JAERI-Data/Code
99-051

Paragon上での入出力プログラミングガイド

2000年1月

上島　豊・荒川拓也*・佐々木明・横田　恒*

日本原子力研究所
Japan Atomic Energy Research Institute

本レポートは、日本原子力研究所が不定期に公刊している研究報告書です。
入手の問合わせは、日本原子力研究所研究情報部研究情報課（〒319-1195　茨城県那珂郡東海村）あて、お申し越し下さい。なお、このほかに財団法人原子力弘済会資料センター（〒319-1195　茨城県那珂郡東海村日本原子力研究所内）で複写による実費頒布を行っております。

This report is issued irregularly.
Inquiries about availability of the reports should be addressed to Research Information Division, Department of Intellectual Resources, Japan Atomic Energy Research Institute, Tokai-mura, Naka-gun, Ibaraki-ken 〒319-1195, Japan.

編集兼発行　日本原子力研究所

JAERI－Data/Code 99－051

Paragon上での入出力プログラミングガイド

日本原子力研究所関西研究所光量子科学研究センター
上島　豊※・荒川　拓也*・佐々木　明・横田　恒*

（1999年12月1日受理）

日本で100並列を越える並列計算が、実際に行われるようになったのは、つい数年ほど前からである。日本原子力研究所（原研）のIntel製 Paragon XP/S 15GP256 <那珂研究所>、75MP834 <関西研究所>は、本格的超並列計算機の先駆けとして光量子、核融合の大規模、超並列計算を目的に導入されている。これらの計算機を使って超並列計算を行うために、多くの超並列計算プログラムが移植や新規作成されている。これらのプログラムは、従来、ワークステーションやベクトル計算機上で動作していたものをそのまま移植したものか、並列用にアルゴリズムの変更を施したものである。

異なる計算機及びオペレーティングシステム（OS）の基でのプログラム開発には、細心の注意とノウハウが必要であるが、Paragonに到ってはユーザ数が極めて少ないため、ノウハウの集積と環境の標準化が大変困難な状況にある。原研関西研究所におけるParagon XP/S 75MP834上での超並列計算プログラム開発において得た情報をParagon上でのスカラー超並列プログラム開発ガイドとしてJAERI-Data/Code 98-030にまとめた。本報告書では、超並列計算機が持つ特殊性が際だつ入出力周りにテーマを限定して、より高速、安定に入出力が実行できる方法をfrotranとcのプログラム実例入りで解説する。

関西研究所：〒572-0019　大阪府寝屋川市三井南町25-1
※　博士研究員
＊　財団法人　高度情報科学技術研究機構

JAERI－Data/Code 99－051

Guide to the Programming of I/O on the Paragon

Yutaka UESHIMA※, Takuya ARAKAWA*, Akira SASAKI and Hisasi YOKOTA*

Advanced Photon Research Center
Kansai Research Establishment
Japan Atomic Energy Research Institute
Mii-mimami-cho, Neyagawa-shi, Osaka

(Received December 1, 1999)

Real parallel calculations with the use of over hundreds computers were performed in Japan since several years ago. The Intel Paragon XP/S 15GP256 < Naka Fusion Research Establishment >、75MP834 < Kansai Research Establishment > were introduced as pioneers in Japan Atomic Energy Research Institute (JAERI) for the purpose of the massive parallel calculations of advanced photon and fusion researches. Recently, a lot of parallel programs have been transplanted and newly produced to perform the parallel calculations with the computers. These programs are almost instant, therefore there are some seeds of trouble in the massive parallel computing.
When programs are developed under different computer and operating system (OS), prudent directions and knowledge are needed. However, integration of knowledge and standardization of environment are quite difficult because number of Paragon system and Paragon's users are very small in Japan. Therefore, we published 'Guide to Development of a Scalar Massive Parallel Programming on Paragon' in JAERI-Data/Code 98-030. In this report, the method of stable and faster I/O is shown with sample programs by fortran and c.

Keywords: Massive Parallel, Paragon, Guide, I/O, Integration of Knowledge

---

※ Post-doctoral Fellow

* Reattach Organization for Information Science & Technology

目　　　次

Contents

This is a blank page.

1. はじめに

ここ４、５年前までは、最も高速な計算機といえば、いわゆるスーパーコンピュータと呼ばれるベクトル型計算機であった。この型の計算機は４、５年で計算速度が１０倍ずつ高速になってきた。しかし、現在、1CPUの演算速度が限界に達してきている。今まで使われていたベクトル計算機の1CPU性能は、数GFLOPSで搭載メモリも数Gbyteであったが、将来、格段に速い単体の計算機は出てきそうにない。これにかわって、より高速な計算機として並列計算機が次々と導入されてきている。

日本原子力研究所（原研）には、Intel製 Paragon XP/S 75MP834 <関西研究所>1, 2)、15GP256 <那珂研究所>が、本格的超並列計算機の先駆けとして光量子、核融合の大規模、超並列計算を目的に導入されている。これらの計算機を使って超並列計算を行うために、多くのプログラムの移植や新規作成が行われている。これらのプログラムは、従来、ワークステーションやベクトル計算機上で動作していたプログラムをそのまま移植したものか、並列用に多少のアルゴリズム変更を施したものである。一般的に、異なる計算機及びオペレーティングシステム（OS）の基でプログラムを開発するには、細心の注意とノウハウが必要であるが、従来、ワークステーションからベクトル計算機、異なるメーカの計算機へのプログラムの移植は、ユーザ数の多さが幸いして、多くのノウハウと標準化された環境のおかげで、比較的短時間で作業を完了することができていた。しかし、超並列計算機に関しては、ユーザ数が極めて少ないため、ノウハウの集積と環境の標準化が大変困難な状況にある。そのため、原研関西研究所におけるParagon XP/S 75MP834上での超並列計算プログラム開発において得た情報をParagon上でのスカラー超並列プログラム開発ガイドとしてJAERI-DATA/CODE 98-030にまとめた。本報告書では、超並列計算機が持つ特殊性が際だつ入出力周りにテーマを限定して、より高速に安定に入出力を実行できる方法をfrotranとcのプログラム実例入りで解説する。

関西研究所で稼働中の分散メモリスカラ超並列計算機Paragon XP/S 75MP834は、システム全体で120GFLOPS、100Gbyteの性能を有している。この数字は、現有の単体の最高速計算機の約100倍の演算速度と搭載メモリの性能を意味するものである4, 5)。実際、この性能がどの程度のものかを理解してもらうために、シミュレーションしている系（レーザー・プラズマ相互作用）で例を挙げる。従来の単体ベクトル計算機では、典型的なシミュレーションで粒子数が数千万粒子、空間格子が1,000×1,000格子のシミュレーションが限界であった。しかし、このParagon上で並列化されたプログラムを使うと、粒子数が数十億粒子、空間格子が1,000×100,000格子または、10,000×10,000格子のシミュレーションが実行できる5, 6)。このシステムサイズの増大がもたらす波及効果は、絶大なものである。もし、空間格子を0.1μm（レーザー波長の1/10）とすると、100μm（レーザー広がり方向）×10cm（レーザー伝搬方向）という、実際の実験に近いレーザーの伝播を計算する

ことができる。また、２次元計算が困難とされていた固体（電子密度~$10^{23}/cm^3$）との相互作用では、初期温度100eVで5μm（レーザー広がり方向）×1μm（固体の厚さ）のレーザーの反射（高調波発生）吸収のシミュレーションも行うことができる。

2.Paragonのディスクシステムとその動作

Paragon-S120MPは、通常のシングルプロセッサのWSとは異なり、800の計算ノード（PE）を持っている。ディスクは、PCクラスターのように各計算ノードごとに持っておらず、システム全体でI/Oのための専用ノード（以下、I/Oノード）が25個設けられている。実際には、1つはシステムが使っているためユーザは24のI/Oノードを使うことができる。このように、ディスクおよびI/Oノード数を制限することで、Paragon-S120MPは長時間の動作安定性を手に入れている。例えば、一つのディスクのトラブルが0.1回/年の発生率としても800台のディスクがあれば、80回/年もトラブルが発生することになり、現実的な使用に耐えられない状況になってしまう。

ParagonでのディスクディレクトリとParagonのI/Oノードの関係について触れておく。各I/Oノードには、一つのraid3のディスクがついている。raid3ディスクは、内部が6つのパーティションにより区切られており、それぞれのパーティションをディスクディレクトリと一対一に対応させている。

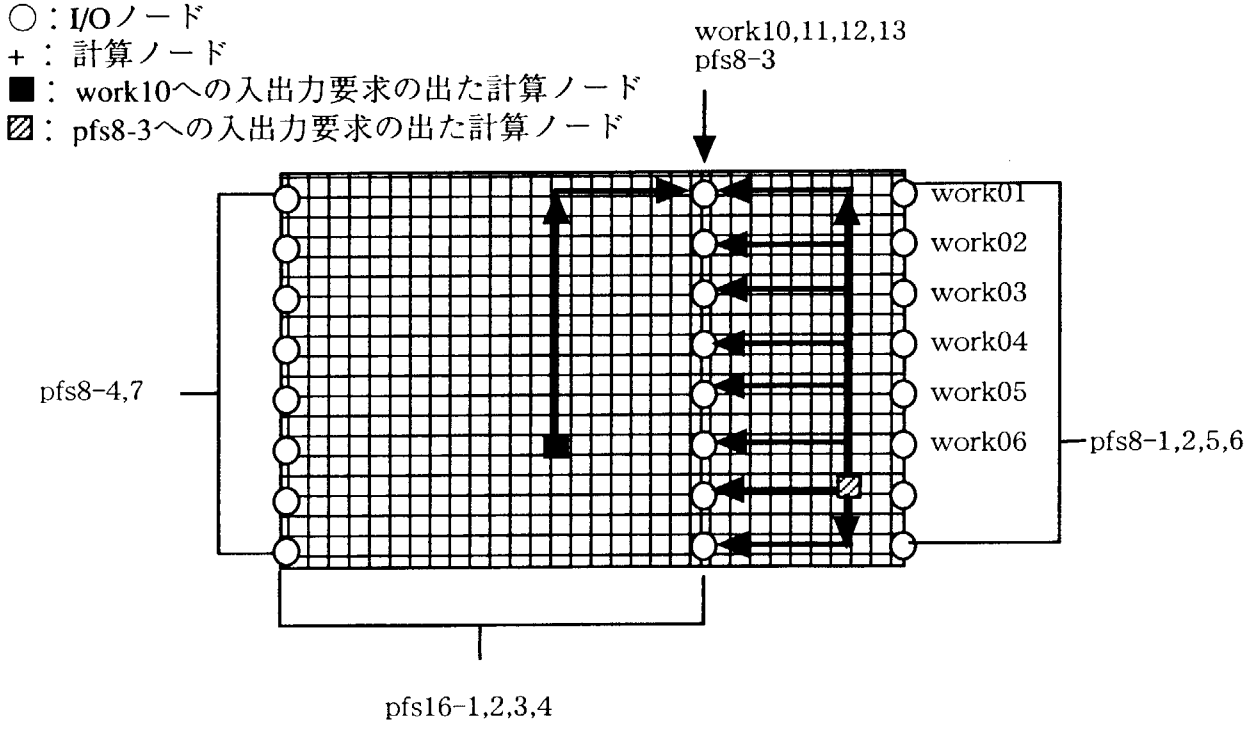

Fig.2.1 ディスクディレクトリーとParagonのI/Oノードの共有関係

ディスクディレクトリは、特性上、大きく2種類に分けられる。一つは、work??で通常のUNIXでつかわれるUNIX File System（ufs）ファイルを格納できるディレクトリであり、もう一方がpfs?-?でufsファイルの他に高速入出力が可能だが他のファイルシステムと互換性のないParallel File System（pfs）ファイルを格納できるディレクトリである。pfsデ

ィレクトリは、Fig.2.1にあるように物理的には一つのディレクトリが複数のI/Oノードにまたがっており、それをソフトウェアー的にリンクして一つのディスクのように見せている。例えば、pfs8-?は8個のI/Oノードをもっており、pfs16-?は16個のI/Oノードをもっている。この様に複数のI/Oノード配下のディスクをあたかも一つのディスクとして使えるようにしたのがpfsディスクシステムである。

例えば、Fig.2.1の真ん中上部のI/Oノード配下にあるディスクは、work10,pfs8-3,pfs16-1,2,3,4の6つのディレクトリを持つことになる。ここでは、計算ノードからあるディスクディレクトリ（work10やpfs8-3)に入出力したときにどのI/Oノードが使われるのかを示す。Paragonのノード構成の全体図は、次のような配置をしており、計算ノードの両端と真ん中に3列のI/Oノード列とその配下に2Gごとにパーティション化されたディスク=ディスクディレクトリがある。例えば、ある計算ノードからwork10、もしくは、pfs8-3へ入出力すると、使用されるI/Oノードとデータ送信経路はFig.2.1のようになる。

workディレクトリへの入出力（Fig.2.2）

実際にある計算ノード（ノード0）からあるディスクディレクトリ（work01）に対して入出力を行う場合、次のように動作する。

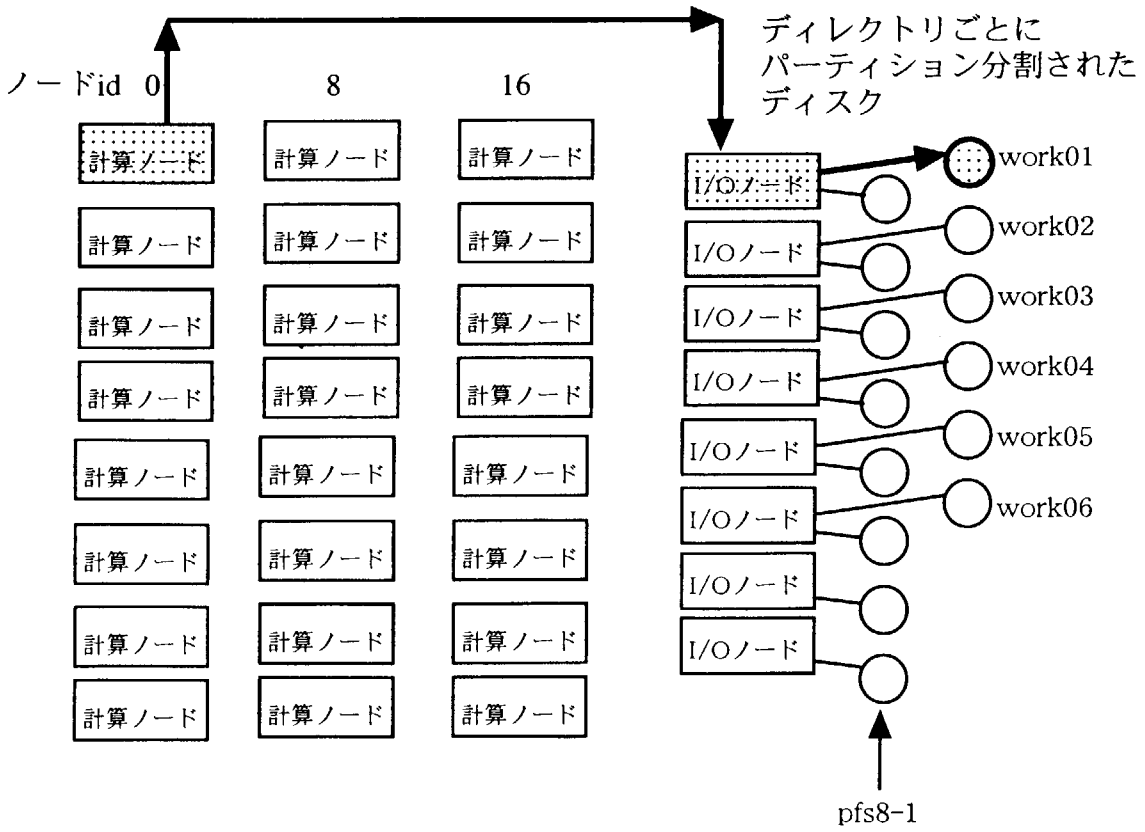

Fig.2.2 workディレクトリへの入出力

ノード0からwork01をパーティションに持つディスクとつながっているI/Oノード(ハッチしているI/Oノード)に対してデータ転送が行われ、I/Oノードのメモリにデータが置かれる。以下、このような動作をI/Oノードが受け付けることをI/O-requestと呼ぶ。その後、I/Oノードメモリからその配下のディスクのwork01のパーティションの部分に書き込まれ、データの出力が完了したことになる。

pfsディレクトリへの入出力 (Fig.2.3)

ある計算ノード（ノード0）からあるディスクディレクトリ（pfs8-1）に対してI/Oを行う場合、次のように動作する。

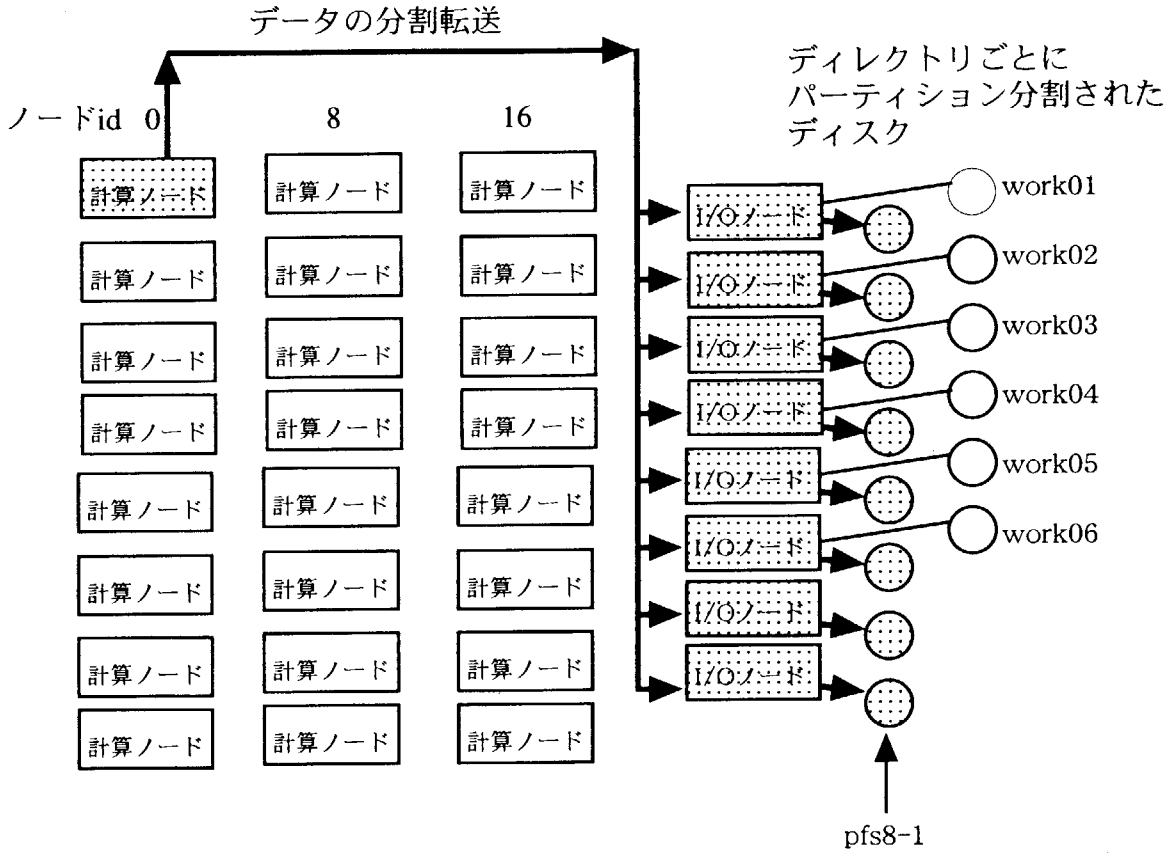

Fig.2.3 pfsディレクトリへの入出力

データは、分割されノード0からpfs8-1を配下に持つ全てのI/Oノード(ハッチしているI/Oノード)に対してデータが転送され、それぞれのI/Oノードのメモリに分割されたデータが置かれる。このとき、pfs8-1に携わる8個のI/Oノードは、それぞれ１つのI/O-requestを受けたことになる。その後、I/Oノードメモリからその配下のディスクのディスクディレクトリに書き込まれ、データの出力が完了したことになる。

後述するように使用モードによって複数のI/Oノードを並列で動作させることができ、

高速な入出力が可能である。また、I/Oノードは、多くのディスクディレクトリによって共有されているので、異なるディスクディレクトリに入出力していてもI/Oノードの負荷が増大する可能性がある。

このように並列計算機では、プログラムを作成・実行する場合、注意しなければならない点がいくつかある。例えば、通常のWSで全く問題がない次のようなプログラムを複数ノードで実行した場合に予期しない事態が生じる場合がある。

プログラミング時に、意識しなければならない事柄は次の通りである。
キーワード：論理機番、I/O-request、I/Oノードメモリ
◎各ノードで、利用できる論理機番は61個である。
◎I/Oノードが同時に受け取れるI/O-request数は、32以下である。
◎I/Oノードのメモリは、64Mbyteである。

例えば、次のプログラムを800ノードで実行した場合を考えてみる。

間違ったfortranプログラム

```
c     mymatrix   ;出力配列
c     ibuffsize  ;出力配列の大きさ
c     idatatype  ;出力配列変数の型のサイズ
c     iid        ;論理機番
      parameter (ibuffsize=70000000,iid=10,idatatype=4)
      real*4 mymatrix(ibuffsize/4)
      open(iid,'/work01/mydata')
      write(iid) mymatrix
      close(iid)
```

間違ったcプログラム

```
      int ibuffsize=70000000;
      int idatatype=sizeof(float);
      FILE *iid = fopen( "/work01/mydata", "w");
      fwrite(mymatrix, ibuffsize, 1, iid);
      fclose( iid );
```

なぜ、「このプログラムがいけないのか?」と言うと、
800ノードで実行されているということは、実際にはこのプログラムが800個同時に実行されていることになる。しかし、/work01/mydataというファイルは一つしかないため重複して同一ファイルをオープン、そして入出力することになる。その結果、I/Oノードに非常

に多くのI/O-requestやI/Oノードメモリの占有が集中して、I/Oノードに過付加がかかる。

1)重複して同一ファイルをオープン、そして入出力してはいけない。（ただし、あとで説明するgopenを使えば可能）
2)800個のI/O-requestが/work01のあるノードに集中することになる。ちなみに、Paragonは、一つのI/Oノードが同時に受け取れるI/O-request数は、32以下である。
3)出力変数mymatrixは70Mbyteであるが、I/Oノードの搭載メモリは64Mbyteであり、メモリオーバーを引き起こす。

さらに簡単な例で間違ったプログラムと正しいプログラムを比較する。4ノードで次のプログラムを実行した場合を考えてみる。

間違ったfortranプログラム

```
open(iid,'/work01/mydata')
write(iid,*) "test"
close(iid)
```

間違ったcプログラム

```
iid = fopen( "/work01/mydata", "w");
fprintf(iid,"test");
fclose( iid );
```

4ノードで実行されているということは、実際にはこのプログラムが4個同時に実行されていることになる。しかし、/work01/mydataというファイルは一つしかないため重複して同一ファイルをオープン、そして入出力することになる。この様な入出力は、計算ノード情報（ファイルのある特定場所がどのノードから書かれたか）が失われており、出力したものを再度入力することさえできない。計算ノード数が増えてくるとこのプログラムによりシステムダウンするので、絶対に使ってはいけない入出力プログラムの例である。もし、全ノードの情報を出力したいのであれば、標準出力write(*,*)を使って、次のようにプログラミングしなければならない。

正しいfortranプログラム

```
write(*,*) "test"
```

正しいcプログラム

```
    printf("test");
```

ある特定のノード（例えば、0ノード）の情報を出力したいときは、次のようにプログラミングすればよい。

正しいfortranプログラム

```
c    iam          ;実行ノードid
      iam = mynode()
      if( iam .eq. 0 )
         open(iid,'/work01/mydata')
         write(iid,*) "test"
         close(iid)
       end if
```

正しいcプログラム

```
/*    iam          ;実行ノードid   */
      int iam = mynode();
      if( iam == 0 ){
          FILE *iid = fopen( "/work01/mydata", "w");
          fprintf(iid,"test");
          fclose( iid );
      }
```

3.入出力のサンプルプログラムと解説

上記の入出力動作条件により、次の4通りの入出力方が比較的簡単であり、推奨される。

(1)ASCII or binaryで単一ノードから単一ufsファイルを入出力する。
　(open,write)
(2)binaryで全ノードから各ノードごと異なるufsファイルを入出力する。
　(open, cwrite)
(3)binaryで全ノードから単一ufsファイルを入出力する。
　(gopen:M_SYNC, cwrite)
(4)binaryで全ノードから単一pfsファイルを入出力する。
　(gopen:M_RECORD, cwrite)

ちなみに、入出力速度は、(1)＜(2)＜(3)＜(4)の順番に高速になる。これらのプログラミングの例を示す前に、Paragonにおいて入出力で使うことができる関数及びsubroutineの特徴を簡単にまとめておく。

open　　：単一ノードごとに異なるファイルを開くことができる。
　　複数ノードで同一ファイルを開いてはいけない。

gopen　　：全ノードで同一binaryファイルを開くことができる。
　　optionにより作られるファイルの形式や I/O-request数が異なる。
　　以下の二つのオプションの使用を薦める。
　　M_SYNC:低速だがufs（UNIX FILE SYSTEM）ファイルを作る。
　　I/O-request=1
M_RECORD　　:高速だが他のファイルシステムと互換性のないpfs
　　（PARALLEL FILE SYSTEM ）ファイルを作る。
　　I/O-request=同時並列入出力ノード数

write(read)　　：単一ノード入出力のみ。
　　I/O-requestが完了するまで、次の実行文へ制御を渡さない。
　　ただし、標準入力、出力、エラー出力は、この限りではない。

cwrite(cread)　：全ノード入出力のみ。

I/O-requestが完了するまで、次の実行文へ制御を渡さない。
発行直後、I/Oノードメモリを解放するために全ノードで同期をとる。

iwrite(iread) ：全ノード入出力のみ。
I/O-requestの完了を待たずに、次の実行文へ制御を渡す。
直後、I/Oノードメモリを解放するための全ノード同期をとらない。

注） iwrite(iread)は、I/O-request数を制限することが難しいので、習熟してから利用するようにすべきである。したがって、ここでは、プログラム例は挙げない。

(1)単一ノードでの単一ufsファイル入出力（open,write）
○ASCII or binary formatのファイルをつくることができる。
○I/Oノードは、1つのI/O-requestを受け取る。
○目視 or ftp（解析プログラム用）summary dataに適している。

```
      inculude 'fnx.h'
c     iam                ;実行ノードid
      iam = mynode()
      if(iam .eq. 0) then              .....①
         open(iid,'/work01/mydata')    .....②
         write(iid,*) 'test'           .....③
         close(iid)                    .....④
      end if
```

プログラム説明
①0番ノード（iam=0）のみ、if文の中を通る。
②0番ノードで論理機番iid=10でファイル/work01/mydataを開く。
③0番ノードで出力する。
④0番ノードで論理機番iid=10のファイル/work01/mydataを閉じる。
すなわち、この場合、0番ノード以外は入出力を行わないことになる。

注）複数ノードでのデバックを行う場合は、このプログラムでなく、標準出力write(*,*)を利用する。

参考）cプログラム

```
     #inculude <nx.h>
/*    iam           ;実行ノードid    */
      int iam = mynode();
      if( iam == 0 ){                                    .....①
            FILE *iid = fopen( "/work01/mydata", "w"); ...②
            fprintf(iid,"test");                        .....③
            fclose( iid );                              .....④
      }
```

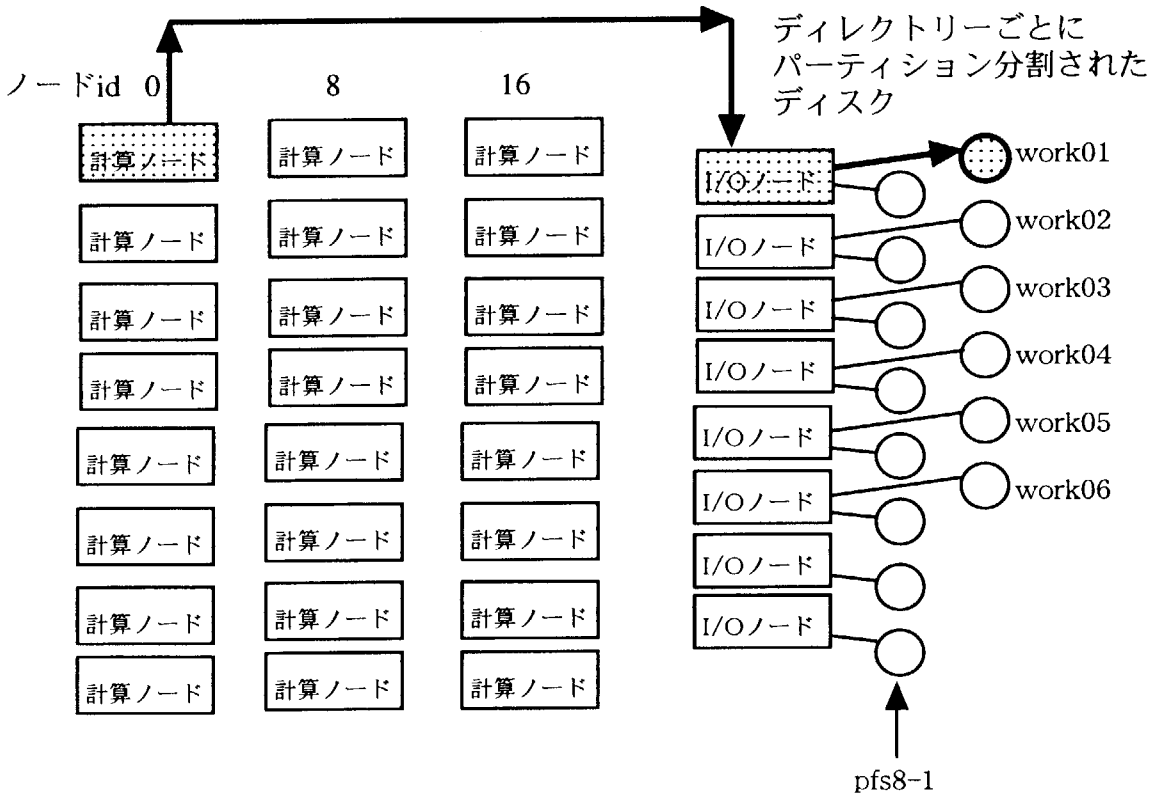

Fig.3.1 単一ノードでの単一ufsファイル入出力

(2)全ノードでの複数ufsファイル入出力（open,cwrite）

○binary formatのファイルをつくることができる。

○I/Oノードは、1つのI/O-requestを受け取る。

○Paragon内でしか利用しないrestart data等に適している。

○全ノード以外での入出力は不可能。

```
c     ibuffsize ;出力配列の大きさ（必ず、idatatypeの公倍数）
c     idatatype ;出力配列変数の型のサイズ
c     iid       ;論理機番
      parameter( ibuffsize=7000000, iid=10, idatatype=4  )
c     npfs      ;使用するpfsに含まれるI/Oノード数
c     iomemory  ;一つのI/Oノードのメモリ量
      parameter( npfs=8, iomemory = 64000000)
c     mymatrix  ;出力配列
      real*4 mymatrix(ibuffsize/idatatype)
      inculude 'fnx.h'

c     iam       ;実行ノードid
      iam = mynode()
c     num       ;実行ノード総数
      num = numnodes()
      write(nodeid,'(i3.3)') iam                    .....①
c     ibuffsize*num/npfs;一つのI/Oノード当たりの最終出力量
c     IDEV              ;mymatrixの入出力分割数
      IDEV     = ibuffsize*num/npfs/iomemory        .....②
      open(iid,'/pfs8-1/mydata.'//nodeid)           .....③
      jjj = 1
      do  1000  ijk = 1,  IDEV
         jjj = ( ijk - 1 )*ibuffsize/idatatype/IDEV+1 .....④
         call cwrite(iid,mymatrix(jjj),ibuffsize/IDEV)  ...⑤
 1000 continue
c     mymatrixを分割したときの残差の処理
      if ( ibuffsize .ne. (jjj-1)*idatatype ) then    .....⑥
         ibuff =  ibuffsize-(jjj-1)*idatatype
         call cwrite(iid,mymatrix(jjj+1),ibuff)  .....⑦
      end if
      close(iid)                                    .....⑧
```

プログラム説明

①write(nodeid,'(i3.3)') iamで各ノードで各自のノードidをnodeidに書式付き出力をする。例えば、iam=5の時は、nodeidは005となる。

②次に、mymatrixの入出力分割数IDEV = 一つのI/Oノード当たりの最終出力量/I/Oノードメモリを求める。

③論理機番iid=10で全ノードで異なるファイル/pfs8-1/mydata.???を開く。
④分割しても連続的に出力できるようにmymatrixの出力ポインター位置を計算する。
⑤全ノードでcwriteで同期をとりながら各ノードごとI/Oノードへデータを送信する。すなわち、I/O-requestを発行する。全ノードデータ送信が終わった段階で、I/Oノードメモリ内のデータをディスクへ完全に書き込む。

④～⑤をIDEV-1回繰り返す。

⑥ibuffsize/idatatypeがIDEVで割り切れないとき、このif文の中を実行する。
⑦割り切れなかった分の残りのデータは、ibuff = ibuffsize-(jjj-1)*idatatype Byteとなる。⑤と同様の手続きでディスクへ完全に書き込む。
⑧論理機番iid=10のファイル/work01/mydataを閉じる。

注）同一ファイルで実行する場合は、後で説明する(4)の入出力方法で行うようにする。全ノードcwriteを発行し終わったとき、すなわち、do 1000のループが一回回ったとき、一つのI/Oノード当たりの総出力量=ibuffsize/IDEV*num/npfsが64MByteを超えないようにIDEVを調整しなければならない。上のプログラムでは、IDEV = 70000000 *512/ 8/ 64000000=70となる。プログラム例の場合、ibuffsize=70000000、idatatype=4、IDEV=70となり、ibuffsize/idatatypeがIDEVで割り切れるので、⑦は実行されない。

参考）fopenが、cwriteとマッチングしないため対応するcによるプログラム例はない。

Fig.3.2と同様にして、計算ノード1, 2....., num-1において、出力ファイルがmydata.001, mydata.002,....mydata.???まで同様の作業を繰り返す。

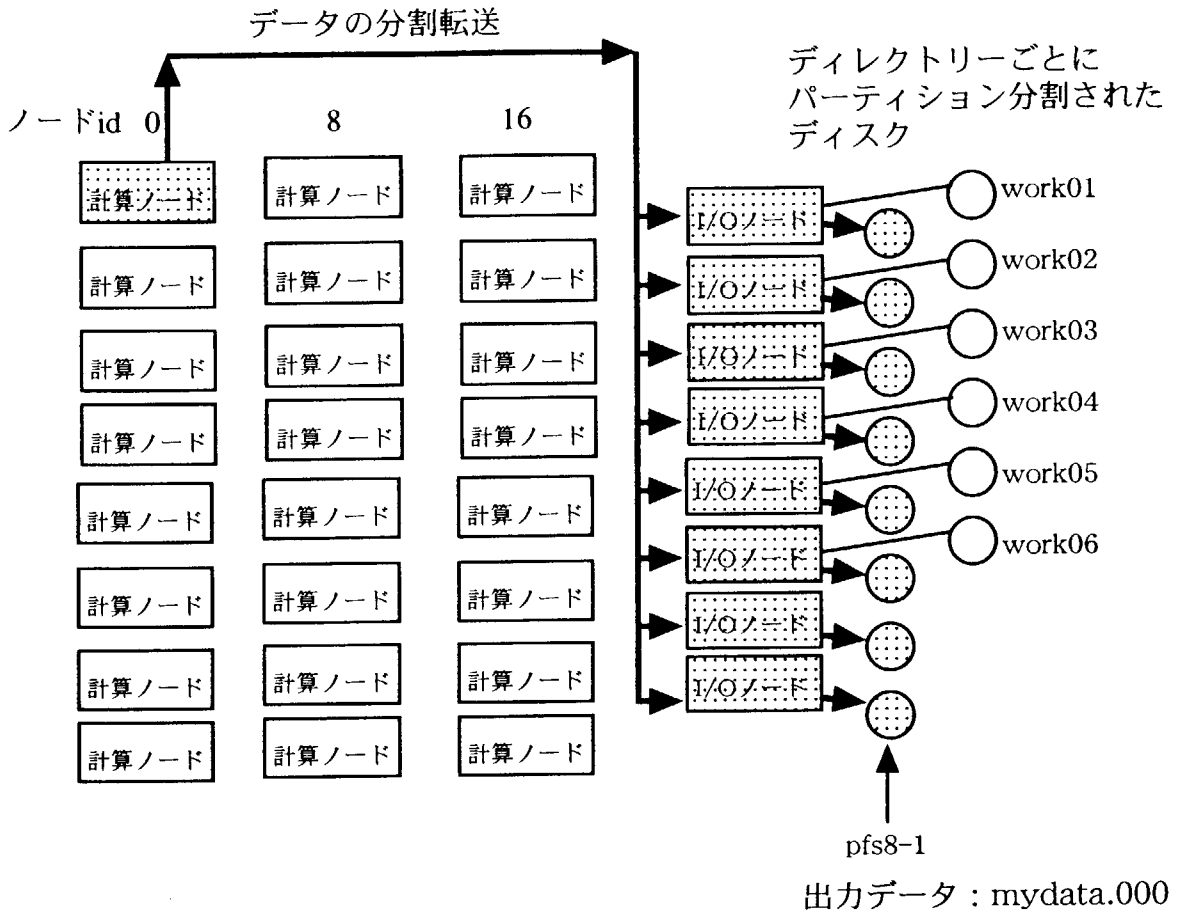

Fig.3.2 全ノードでの複数ufsファイル入出力

(3)全ノードでの単一ufsファイル入出力（gopen:M_SYNC, cwrite）

○binary formatのファイルをつくることができる。

○I/Oノードは、1つのI/O-requestを受け取る。

○Paragonの外部に出力するftp（解析プログラム）用detail dataに適している。

○全ノード以外での入出力は不可能。

```
c     ibuffsize ;出力配列の大きさ
c     idatatype ;出力配列変数の型のサイズ
c     iid       ;論理機番
      parameter( ibuffsize=9000000, iid=10, idatatype=4  )
c     npfs      ;使用するpfsに含まれるI/Oノード数
c     iomemory  ;一つのI/Oノードのメモリ量
      parameter( npfs=8, iomemory = 64000000)
c     mymatrix  ;出力配列
      real*4 mymatrix(ibuffsize/idatatype)
      inculude 'fnx.h'

c     iam       ;実行ノードid
      iam = mynode()
c     num       ;実行ノード総数
      num = numnodes()
c     ibuffsize*num/npfs;一つのI/Oノード当たりの最終出力量
c     IDEV              ;mymatrixの入出力分割数
      IDEV     = ibuffsize*num/npfs/iomemory          .....①
      call gopen(iid, '/pfs8-6/mydata', M_SYNC)      .....②
      jjj = 1
      do  1000  ijk = 1,  IDEV
         jjj = ( ijk - 1 )*ibuffsize/idatatype/IDEV+1 .....③
         call  cwrite(iid, mymatrix(jjj) , ibuffsize/IDEV)   ....④
 1000 continue
c     mymatrixを分割したときの残差の処理
      if ( ibuffsize .ne. (jjj-1)*idatatype ) then     .....⑤
         ibuff =  ibuffsize-(jjj-1)*idatatype
         call cwrite(iid,mymatrix(jjj+1),ibuff)        .....⑥
      end if
      close(iid)                                      .....⑦
```

プログラム説明

①mymatrixの入出力分割数IDEV＝一つのI/Oノード当たりの最終出力量/I/Oノードメモリを求める。

②次に、論理機番iid=10でM_SYNCモードのgopenを使い全ノードで同一ファイル/pfs8-6/mydataを開く。

③分割しても連続的に出力できるようにmymatrixの出力ポインター位置を計算する。

④全ノードでcwriteで同期をとりながら各ノードごとにI/Oノードへデータを送信する。すなわち、I/O-requestを発行する。全ノード送信が終わった段階で、I/Oノードメモリ内のデータをディスクへ完全に書き込む。

③～④をIDEV-1回繰り返す。

⑤ibuffsize/idatatypeがIDEVで割り切れないとき、このif文の中を実行する。

⑥割り切れなかった分の残りのデータは、ibuff = ibuffsize-(jjj-1)*idatatype Byteとなる。④と同様の手続きでディスクへ完全に書き込む。

⑦論理機番iid=10のファイル/pfs8-6/mydataを閉じる。

注）(2)のように、全ノードcwriteを発行したときに入出力がI/Oノードのメモリ容量を超えないようにしなければならない。すなわち、do 1000のループが一回回ったとき、一つのI/Oノード当たりの総出力量=ibuffsize/IDEV*num/npfsが64MByteを超えないようにIDEVを調整しなければならない。上のプログラムでは、IDEV=900000*512/8/64000000=0.9->0となる。この場合、IDEV=0なので、⑥が実行され、分割なしで出力が完了することになる。

参考）cによるプログラム(gopenの引数がfortranと異なる。)

```
    #inculude <nx.h>
     int jjj,ijk;
/*   mymatrix  ;出力配列                          */
/*   ibuffsize ;出力配列の大きさ                  */
     int ibuffsize=900000;
/*   idatatype ;出力配列変数の型のサイズ          */
     int idatatype=4;
/*   npfs      ;使用するpfsに含まれるI/Oノード数*/
     int npfs=8;
/*   iomemory  ;一つのI/Oノードのメモリ量          */
     int iomemory = 64000000;
/*   iam       ;実行ノードid                      */
     int iam = mynode();
/*   num       ;実行ノード総数                    */
     int num = numnodes();
/*   ibuffsize*num/npfs;一つのI/Oノード当たりの最終出力量 */
/*   IDEV              ;mymatrixの入出力分割数          */
     int IDEV    = ibuffsize*num/npfs/iomemory+1;      .....①
     int iid = gopen("/pfs8-6/mydata"
             ,O_CREAT | O_TRUNC | ORDWD,M_SYNC,0644);.....②
```

```
      for( ijk=1 ; ijk <= IDEV ; ijk++ ) {
          jjj = (ijk-1)*ibuffsize/idatatype/IDEV+1;     .....③
           cwrite(iid,mymatrix[jjj],ibuffsize/IDEV);    .....④
      }
/*     mymatrixを分割したときの残差の処理                    */
          if(ibuffsize!==(jjj-1)*idatatype){            .....⑤
          ibuff =  ibuffsize-(jjj-1)*idatatype;
          cwrite(iid,mymatrix[jjj+1],ibuff);            .....⑥
      }
       close(iid);                                      .....⑦
```

Fig.3.3と同様に、計算ノード1, 2....., num-1で同一ファイルに追加書き込みする。

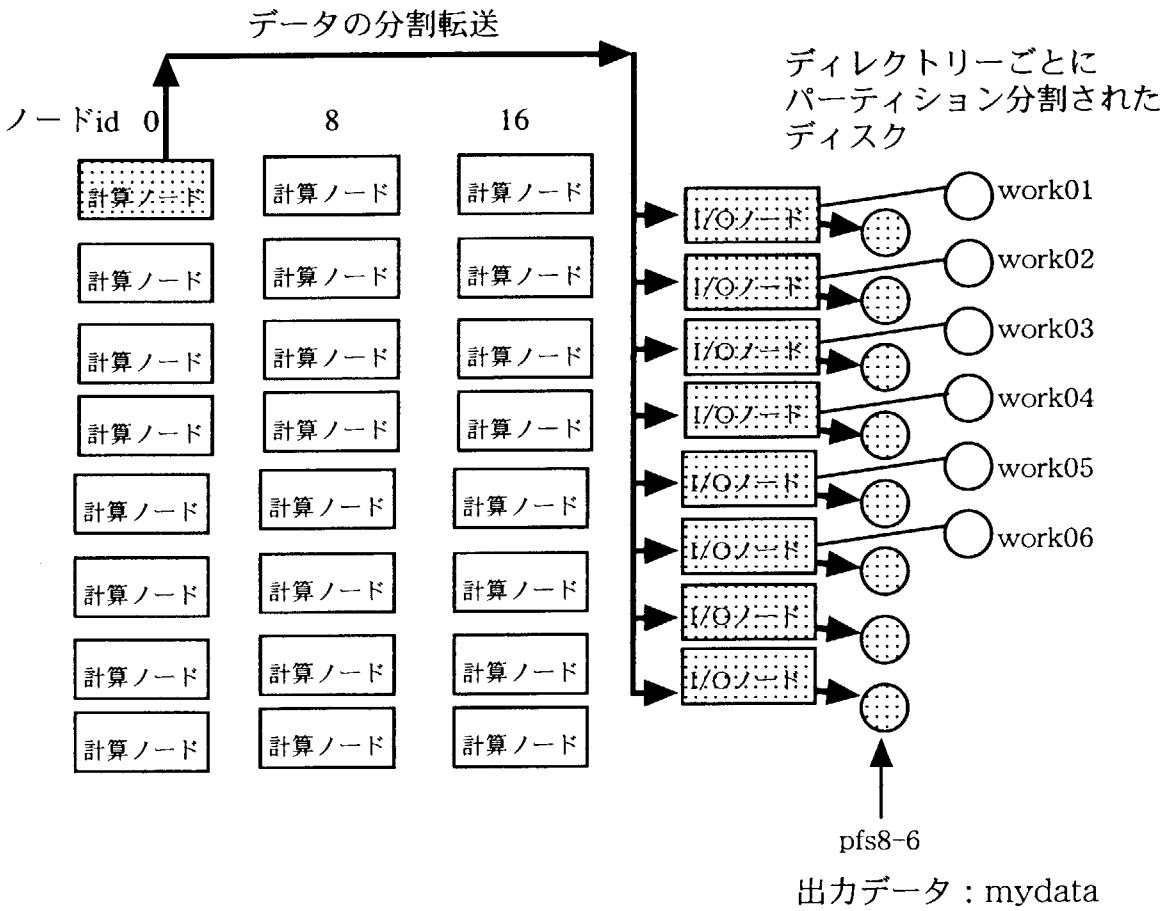

Fig.3.3 全ノードでの単一ufsファイル入出力

(4)全ノードでの単一pfsファイル入出力（gopen:M_RECORD, cwrite）

○binary formatのファイルをつくることができる。

○I/Oノードは、同時並列入出力ノード数と同数のI/O-requestを受け取る。

○Paragon内でしか利用しないrestart data等に適している。

○(2)より20倍ほど高速。

○全ノード以外での入出力は不可能。

```
c     ibuffsize ;出力配列の大きさ
c     idatatype ;出力配列変数の型のサイズ
c     iid       ;論理機番
      parameter( ibuffsize=7000000, iid=10, idatatype=4  )
c     npfs      ;使用するpfsに含まれるI/Oノード数
c     iomemory  ;一つのI/Oノードのメモリ量
c     nodecwrite   ;同時発行入出力ノード数
      parameter( npfs=8, iomemory = 64000000, nodecwrite=8 )
c     mymatrix  ;出力配列
      real*4 mymatrix(ibuffsize/idatatype)
      inculude 'fnx.h'

c     iam       ;実行ノードid
      iam = mynode();
c     num       ;実行ノード総数
      num = numnodes();

c     irw           ;同時発行cwriteの繰り返し数
      irw      = (num+nodecwrite-1)/nodecwrite              .....①
c     ibuffsize*num/npfs;一つのI/Oノード当たりの最終出力量
c     IDEV              ;mymatrixの入出力分割数
      IDEV     = ibuffsize*num/npfs/iomemory                ....②
      call gopen(iid,/pfs8-1/mydata', M_RECORD)             ....③
      jjj = 1
      do  1000  ijk = 1,  IDEV
      do  1100  ir = 1, irw
            if ( mod(iam,irw).eq.ir-1 )   then              ....④
               jjj = ( ijk - 1 )*ibuffsize/idatatype/IDEV+1 ....⑤
               call cwrite(iid, mymatrix(jjj),ibuffsize/IDEV)  ....⑥
            end if
            call gsync()                                    ....⑦
 1100 continue                                              ....⑧
 1000 continue
```

```
c     mymatrixを分割したときの残差の処理
      if ( ibuffsize .ne. (jjj-1)*idatatype ) then          .....⑨
      do  1200  ir = 1, irw
            if ( mod(iam,irw).eq.ir-1 )   then
               ibuff =  ibuffsize-(jjj-1)*idatatype
               call cwrite(iid, mymatrix(jjj+1),ibuff)          ....⑩
            end if
            call gsync()
 1200 continue
      end if
      close(iid)                                          .....⑪
```

プログラム説明

①同時発行cwriteの繰り返し数irwを求める。

②次に、mymatrixの入出力分割数IDEV＝一つのI/Oノード当たりの最終出力量/I/Oノードメモリを求める。

③次に、論理機番iid=10でM_RECORDモードのgopenを使い全ノードで同一ファイル/pfs8-1/mydataを開く。

④同時発行入出力ノード数 nodecwriteずつ、if文の中を通る。

⑤分割しても連続的に出力できるようにmymatrixの出力ポインター位置を計算する。

⑥if文を通った同時発行入出力ノードでcwriteで並列にI/Oノードへデータを送信する。すなわち、I/Oノード当たりI/O-requestを同時発行入出力ノード数だけ発行する。

⑦全ノードで同期をとり、I/OノードにI/O-requestが残っていないことを確認する。

④～⑦をirw-1回繰り返す。

⑧全ノード送信が終わった段階で、I/Oノードメモリ内のデータをディスクへ完全に書き込む。

④～⑧をirw*（IDEV-1）回繰り返す。

⑨ibuffsize/idatatypeがIDEVで割り切れないとき、このif文の中を実行する。

⑩割り切れなかった分の残りのデータは、ibuff＝ ibuffsize-(jjj-1)*idatatype Byteとなる。⑧と同様の手続きでディスクへ完全に書き込む。

⑪論理機番iid=10のファイル/pfs8-1/mydataを閉じる。

注）同時並列入出力ノード数は、I/O-request数と等価なので上限の32を超えないようにしなければならない。irwは同時並列入出力ノード数を制御するために一度でのcwrite発行数を決定する変数で、同時並列入出力ノード数を8以下とし、計算ノード数を512とすると、irw>計算ノード数/同時並列入出力ノード数=(512+8-1)/8=64.875->64として求めればよい。ここで、同時並列入出力ノード数とは、do1100の一回のループ、すなわちif (mod(iam,irw).eq.ir-1)～end ifまでにcwriteが発行されたノード数である。全ノードcwriteを

発行し終わった（=do1100のループが終わった）とき、一つのI/Oノード当たりの総出力量=ibuffsize/IDEV*num/I/Oノード数が64MByteを超えないようにしなければならない。IDEVは、全ノードcwriteが発行されたときに入出力がI/Oノードのメモリ容量を超えないように制御する変数で、IDEV =ibuffsize*num/ npfs /iomemoryとして求めればよい。上のプログラムでは、IDEV=70000000*512/8/64000000=70となる。この場合、ibuffsize = 70000000 / idatatype =4がIDEV=70で割り切れるので、*は実行されない。

参考）cによるプログラム(gopenの引数がfortranと異なる。)

```
#inculude <nx.h>
    int jjj,ijk;
/*    mymatrix  ;出力配列                              */
/*    ibuffsize ;出力配列の大きさ                      */
      int ibuffsize=7000000;
/*    idatatype ;出力配列変数の型のサイズ              */
      int idatatype=4;
/*    npfs      ;使用するpfsに含まれるI/Oノード数*/
      int npfs=8;
/*    iomemory  ;一つのI/Oノードのメモリ量              */
      int iomemory = 64000000;
/*    nodecwrite   ;同時発行入出力ノード数             */
      int nodecwrite=8;
/*    iam       ;実行ノードid                          */
      int iam = mynode();
/*    num       ;実行ノード総数                        */
      int num = numnodes()
/*    irw           ;同時発行cwriteの繰り返し数   */
      int irw       = (num+nodecwrite-1)/nodecwrite;                  ..........①
/*    ibuffsize*num/npfs;一つのI/Oノード当たりの最終出力量   */
/*    IDEV              ;mymatrixの入出力分割数              */
      int IDEV      = ibuffsize*num/npfs/iomemory;                    ..........②
      int iid=
      gopen("/pfs8-1/mydata", O_CREAT | O_TRUNC | O_RDWR,M_RECORD, 0644 );..③
      jjj = 1;
     for( ijk=1 ; ijk <= IDEV ; ijk++ ) {
          for( ir=1 ; ir <= irw ; i++ ) {
               if( (iam % irw) == (ir-1) ){                           ..........④
                    jjj = ( ijk - 1 )*ibuffsize/idatatype/IDEV+1;     ...⑤
                    cwrite(iid, mymatrix[jjj], ibuffsize/IDEV);       .....⑥
               }
               gsync();                                               ..........⑦
          }
     }                                                               ..........⑧
```

```
/*      mymatrixを分割したときの残差の処理                    */
     for( ir=1 ; ir <= irw ; i++ ) {
       if( ibuffsize !== (jjj-1)*idatatype ){                    .....⑨
         ibuff =  ibuffsize-(jjj-1)*idatatype;
         cwrite(iid,mymatrix[jjj+1],ibuff);                      .....⑩
       }
     }
      close( iid );                                             .....⑪
```

Fig.3.4 と同様に、計算ノード（8～15）,....., (num-8, num-1)で同一ファイルに追加書き込みする。

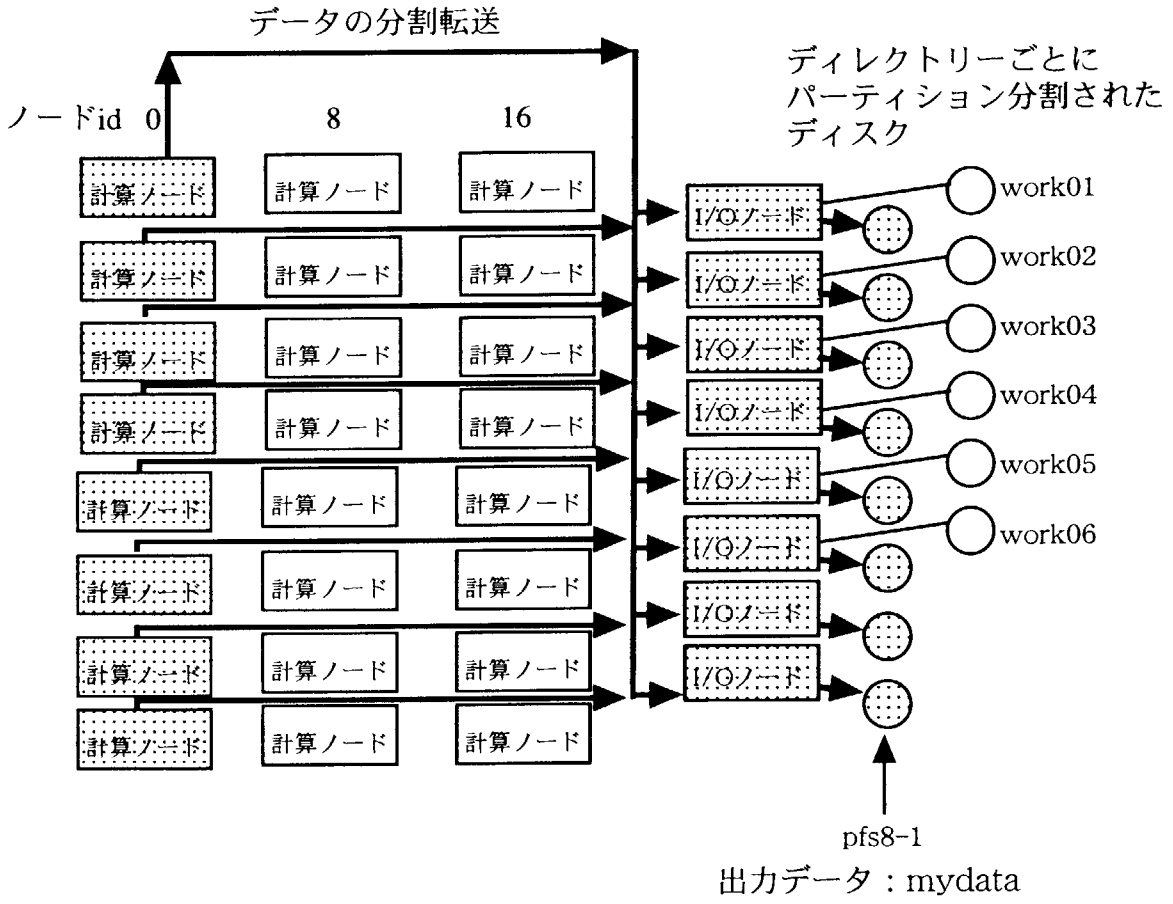

Fig.3.4 全ノードでの単一pfsファイル入出力

4.入出力に関わる制限事項

◎各ノードで、利用できる論理機番は61個である。

1）Paragonは、複数の計算ノードをもっているが、一つの計算ノードに着目すると同時に論理機番は61個までしか占有できない。1つの計算ノードから常時64種類以上の入出力ファイルを取り扱う場合、openを使わず、c言語の入出力関数（cではI/Oポインターを EOFにおくことは可能）を作りそれをfortranから呼び出すようにしなければならない。
注）実際には、64個の論理機番を扱えるのだが、標準出力等で3つはすでに予約されている。

◎I/Oノードが同時に受け取れるI/O-request数は、32以下である。

2）Paragonは、複数のI/Oノードをもっている。Paragonで実行されている全てのプロセスで1つのI/Oノードあたり同時に32個までしかI/O-requestを取り扱えない。ここで言うI/O-requestとは、前述の参考プログラムで説明したとおり、open文でノードごとに異なるファイルをcwrite(cread)で入出力する場合(2)には、逐次処理されるため、I/O-request数は1となる。gopen文を使った場合、M_SYNCモード(3)でのI/O-request数は、1であるが、M_RECORDモード(4)の場合、同時に並列入出力ノード数だけI/O-requestが発行される。それ以外にもUNIXコマンドのls, mv, ftpなどもI/O-requestを1つずつ発生させる。

◎I/Oノードのメモリは、64Mbyteである。（だが、その状態は直接的には知り得ない）

3）I/Oノードのメモリは、64MByteであり、並列入出力をした場合、一回（全ノードがI/Oをする）の入出力がI/Oノードのメモリ容量64MByteを超えないようにプログラムしなければならない。

5. 結び

Paragonの動作不確実の原因の一つは、ディスクI/Oが通信負荷を増加させることにある。各ノードがディスクを持ち、スワップ領域をローカルディスク上にとれば、通信負荷を増やさず、仮想記憶を活用することができてプログラムの柔軟性が増えるだろう。また、メッセージバッファ容量に関して、初期に指定した大きさをノード数で均等に分ける方式はシステムスケーラビリティを損なっており、より柔軟にメッセージバッファを活用できることが望まれる。さらに、FORTRANの通信及び入出力システム関数などの関数やサブルーティンがより適切なエラーを返すことが、プログラムの信頼性を高めるために望まれる。

謝　辞

本研究は、日本原子力研究所関西研究所光量子科学センターで行われたものであり、すばらしい研究の場を提供していただくとともに、研究に対する深い御理解と御支援をいただいた大野英雄所長に謝意を表します。協力体制を整備し研究を推進してくだだきました有澤孝センター長に深く感謝いたします。また、光量子シミュレーション研究グループにおいて、この研究を強く御支援していただきました田島俊樹グループリーダー、井原均サブグループリーダー、那珂研究所炉心プラズマ研究部プラズマ理論研究室長岸本泰明氏に深厚なる謝意を表します。最後に、関西研駐在の情報システム管理課及び富士通、Intel各社のシステムエンジニアの皆様には、いつも計算機運用支援及び情報提供などでお世話になりました。改めて、ここで厚くお礼申し上げます。

参考文献

1)関西研究所　並列計算機　(Paragon)　利用手引き
日本原子力研究所計算科学技術推進センター
2)PARAGON TM Fortran Compiler User's Guide
3)PARAGON TM Fortran System Calls Reference Manual
4)佐々木明 "MPIを用いた２次元流体シミュレーション", JAERI-Data/Code 97-048
5)上島豊、荒川拓也、佐々木明、横田恒 "Paragon上での超並列プログラム開発ガイド", JAERI-Data/Code 98-030
6)K. Tani, "*Advanced Phton Simulation*",
Journal of Plasma and Fusion Reserch 72, 935 (1996)
(光量子シミュレーション、プラズマ・核融合学会誌)

This is a blank page.

国際単位系(SI)と換算表

表1　SI基本単位および補助単位

量	名称	記号
長さ	メートル	m
質量	キログラム	kg
時間	秒	s
電流	アンペア	A
熱力学温度	ケルビン	K
物質量	モル	mol
光度	カンデラ	cd
平面角	ラジアン	rad
立体角	ステラジアン	sr

表2　SIと併用される単位

名称	記号
分，時，日	min，h，d
度，分，秒	°，′，″
リットル	l，L
トン	t
電子ボルト	eV
原子質量単位	u

1 eV＝1.60218×10^{-19} J

1 u＝1.66054×10^{-27} kg

表5　SI接頭語

倍数	接頭語	記号
10^{18}	エクサ	E
10^{15}	ペタ	P
10^{12}	テラ	T
10^{9}	ギガ	G
10^{6}	メガ	M
10^{3}	キロ	k
10^{2}	ヘクト	h
10^{1}	デカ	da
10^{-1}	デシ	d
10^{-2}	センチ	c
10^{-3}	ミリ	m
10^{-6}	マイクロ	μ
10^{-9}	ナノ	n
10^{-12}	ピコ	p
10^{-15}	フェムト	f
10^{-18}	アト	a

表3　固有の名称をもつSI組立単位

量	名称	記号	他のSI単位による表現
周波数	ヘルツ	Hz	s^{-1}
力	ニュートン	N	$m\cdot kg/s^2$
圧力，応力	パスカル	Pa	N/m^2
エネルギー，仕事，熱量	ジュール	J	N·m
工率，放射束	ワット	W	J/s
電気量，電荷	クーロン	C	A·s
電位，電圧，起電力	ボルト	V	W/A
静電容量	ファラド	F	C/V
電気抵抗	オーム	Ω	V/A
コンダクタンス	ジーメンス	S	A/V
磁束	ウェーバ	Wb	V·s
磁束密度	テスラ	T	Wb/m^2
インダクタンス	ヘンリー	H	Wb/A
セルシウス温度	セルシウス度	℃	
光束	ルーメン	lm	cd·sr
照度	ルクス	lx	lm/m^2
放射能	ベクレル	Bq	s^{-1}
吸収線量	グレイ	Gy	J/kg
線量当量	シーベルト	Sv	J/kg

表4　SIと共に暫定的に維持される単位

名称	記号
オングストローム	Å
バーン	b
バール	bar
ガル	Gal
キュリー	Ci
レントゲン	R
ラド	rad
レム	rem

1 Å＝0.1 nm＝10^{-10} m

1 b＝100 fm^2＝10^{-28} m^2

1 bar＝0.1 MPa＝10^5 Pa

1 Gal＝1 cm/s^2＝10^{-2} m/s^2

1 Ci＝3.7×10^{10} Bq

1 R＝2.58×10^{-4} C/kg

1 rad＝1 cGy＝10^{-2} Gy

1 rem＝1 cSv＝10^{-2} Sv

（注）

1. 表1－5は「国際単位系」第5版，国際度量衡局 1985年刊行による。ただし，1 eVおよび1 uの値はCODATAの1986年推奨値によった。
2. 表4には海里，ノット，アール，ヘクタールも含まれているが日常の単位なのでここでは省略した。
3. barは，JISでは流体の圧力を表わす場合に限り表2のカテゴリーに分類されている。
4. EC閣僚理事会指令ではbar，barnおよび「血圧の単位」mmHgを表2のカテゴリーに入れている。

換算表

力	N(=10^5 dyn)	kgf	lbf
	1	0.101972	0.224809
	9.80665	1	2.20462
	4.44822	0.453592	1

粘度　1 Pa·s(N·s/m^2)＝10 P(ポアズ)(g/(cm·s))

動粘度　1 m^2/s＝10^4 St(ストークス)(cm^2/s)

圧力	MPa(=10 bar)	kgf/cm^2	atm	mmHg(Torr)	lbf/in^2(psi)
	1	10.1972	9.86923	7.50062×10^3	145.038
	0.0980665	1	0.967841	735.559	14.2233
	0.101325	1.03323	1	760	14.6959
	1.33322×10^{-4}	1.35951×10^{-3}	1.31579×10^{-3}	1	1.93368×10^{-2}
	6.89476×10^{-3}	7.03070×10^{-2}	6.80460×10^{-2}	51.7149	1

エネルギー・仕事・熱量	J(=10^7 erg)	kgf·m	kW·h	cal(計量法)	Btu	ft·lbf	eV
	1	0.101972	2.77778×10^{-7}	0.238889	9.47813×10^{-4}	0.737562	6.24150×10^{18}
	9.80665	1	2.72407×10^{-6}	2.34270	9.29487×10^{-3}	7.23301	6.12082×10^{19}
	3.6×10^6	3.67098×10^5	1	8.59999×10^5	3412.13	2.65522×10^6	2.24694×10^{25}
	4.18605	0.426858	1.16279×10^{-6}	1	3.96759×10^{-3}	3.08747	2.61272×10^{19}
	1055.06	107.586	2.93072×10^{-4}	252.042	1	778.172	6.58515×10^{21}
	1.35582	0.138255	3.76616×10^{-7}	0.323890	1.28506×10^{-3}	1	8.46233×10^{18}
	1.60218×10^{-19}	1.63377×10^{-20}	4.45050×10^{-26}	3.82743×10^{-20}	1.51857×10^{-22}	1.18171×10^{-19}	1

1 cal＝4.18605 J(計量法)

＝4.184 J　(熱化学)

＝4.1855 J　(15 °C)

＝4.1868 J(国際蒸気表)

仕事率　1 PS(仏馬力)

＝75 kgf·m/s

＝735.499 W

放射能	Bq	Ci
	1	2.70270×10^{-11}
	3.7×10^{10}	1

吸収線量	Gy	rad
	1	100
	0.01	1

照射線量	C/kg	R
	1	3876
	2.58×10^{-4}	1

線量当量	Sv	rem
	1	100
	0.01	1

(86年12月26日現在)

Paragon上での入出力プログラミングガイド