2: スナップショットと原子配置の分布 (Case M1)



(a) Snapshot (b) Distribution $n(r,z)$ (c) $n(r,z)(0 \le r < 15\text{Å})$

図 3: スナップショットと原子配置の分布 (Case M2)

3 並列化ソフト MPI の概要

実際のシミュレーションでは，長時間の計算を実行した結果を示しているが，以下では，この問題に対して，初期の 25 ステップ (25 回の繰り返し) の計算を問題とする．この繰り返し計算では，定性的な計算量を比較する場合には十分であると考えられる．以下，SX-4[3] 上でのシミュレーションを追って述べる．

3.1 メッセージ通信操作の仕様標準 MPI について

MPI とは，分散型の並列処理プログラミングを行なう上で，任意のプログラム処理を受け持つ単位となるプロセス処理や，そのプロセスの間でデータをやりとりするために用いられるメッセージ通信操作の仕

様標準である.

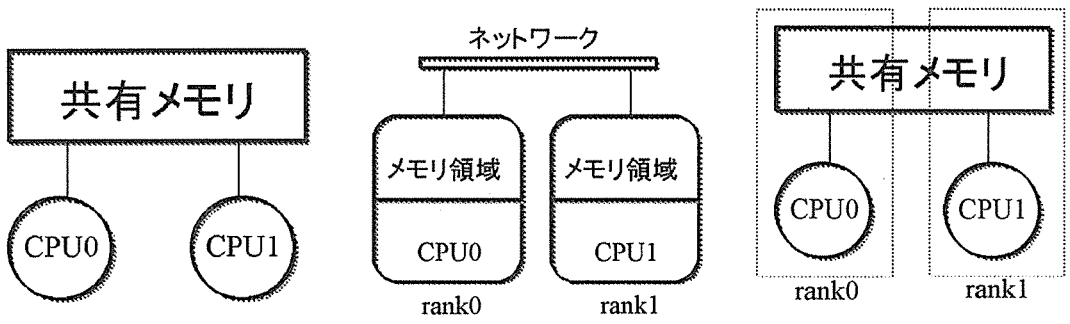
MPI が計算機のアーキテクチャに依存しない移植性の高いプログラミングモデルであることが認識されるようになり，このモデルを実装するために必要な基本機能の研究も盛んに行なわれた．1992 年には，MPI 標準規格を制定するための独立した組織 (MPIF) を設立，1994 年に標準規格第 1 版 MPI1.0 が発表され，サンプル実装として，MPICH が公開された．1997 年には，MPI2.0 が発表され，現在多くのベンダが実装を進めている [4]-[8].

大阪大学大型計算機センターの SX-4 に搭載されている MPI/SX は，標準規格第 1 版 MPI1.0 に準拠した機能を提供しており，SX-4 の特徴ののひとつである共有メモリーを活かした高速な通信を実現している [9].

3.2 並列計算機の形態

並列分散処理を行なう環境において，メモリ管理方式は，図 4 に示すとおり，共有メモリ型 (図 4(a))，分散メモリ型 (図 4(b)) に分類できる [10]．共有メモリは，全プロセッサでメモリ空間を共有しているため，相互の通信では，そのメモリ空間を直接アクセスすることができるため，通信が超高速で行なわれる．分散メモリ型では各プロセッサでメモリ空間が独立なため，相互の通信では，各プロセッサがデータ転送処理を行なう必要がある．この分散メモリ型並列処理環境において，並列に動作しているプログラム単位間で，データをやりとりしながら進めていく処理方式をメッセージパッシングと呼ぶ．最近では，ネットワークに接続された複数のワークステーションや PC を 1 つの仮想的な並列分散処理環境とみなし活用され始めている．これは，ネットワークを介して通信を行なうため，計算速度は通信速度が律速となる面があるが，安価に高速な処理を実現する手段として，近年注目を集めている．

報告者らは，共有メモリ型マシンである SX-4 で MPI/SX を用いて並列計算するにあたり，`-G local` オプションを利用することにより，仮想的には分散型メモリでありながら，ネットワークを介さないため超高速で通信できる図 4(c) のようなメモリ環境のもとで計算を実施した.



(a) 共有メモリ型　(b) 分散メモリ型　(c) 本稿でのメモリ型

図 4: 並列計算機のメモリ管理方式

4 SX-4 における MPI/SX による並列シミュレーション

SX-4 マシンを用いて，MPI/SX による並列計算を実行し，計算速度を比較する．はじめに，具体的な並列計算の例を説明し，続く数節で，MPI 関数をソースファイルを引用しつつ，詳説する.

4.1 MPI/SX による通信を含む具体的な並列計算の例 (モデルと流れ図)

若干偏った具体例にはなるが，具体的に扱う問題とその計算手順を紹介しておくほうが，続く 4.2 節以降の MPI/SX 関数を含んだプログラムでなされているプロセスの理解が簡単になると思われるので，ここでは，申請者らが並列計算をする際に扱った，具体的な問題と，そのモデル化，計算の流れを紹介する.

解析領域を図 5 のように，モデル中央の分割面で 1，2 の 2 つの部分領域に分ける．その結果，部分領域 1，2 の問題の原子数は，表 4 のようになり，それぞれ，領域 1 の座標データを in0.dat として，rank0 に，領域 2 の座標データを in1.dat として，rank1 に入力する．

通常の分子動力学シミュレーションでは，まず，粒子登録 (ブックキーピング) 法によりポテンシャルのカットオフ半径よりも適切に大きい領域に対して，相互作用する可能性のある原子の組をリストベクトルとして登録しておき，毎ステップの原子間力の評価は，あらかじめ登録した原子の組の中で実際に相互作用するものについてのみ計算を行なう方法を用いている．時間の経過とともに原子の運動により配置が変化するので，適切なステップ数で再度粒子登録を行なうという方法を繰り返す．

全領域を部分領域に分割することによって，境界線付近の原子の相互作用力を正確に評価するためには，分割した相手側の原子に関する座標等の情報が必要となる．ここでは境界線からある距離 D 内の領域 1R，2R にある原子の座標データを，図 5(ち) のように 1R'，2R' に通信によって送る．こうすることで，境界線付近の原子の粒子登録が正しく行なわれ，正確な相互作用力を求めることができる．

また，相互作用力の計算の前に行なう通信は，原子の座標データが更新されて誤っている領域 1R'，2R' を正確な座標データを持っている領域 1R，2R に置き換えるために行なう．MPI を使った並列計算の流れを図 6 に示す．なお，図 5，6 中の (こ)(ち) は次節以降のソースファイル中，及び，その説明の符号に対応している．

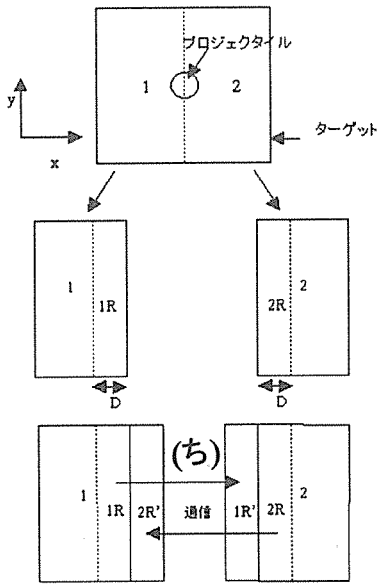


図 5: 原子の座標データの分割と送信

表 4: 原子数の分割

並列計算領域	無 全領域	有 領域 1	有 領域 2
原子数			
プロジェクタイル	1721	812	909
ターゲット	32928	16464	16464
総数	34649	17276	17373

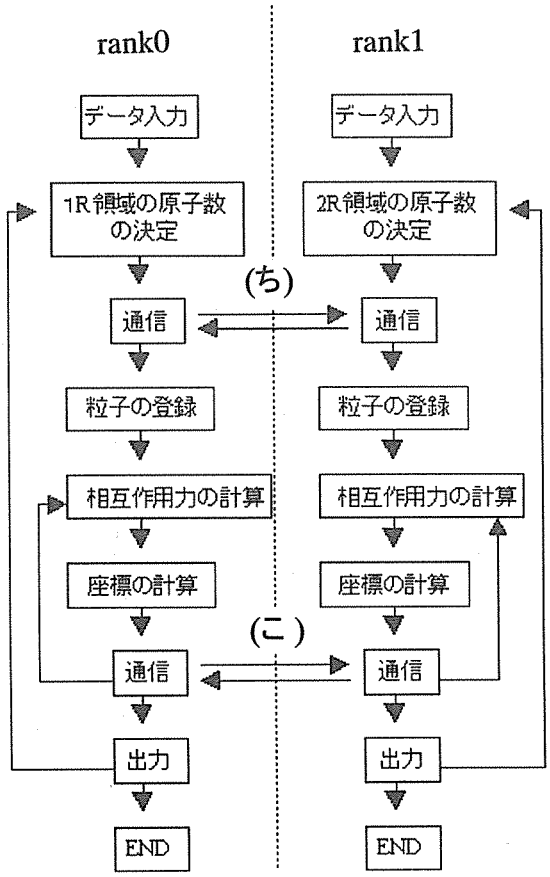


図 6: 並列計算の流れ

以下では，このアルゴリズムを実行するために必要な MPI 関数を使った Programing を紹介する．

4.2 MPI/SX 関数を含むプログラムソースの概要と MPI/SX 関数

ここでは，まず，実際に使ったソースプログラムを引用しながら，計算の流れを説明する．次に，最低限必要な MPI コマンドを紹介し，その後で，実際に試みたコンパイルの方法，実行方法について述べる．

4.2.1 MPI/SX 関数を含むプログラムソースの概要

- メインルーチン MDMAIN (抜粋)

```
C ***********************************************************************
      PROGRAM MDMAIN
C ***********************************************************************
C---+*---1----+----2----+----3----+----4----+----5----+----6----+----7--
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
C
      INCLUDE 'mpif.h'                                                  ...(あ)
C
      PARAMETER (NA=2,NAN=(NA*(NA+1))/2)
                     : (中 略)
      COMMON /SHARED/ NSHARE(NB),QSX(NB),QSY(NB),QSZ(NB),QVX(NB),QVY(NB)
     &      ,QVZ(NB)
      CHARACTER*256 FILE3,FILE4,FILE8,FILE9,FILE10,FILE11,FILE12
C
      call MPI_INIT( ierr )                                             ...(い)
      call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )              ...(う)(え)
      call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr )
      print *, "Process ", myid, " of ", numprocs, " is alive"
C
      if(myid.eq.0) then                                          <-----+
         I=3                                                            |
         WRITE(6,1100)I                                                 | ...(お)
         READ(5,1000) FILE3                                             |
                :(中略)                                                 |
      endif                                                       <-----+
 1000 FORMAT(A)
 1100 FORMAT(' ','FILE NAME UNIT=',I2)
                     : (中 略)
      call MPI_BCAST(FILE3,256,MPI_CHARACTER,0,MPI_COMM_WORLD,ierr)  <---+
                     : (中 略)                                            |(か)
      call MPI_BCAST(FILE12,256,MPI_CHARACTER,0,MPI_COMM_WORLD,ierr) <---+

      CALL FILES(FILE3,FILE4,FILE8,FILE9,FILE10,FILE11,FILE12)          ...(き)
                     : (中 略)
      CALL BOOKKEEP(RC,IER)                                             ...(く)
                     : (中 略)
      NATOMP=NATOM-NATOMV                                               ...(け)
      DO 10 I=1,NSTEP
        IF(MOD(I,NLSTEP).EQ.0) THEN
           CALL BOOKKEEP(RC,IER)
        END IF
        CALL SOLVE                                                      ...(こ)
        NATOMP=NATOM-NATOMV                                             ...(け)
 10   CONTINUE
      call MPI_FINALIZE(ierr)                                           ...(さ)
C
      STOP
      END
```

- サブルーチン FILES (抜粋)

```
C ***********************************************************************
      SUBROUTINE FILES(FILE3,FILE4,FILE8,FILE9,FILE10,FILE11,FILE12)
```

```
C ***************************************************************************
C---+*---1----+----2----+----3----+----4----+----5----+----6----+----7--
      COMMON /MPIID/ myid
      CHARACTER*256 FILE3,FILE4,FILE8,FILE9,FILE10,FILE11,FILE12
C
c 2100 FORMAT(' ','MYID is =',I2,'FILE = ',I2)
C *** INPUT FILE 3333333333333333333333333333333333333333333333333333333333
C
      OPEN(3,FILE=FILE3,STATUS='UNKNOWN')                              ...(し)
C
      IF(myid.eq.0) then                                          <-----+
        OPEN(4,FILE=FILE4,STATUS='UNKNOWN')                             |
                        :(中 略)                                        |
      ELSE IF (myid.eq.1) then                                          |...(す)
        OPEN(4,FILE=FILE10,STATUS='UNKNOWN')                            |
                        :(中 略)                                        |
      END IF                                                      <-----+
C
C *** BUNKATSU ZAHYOU FILE
C
      IFILE=20+myid
      IF(myid.eq.0) then                                          <-----+
         WRITE(6,*) myid,"in0.dat"                                      |
         OPEN(20+myid,FILE="in0.dat",STATUS='UNKNOWN')                  |
      ELSE                                                              |...(せ)
         WRITE(6,*) myid,"in1.dat"                                      |
         OPEN(20+myid,FILE="in1.dat",STATUS='UNKNOWN')                  |
      ENDIF                                                       <-----+
      RETURN
      END
```

- サブルーチン BOOKKEEP (抜粋)(調べた座標データを，別の rank へ通信している．)

```
      SUBROUTINE BOOKKEEP(RC,IER)
C ***************************************************************************
C---+*---1----+----2----+----3----+----4----+----5----+----6----+----7--
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
C
      INCLUDE 'mpif.h'                                                 ...(そ)
                          :(中略)
C
      CALL MPI_BARRIER(MPI_COMM_WORLD,ierr)                            ...(た)
C
      CALL MPI_ISEND(NATOMS,1,MPI_INTEGER,IDSEND,1,             <-----+
     &  MPI_COMM_WORLD,IS11,ierr)                                     |
      CALL MPI_IRECV(NATOMV,1,MPI_INTEGER,IDRECV,1,                   | ...(ち)
     &  MPI_COMM_WORLD,IV11,ierr)                               <-----+
      CALL MPI_WAIT(IV11,ISTATUS,ierr)
      CALL MPI_WAIT(IS11,ISTATUS,ierr)                                 ...(つ)
                          :(後略)
```

- サブルーチン SOLVE (抜粋)(座標データを，別の rank へ通信し，サブルーチン FORCE を call して，

相互作用力を計算する．)

```
      SUBROUTINE SOLVE
C ***********************************************************************
C---+*---1----+----2----+----3----+----4----+----5----+----6----+----7--
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      INCLUDE 'mpif.h'
C
                          :(中略)
C
      write(*,*) myid, 'SOLVE SEND RECV'
      IDSEND=IDSENDN(MYID+1)
      IDRECV=IDRECVN(MYID+1)
      CALL MPI_ISEND(NATOMS,1,MPI_INTEGER,IDSEND,1,          <----+
     &  MPI_COMM_WORLD,IS11,ierr)                                 |
      CALL MPI_IRECV(NATOMV,1,MPI_INTEGER,IDRECV,1,                | ...(ち)
     &  MPI_COMM_WORLD,IV11,ierr)                           <----+
                          :(中略)
      CALL MPI_WAIT(IV11,ISTATUS,ierr)
      CALL MPI_WAIT(IS11,ISTATUS,ierr)                           ...(つ)
                          :(中略)
      CALL FORCE(1)
                          :(後略)
```

- メインルーチン MDMAIN

(あ) MPI ライブラリを使用するために，ヘッダファイル (mpif.h) を読み込む．

(い) MPI プロセスの初期化

(う) プロセスの識別
どのプロセスがどのプロセスにメッセージを送るのかを指定する手段が必要となる．MPI では，プロセスをランクによって識別する．プロセスは，自分のランクを関数 MPI_COMM_RANK で知ることができる．

(え) これを実行したプロセスの MPI_COMM_WORLD におけるランクが myrank に返ってくる．MPI_COMM_WORLD は，通信可能な全てのプロセスを含む初期コミュニケータでありシステムで定義されている．

(お) rank0 にファイルの入出力ファイルの名前を読み込む．

(か) ファイル名を全ての rank にブロードキャストする．

(き) 各ランク毎のファイルオープン．(files.f; 後述)

(く) rank 間の座標データの送受信．(bookkeep.f; 後述)

(け) 各ランクの原子数から，通信によって送られてきた原子数を引き，座標を求める原子の数を決定する．

(こ) 各ランク間の座標データを送受信して，正確な原子間の相互作用を求めて，(け) で求めた原子数分の座標を求める．

(さ) MPI プロセスの終了

- サブルーチン FILES

(し) 各ランクでファイル"input.dat" をオープンする．

(す) rank0 で，FILE4 に割り当てられた出力ファイルをオープンし，rank1 で，FILE10 に割り当てられた出力ファイルをオープンする．

(せ) rank0 で，座標データファイル"in0.dat" をオープンし，rank1 で，座標データファイル"in1.dat" をオープンする．

- サブルーチン BOOKKEEP

(そ) MPI ライブラリを使用するために，ヘッダファイル (mpif.h) を読み込む．

(た) 全てのプロセスが呼出を完了するまでブロックする．

(ち) rank 間の送受信．

データの送受信には，次の 3 つの情報を指定する必要がある．

- メッセージバッファ
- 通信相手
- 通信コンテクスト

メッセージバッファは，バッファの先頭アドレス，データの個数，単位データ型の 3 つの値で指定される．これによって，送信の際には送るべきデータの所在が示され，受信の際には受け取ったデータをどこに置くかが決められる．

通信相手には，プロセスを指定できれば良いから，プロセスグループ，ランクの対を指定すれば良い．

通信コンテクストは，転送されるデータが何に関するものかを示すもので，会話で言えば，用件や話題に当たる．通信コンテクストの指定は，タグとコミュニケータを指定するば良い．

(つ) 非ブロッキング操作が完了するまで待機する．MPI_WAIT は，REQUEST(ここでは，IV11,IS11) によって識別された操作が完了した時点で戻る．

4.2.2 基礎的な MPI/SX の関数

ここでは，前節で用いた基礎的な MPI の関数の仕様をまとめる．

- MPI_INIT(INTEGER IERROR)

INTEGER IERROR　FORTRAN の戻りコードを指定する．

MPI を初期化する．全てのプログラムでは，必ずこの関数を呼び出してから，他の関数を呼び出すようにする．

- MPI_FINALIZE(INTEGER IERROR)

INTEGER IERROR	FORTRAN の戻りコードを指定する.

この関数が最後の MPI の呼び出しであることを確認する. この関数の後に行なわれる MPI 呼び出しは, 全てエラーとして示される.

- MPI_COMM_SIZE(INTEGER COMM,INTEGER SIZE,INTEGER IERROR)

COMM	連絡機構 (ハンドル) を指定する.
SIZE	整数で COMM のグループ内のプロセス数を指定する.
IERROR	FORTRAN の戻りコードを指定する.

コミュニケータに関わるグループ中のプロセスの個数を得る. SIZE は, MPI_COMM_SIZE の戻り値です. ここで, ランク及びサイズの情報を使用して, 利用可能なプロセス全体の作業を分割できます.

- MPI_COMM_RANK(INTEGER COMM,INTEGER RANK,INTEGER IERROR)

COMM	連絡機構 (ハンドル) を指定する.
RANK	整数で, COMM のグループ内の呼び出しプロセスのランクを指定します.
IERROR	FORTRAN の戻りコードを指定する.

コミュニケータに関わるグループ中の自プロセスのランクを得る. COMM が相互連絡機構である場合, RANK はローカルグループ内のローカルプロセスのランクとなる.

- MPI_BARRIER(COMM,INTEGER IERROR)

COMM	連絡機構 (ハンドル) を指定する.
IERROR	FORTRAN の戻りコードを指定する.

このルーチンは, 全てのプロセスが呼出を完了するまでブロックする.

- MPI_ISEND(CHOICE BUF,INTEGER COUNT,INTEGER DATATYPE,INTEGER DEST,INTEGER TAG,INTEGER COMM,INTEGER IERROR)

BUF	送信バッファの初期アドレスを指定する.
COUNT	送信バッファ内の要素数を指定する.
DATATYPE	各送信バッファ要素のデータタイプを指定する.
DEST	COMM 内の宛先タスクのランクを指定する.
TAG	メッセージタグを指定する.
COMM	連絡機構 (ハンドル) を指定する.
IERROR	FORTRAN の戻りコードを指定する.

この関数は 1 対 1 の非ブロッキング送信である. これによって, データタイプのカウント要素が, バッファから DEST で指定されたタスクに送信される.

- MPI_IRECV(CHOICE BUF,INTEGER COUNT,INTEGER DATATYPE,INTEGER SOURCE,INTEGER TAG,INTEGER COMM,INTEGER STATUS,INTEGER IERROR)

BUF	送信バッファの初期アドレスを指定する.
COUNT	送信バッファ内の要素数を指定する.
DATATYPE	各送信バッファ要素のデータタイプを指定する.
SOURCE	COMM 内のソースタスクのランクを指定する.
TAG	メッセージタグを指定する.
COMM	連絡機構 (ハンドル) を指定する.
STATUS	ステータスオブジェクトを指定する.
IERROR	FORTRAN の戻りコードを指定する.

この関数は1対1の非ブロッキング受信である．受信バッファは，アドレスバッファで始まり，データタイプによって指定されたタイプの連続要素をCOUNTのためのスペースを含む記憶領域である．

- MPI_WAIT(REQUEST,STATUS,IERROR)

REQUEST	待機要求(ハンドル)を指定する．
STATUS	ステータスオブジェクトを指定する．
IERROR	FORTRANの戻りコードを指定する．

非ブロッキング操作が完了するまで待機する．

4.3 MPI，指示行による並列化を用いない場合(MPI ×，並列化指示行 ×)の実行手順

MPI，指示行による並列化を用いないでコンパイル，実行する．この方法で求めた計算時間をMPIや指示行による並列化を用いて求めた計算時間と比較するための基準とする．

SX-4には，f77sx，f77，f90の3つのFORTRANコンパイラーが用意されている．ここでは，f77を使用し，以下のオプションをつけてコンパイルした[11]-[14].

```
% f77 -G local -o mdemo3d mdemo3d.f
```

ここで，用いているオプションの意味は次の通りである．

-G local	ローカルコモン領域に割り付けることを指定する．
-o mdemo3d	実行ファイル名を‘‘mdemo3d’’とする．

スクリプトファイル(run.nqs)を作成し,qsubコマンドを用いてサブミットする．-G localをつけることで，計算時間がわずかに短くなった．実行すると，stdout.datに計算開始時刻と計算終了時刻の情報が入る．stderr.datにシステム時間，ベクトル演算率等のプログラム情報が記録される．計算時間の比較は，表5にまとめる．

4.4 指示行による並列化のみを用いる場合(MPI ×，指示行 ○)の実行手順

指示行による並列化のみを用いる場合のコンパイルの方法を，以下で示す．プログラムの実行は，NQSジョブによって行なう．

```
% f77 -P auto -C vopt -pi -Wo'-l auto.lst' -o  mdemo3ds mdemo3ds.f
```

詳細については，昨年度のモニター報告会レポート[15]を参考にして頂きたい．

4.5 MPIのみを用いる場合(MPI ○，指示行 ×)の実行手順

ワークステーションで用いたプログラムでは，それぞれのマシンに同じ名前のデータファイルを出力していたが，SX-4では区別するために異なる名前のデータファイル(例えば，rank0では“energy0.dat”，rank1では”energy1.dat2”など)を出力するようにする．次の通り，コンパイルした．

```
% f77 -o ./mdemo3d2 mdemo3d2.f -G local -I./ -I/usr/include -lmpi -lt
```

ここで，新たに加わったオプションの意味は次の通りである．

-I./ -I/usr/include	インクルードファイルの指定．
-lmpi	mpiライブラリを読み込む．

a. シェルスクリプトの作成　以下のようなスクリプトファイル (runmpi.nqs) を作成した.

```
#!/usr/bin/sh
#@$-q p8
#@$-lt 8:00:00
#@$-o stdout.dat
#@$-e stderr.dat
#@$
echo 'BEGIN '; date
cd /home/ccns01/user5/x60440/Debris/Lcase5
setenv F_FILEINF YES
setenv F_PROGINF DETAIL
src/mdemo3d2 < stdin.dat
mpisx -p 2 -e src/mdemo3d2 < stdin.dat > stdout.dat
echo 'END ';date
```

作成したスクリプトファイル "runmpi.nqs" を使って，nqs 実行する.

なお，MPI を用いて得られた結果は，MPI を用いない場合の結果とポテンシャルエネルギーを比較し，良い一致を得ている [16].

4.6　MPI を使用しない場合とする場合の実行結果の比較 (指示行なし)

MPI の使用のしない場合とする場合の実行後の，プログラム情報を以下にまとめる.

表 5: MPI を使用しない場合とする場合の実行結果の比較

	MPI なし	MPI なし (指示行有り)	MPI 使用	
			rank0	rank1
経過時間 (秒)	125.184703	137.324530	68.440039	68.982707
ユーザ時間 (秒)	121.945182	487.047303	65.457567	65.378983
システム時間 (秒)	2.824707	4.641582	1.892774	1.777274
ベクトル命令実行時間 (秒)	13.103370	6.603979	4.602956	4.655420
全命令実行数	7302534965.	35250302407.	4077324917.	4106371453.
ベクトル命令実行数	111208697.	55958168.	39191025.	39250876.
ベクトル命令実行要素数	28062176338.	14120541275.	9779154745.	9712507215.
浮動小数点データ実行要素数	15394117581.	19842120911.	5809010022.	5772065521.
MOPS 値	289.093033	101.252764	211.087722	210.765405
MFLOPS 値	126.238013	40.739618	88.744668	88.286255
MOPS 値 (実行時間換算)	×	381.780380	×	×
MFLOPS 値 (実行時間換算)	×	153.611478	×	×
平均ベクトル長	252.337965	252.341022	249.525363	247.446890
ベクトル演算率 (%)	79.601101	28.633426	70.774774	70.484540
メモリ使用量 (MB)	25.031250	59.000000	35.031250	35.031250
最大同時実行可能プロセッサ数	1.	8.	1.	1.
1 台以上で実行した時間 (秒)	×	129.170822	×	×
2 台以上で実行した時間 (秒)	×	54.463822	×	×
3 台以上で実行した時間 (秒)	×	52.105028	×	×
4 台以上で実行した時間 (秒)	×	51.648232	×	×
5 台以上で実行した時間 (秒)	×	51.580740	×	×
6 台以上で実行した時間 (秒)	×	51.497951	×	×
7 台以上で実行した時間 (秒)	×	51.397984	×	×
8 台以上で実行した時間 (秒)	×	45.183254	×	×
イベントビジー回数	×	0.	×	×
イベント待ち時間 (秒)	×	0.00000	×	×
ロックビジー回数	×	0.	×	×
ロック待ち時間 (秒)	×	0.000000	×	×
バリアビジー回数	×	0.	×	×
バリア待ち時間 (秒)	×	0.000000	×	×

経過時間を比較すると，MPI/SX[17] を用いた方が，用いない場合 (指示行なし) よりも，45%程度短縮されていることが確認できる．また，指示行を入れると，経過時間は増加していることが確認できる．ただし，これは並列化率が低いために，並列処理性能が出ていないなどが原因と考えられる．そのため，能率の良い並列化計算を行なうためのプログラムを構築する必要がある．

5 コンパイル，実行時の注意 (気をつける) 点

最も効率的な計算方法を見つけるために，さまざまなコンパイル方法，実行方法を試してみた．その時自分が苦労した点，困った点を以下で紹介する．

鉄則 1 ワークステーションで使っていたインクルードファイル (mpif.h) などを，コンパイルするディレクトリに置かない．

【説明 (失敗例)】

インクルードファイルの検索順序は，入力ソースファイルが格納されているディレクトリ，オプションで指定されたディレクトリ，カレントディレクトリの順である．

mpich 用のインクルードファイルをソースファイルと同じディレクトリ内に置いていたので，mpich 用のヘッダーファイルが読み込まれてしまい，計算実行時に次のようなエラーが表示された．

```
0 - MPI_COMM_RANK : Invalid communicator: Null communicator
[0]  Aborting program !
[0] Aborting program!
1 - MPI_COMM_RANK : Invalid communicator: Null communicator
[1]  Aborting program !
[1] Aborting program!
ccsx40: mpid: MPI process terminated by exit(5)

%NQS(INFO):  The request received a signal SIGTERM (software termination \
signal).
```

鉄則 2 -thread オプションは，(現在の OS のバージョンでは，) 使わない．

【説明 (失敗例)】

現在の OS では，thread がサポートされていないようなので，オプション-thread をつけて実行すると次のようなエラーがでる．

```
% mpisx -p 2 -thread -e src/mdemo3d2
```

```
mpisx: warning: [-thread] is obsolete.
%NQS(INFO):The request received a signal SIGTERM (software termination signal)
```

鉄則 3 コンパイル時の-o オプション，ソースファイルの位置は，f77 の直後に置く．

```
% f77 -o ./mdemo3d2s mdemo3d2s.f -P multi -G local -I/usr/include -lmpi
```

【説明 (失敗例)】

次のような -o オプション，ソースファイルの位置で，コンパイルすると次のようなエラーが表示される．

```
% f77 -P multi -G local -I/usr/include -lmpi -lt -o mdemo3d2s mdemo3d2s.f
```

```
未定義シンボルが発生しました.
シンボル名                              ファイル名
mpi_dup_fn_                             ./mdemo3d2s.o
mpi_null_copy_fn_                       ./mdemo3d2s.o
mpi_null_delete_fn_                     ./mdemo3d2s.o
mpi_init_                               ./mdemo3d2s.o
mpi_bcast_                              ./mdemo3d2s.o
mpi_wtime_                              ./mdemo3d2s.o
mpi_wtick_                              ./mdemo3d2s.o
mpi_comm_rank_                          ./mdemo3d2s.o
mpi_comm_size_                          ./mdemo3d2s.o
mpi_finalize_                           ./mdemo3d2s.o
mpi_wait_                               ./mdemo3d2s.o
mpi_isend_                              ./mdemo3d2s.o
mpi_irecv_                              ./mdemo3d2s.o
mpi_barrier_                            ./mdemo3d2s.o
ld fatal: シンボル参照でエラーが発生したため，ファイル
mdemo3d2s に書き込まれません.
f77 fatal : ld コマンドでエラーが検出された : 13
```

6 残された問題点 (MPI/SX，指示行による並列化を両方用いた場合の実行不可例)

MPI/SX を用いることで，rank0 と rank1 に分けられたそれぞれの計算おいて，並列化指示行を挿入することによる並列化を試みた．

コンパイル時の引数を変えて計算を実行してみたが，エラーメッセージが出力され，計算結果ファイルは出力されなかった．また，core が生じる場合と生じない場合があった．

現在，原因を検討中であるが，この問題点について報告する．

6.1 利用した並列化指示行

並列化指示行は，昨年度 SX-4 モニター報告会で高速化の効果のあったものをそのまま挿入した．指示行を含むソースファイルの抜粋を以下に示す．

- メインルーチン MDMAIN

```
*odirf concur,inner,cncall
C ***********************************************************************
      PROGRAM MDMAIN
C ***********************************************************************
C---+*---1----+----2----+----3----+----4----+----5----+----6----+----7--
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
                :(中略)
      CALL FORCE(0)
                :(中略)
      DO 10 I=1,NSTEP
        CALL SOLVE      (このサブルーチンで，FORCE が call される)
                :(中略)
```

```
   10 CONTINUE
                :(中略)
      CALL OUTPUT(0)
C
      STOP
      END
```

- サブルーチン FORCE

```
      SUBROUTINE FORCE(IE)
C ***************************************************************************
C---+*---1----+----2----+----3----+----4----+----5----+----6----+----7--
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
C
*VDIR novector
*PDIR PARDO FOR
*odirl concur,inner
      DO 20 K=1,KKMAX
                :(BIGLOOP ; 相互作用力を計算する大きなループ ; 中略)
   20 CONTINUE
      RETURN
      END
```

6.2 コンパイルオプションと実行結果

コンパイル時の引数を変えて計算を実行してみたが，データファイルは出力されなかった．また，core が生じる場合と生じない場合があった．

【エラー 1】

```
% f77 -o ./mdemo3d2s mdemo3d2s.f -P auto -G local -I/usr/include -lmpi -lt
```

ジョブを実行した結果，上と同じエラーが生じる．core は生じない．

【エラー 2】

```
% f77 -o ./mdemo3d2s mdemo3d2s.f -P multi -G local -I/usr/include -lmpi -lt
```

NQS ジョブを実行した結果，次のようなエラーが生じ，core をはく．

```
  *** 152 順番探査入出力文の実行でファイル終了記録を検出した UNIT=5 REC=5 PROG=mdmain\
 ELN=79(8600199c) TASKID=1
```

エラー発生時のコンパイルオプションと実行結果を表 6 としてまとめる．

表 6: コンパイルオプションと実行結果

コンパイル時の引数				コンパイルの結果	実行結果
-P multi	-G local		-I/usr/include -lmpi -lt	○	エラー (core が生じる)
-P multi			-I/usr/include -lmpi -lt	○	エラー (core が生じる)
-P auto	-G local		-I/usr/include -lmpi -lt	○	エラー
-P auto		-pi	-I/usr/include -lmpi -lt	○	エラー

参考文献 [14] によれば，MPI/SX を用いる場合の問題点として，“並列化は手動で行なわなければならない” とあるので，`-P auto` オプションが利用できない (エラー 1) のは，当然かも知れない．しかしながら，`-P multi` オプションを入れても，エラーとなった (エラー 2) ことから，MPI/SX を利用しない場合，効率があがった並列化指示行であっても，MPI/SX を利用する場合は，データ通信との兼ね合いなど，十分に検討する必要があったものと思われる．

今後の課題として，MPI/SX を利用する場合に，並列化指示行をプログラムに挿入する際の注意点，コンパイル方法などを検討する予定である．

7 おわりに

大阪大学計算機センターの SX-4 マシンを使って，MPI/SX を使った，並列分子動力学シミュレーションを行なった．MPI を用いると，計算時間は約半分に短縮された．もちろん，モデル化が異なっているので，当然ではあるが，WS で，実行した MPI プログラムが，何一つ変更することなく，簡単に SX-4 上で実行できることは，非常にユーザーにとってありがたいことである．

今後，プログラムに挿入する並列化指示行，および，その際のコンパイル方法を調べ，計算をより能率的にできる方法を見つけたい．これが達成できれば，MPI/SX と，SX-4 の並列化機能 (指示行など) をうまく使って，最小のプログラムの変更で，多数の CPU を使った並列計算が実施できるはずであり，SX-4 上での計算に大きく期待できる．

参考文献

[1] 北川浩，中谷彰宏，尾方成信，土橋謙祐，日本機械学会講演論文集 (関西支部第 73 期定時総会講演会)，No.984-1(1998-3),(10-3)-(10-4).

[2] 日置慎治，大阪大学計算機センターニュース，28-1(1998)，125-134.

[3] 八坂哲雄，’宇宙のゴミ問題’，1997, 裳華堂.

[4] Piekutowski, A. J. *et al.*, *Int. J. Impact. Engng.*, **17** (1995)，627-638.

[5] 中西浩一ほか， 大阪大学計算機センターニュース，26-4(1997)，25-41.

[6] 湯淺太一ほか，’はじめての並列プログラミング’，共立出版.

[7] 例えば，`http://www.ppc.nec.co.jp/mpi-j`.

[8] 例えば，`http://www.ibm.co.jp/manuals`.

[9] 例えば，`http://www.nkk.co.jp/comp/stc/mpi`.

[10] 例えば，`http://www.mcs.anl.gov/mpi/mpich`.

[11] 大阪大学大型計算機センター速報，267,(1997),7-8.

[12] 超並列コンピューティング，`http://www.fuji-ric.co.jp/koike/massivepara.html`.

[13] 例えば，`http://www.center.osaka-u.ac.jp/j/tebiki/7-1.htm`.

[14] 例えば，`http://www.center.osaka-u.ac.jp/user-service/fortransx/sx.html`.

[15] 例えば，`http://www.center.osaka-u.ac.jp/user-service/sx4-8.1/g1af09e/intro.html` .

[16] 中谷敬子，大阪大学計算機センターニュース，28-1(1998)，98-106.

[17] 栢木良典ほか，第 22 回 NCP 研究会シンポジウム論文集，(1998)，27-30.

[18] 例えば，`http://www.center.osaka-u.ac.jp/center/cc-home-jp.html`.