JAERI-Data/Code
2000-038

原子力コードの高速化（スカラ並列化編）

－平成11年度作業報告書－

2000年12月

箭竹　陽一*・久米　悦雄・川井　渉*・根本　俊行*
川崎　信夫*・足立　将晶・石附　茂*・小笠原　忍

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 2000－038

原子力コードの高速化 (スカラ並列化編)
－ 平成 11 年度作業報告書 －

日本原子力研究所計算科学技術推進センター
箭竹 陽一 ** ・久米 悦雄・川井 渉 * ・根本 俊行 *
川崎 信夫 * ・足立 将晶 ※ ・石附 茂 * ・小笠原 忍 ※

(2000 年 10 月 2 日受理)

本報告書は，平成 11 年度に計算科学技術推進センター情報システム管理課で行った原子力コードの高速化作業のうち，Paragon におけるスカラ並列化作業部分について記述したものである．原子力コードの高速化作業は，平成 11 年度に 18 件行われた．これらの作業内容は，今後同種の作業を行う上での参考となりうるよう，作業を大別して「ベクトル／並列化編」，「スカラ並列化編」及び「移植編」の 3 分冊にまとめた．

本報告書の「スカラ並列化編」では，高エネルギー核子・中間子輸送計算コード NMTC，プラソフプラズマシミュレーションコード DA-VLASOV 及び中性子・光子結合モンテカルロ輸送計算コード MCNP4B2 を対象に実施した Paragon 向けのスカラ並列化作業について記述している．

日本原子力研究所（東海駐在）：〒 319-1195 茨城県那珂郡東海村白方白根 2-4
※ 外来研究員：富士通株式会社
** 株式会社日立製作所
＊ 富士通株式会社

JAERI－Data/Code 2000－038

Vectorization, Parallelization and Porting of Nuclear Codes (Parallelization on Scalar Processors) - Progress Report Fiscal 1999 -

Yo-ichi YATAKE**, Etsuo KUME, Wataru KAWAI*, Toshiyuki NEMOTO*, Nobuo KAWASAKI*, Masaaki ADACHI※, Shigeru ISHIZUKI* and Shinobu OGASAWARA※

Center for Promotion of Computational Science and Engineering
(Tokai Site)
Japan Atomic Energy Research Institute
Tokai-mura, Naka-gun, Ibaraki-ken

(Received October 2, 2000)

Several computer codes in the nuclear field have been vectorized, parallelized and transported on the FUJITSU VPP500 system, the AP3000 system, the SX-4 system and the Paragon system at Center for Promotion of Computational Science and Engineering in Japan Atomic Energy Research Institute. We dealt with 18 codes in fiscal 1999. These results are reported in 3 parts, i.e., the vectorization and the parallelization part on vector processors, the parallelization part on scalar processors and the porting part. In this report, we describe the parallelization on scalar processors.

In this parallelization on scalar processors part, the parallelization of Nucleon Meson Transport Code (NMTC), Differential Algebraic VLASOV code (DA-VLASOV) and Monte Carlo N-Particle transport code (MCNP4B2) on the Paragon system are described.

Keywords : NMTC, DA-VLASOV, MCNP4B2, Parallelization, Paragon, Nuclear Codes

※ On leave from FUJITSU, Ltd
** HITACHI,Ltd
* FUJITSU,Ltd

目　　次

1. はじめに 1
2. NMTC コードの並列化 3
　2.1 コード概要 3
　2.2 Paragon への移植 3
　2.3 並列化 3
　2.4 並列化の効果 5
　2.5 まとめ 5
3. DA-VLASOV コードの並列化 15
　3.1 コード概要 15
　3.2 Paragon への移植 15
　3.3 並列化 15
　3.4 並列化の効果 18
　3.5 まとめ 19
4. MCNP4B2 コードの並列化 36
　4.1 概要 36
　4.2 並列化 36
　4.3 並列化の効果 38
　4.4 まとめ 38
5. おわりに 57
謝辞 57

Contents

1. Introduction 1
2. Parallelization of NMTC 3
2.1 Overview of NMTC 3
2.2 Porting to Paragon System 3
2.3 Parallelization 3
2.4 Effects of Parallelization 5
2.5 Summary 5
3. Parallelization of DA-VLASOV 15
3.1 Overview of DA-VLASOV 15
3.2 Porting to Paragon System 15
3.3 Parallelization 15
3.4 Effects of Parallelization 18
3.5 Summary 19
4. Parallelization of MCNP4B2 36
4.1 Overview 36
4.2 Parallelization 36
4.3 Effects of Parallelization 38
4.4 Summary 38
5. Concluding Remarks 57
Acknowledgements 57

1. はじめに

計算科学技術推進センター情報システム管理課では，原研が保有する各種スーパーコンピュータの効率的な運用とコンピュータ資源の有効利用を促進するため，計算需要の多い原子力コードをユーザに代わってスーパーコンピュータ上に整備し，それぞれのコードに最適な高速化を施す作業を実施している．この作業は，コンピュータの効率的利用を推進するのみならず，ユーザの計算待ち時間の短縮を通じてユーザの仕事の効率化へも貢献するものと思われる．

原子力コードの高速化作業は，平成11年度に18件行われた．これらの作業内容は，今後同種の作業を行う上での参考となりうるよう，作業を大別して，「ベクトル／並列化編」，「スカラ並列化編」及び「移植編」の3分冊にまとめた．

本報告書の「スカラ並列化編」では，高エネルギー核子・中間子輸送計算コードNMTC，ブラソフプラズマシミュレーションコードDA-VLASOV及び中性子・光子結合モンテカルロ輸送計算コードMCNP4B2を対象に実施したParagon向けのスカラ並列化作業について記述している．

別冊の「ベクトル／並列化編」では，JAMコード及び3次元熱流体解析コードSTREAMのベクトル化作業について，相対論的分子軌道法コードRSCAT，相対論的密度汎関数法コードRDFT及び高速3次元中性子拡散ノード法コードMOSRA-Lightのベクトル並列化作業について記述している．また，別冊の「移植編」では，生体分子の分子動力学パッケージAMBER5，（連続・多群）汎用中性子・光子輸送計算モンテカルロコードMVP/GMVP，MCNPライブラリ自動編集システムautonj，SPECTOR/SPECOMPコード，核融合炉事故解析コードMELCOR-FUS及びサブチャンネル解析コードCOBRA-TFのVPP500またはAP3000への整備について記述している．なお，平成11年度に実施した高速化作業のうち，ここで取り上げなかったいくつかのコードに関しては，ユーザとの連名により別途JAERI-Data/Codeを執筆する予定であるので，そちらを参照されたい．

2章では，高エネルギー核子・中間子輸送計算コードNMTCのParagonにおける並列化作業について述べる．本作業ではMPIを用いて関西研Paragon向き並列化を行った．計算コストが高いモンテカルロ計算部を並列に実行し，各物理量をノード間で足し合わせて統計をとるようにした．この結果，最大256ノードの並列実行で146倍の速度向上が得られた．

3章では，ブラソフプラズマシミュレーションコードDA-VLASOVのParagonにおける並列化作業について述べる．本作業では，まず，当該コードをVPP500からParagon上へ移植し，正常動作するように修正を施した後，MPIを用いてParagon向き並列化作業を行った．並列化作業では，差分式の並列化には「ネイバリング通信」，ルーチンADVINDXでは，各計算ノードが担当する領域の結果をすべてノード0に集めて全体の値を管理し，その値を各計算ノードに送信する方法を用いた．しかし，ADVINDXでの計算量及び通信量の増大により，計算コストの集中する本ルーチンでの並列化効率が低く，最大32ノードで速度向上率は約8.3倍となった．

4章では，中性子・光子結合モンテカルロ輸送計算コードMCNP4B2のParagonにおける並列化作業について述べる．本作業では，NX版である当該コードをMPI版へ変換した．MPI及び通信バッファの設定にはC言語を使用し，ノード0からの他ノードへのデータ分配通信には，

対角送信法を用いた．この結果，最大64ノードの並列実行で約32.5倍の速度向上が得られた．

なお，本報告書の作業は箭竹が担当した．

2. NMTCコードの並列化

2.1 コード概要

本作業の目的は，高エネルギー核子・中間子輸送計算コードＮＭＴＣ (Nucleon Meson Transport Code) のスカラー並列による並列化である．ＮＭＴＣは，高エネルギー核子・中間子輸送コードであり，陽子，中性子，π中間子によって引き起こされる核反応と，これらの粒子の非均質媒質中での輸送過程計算を行う．本コードは独立に開発された幾つかの計算コードを，その基本構成ルーチンとして組み込み，更に，それに関連した数多くのサブルーチンを包含した構造となっている．核反応計算部として，核内カスケード計算部，蒸発計算部，核分裂反応計算部に大別される．これに幾何形状定義部などの入力部，物質中の輸送計算部，計算結果出力部が接続したコード構成となっている [1][2]．尚，通信ライブラリにはＭＰＩを使用している．

2.2 Paragonへの移植

DEC-Alphaで動作しているＮＭＴＣコードのソースファイルをベースにし，Paragon 1ノードへの移植作業を実施した．移植作業時に修正したルーチン名と修正内容をTable 2.1，Table 2.2に示す．DEC-AlphaとParagonとの計算結果に有意な差はなく，Paragon 1ノードへの移植作業の妥当性を確認した．

2.3 並列化

2.3.1 並列化方針

並列化整備の方針を以下に示す．

(1) データ入力部は，代表ノード (ノード０) でデータを入力し，データを各ノードに転送する方法と，各ノードの読み込みを非同期とする方法を組み合わせる．

(2) 各計算ノードで使用する乱数系列を，同一のものとしない．

(3) 計算コストが高いモンテカルロ計算部（ovly12）を並列に実行し，その後，各物理量をノード間で足し合わせて，統計を行う事とする．

(4) データ出力部は，代表ノード (ノード０) から出力する方法と，各ノードから直接出力する方法で構成する．

2.3.2 並列化設計

本節では，並列化設計について説明する．

2.3.2.1 データ入力部の並列化

上記並列化方針に従って，各ノードが，同じ初期データを所有する様に修正した．修正したルーチン名とファイル名を Table 2.3に示す．各ノードが，同じ初期データを所有しているか否か確認する為，並列化した入力部の親ルーチン (ovly11) を多ノードで実行した後，並列化していない計算部 (ovly12) を各ノードで実行した．各々のノードでの計算結果は一致し，データ入力部の並列化処理が正常に実施されていることを確認した．

2.3.2.2 乱数設定方法

初期乱数を発生させるルーチン randmc 内で，各ノードが同じ乱数系列を使用しないように，変数 mynode【計算ノード番号: 0～（計算ノード総数－1）】を利用して，初期乱数を設定した． Fig. 2.1にルーチン randmc のソースリストを示す． Fig. 2.1内のライン番号 1 で，乱数 rijk を各ノード毎に異なる数値としている．

2.3.2.3 モンテカルロ計算部の並列化

モンテカルロ計算部の親ルーチンである ovly12 のソースリストを， Fig. 2.2に示す．ルーチン ovly12 での仮想粒子の入射に関するイベントループ (文番号 25 でのループ) を並列に実行し，その後，各物理量をノード間で足し合わせて，統計を行う事とする．各ノードが担当するイベント数【maxcas_me】を算出するルーチン prange （Fig. 2.3参照）を，イベントループを実行する前にコールして， maxcas (入射のイベント数) から， maxcas_me を算出する． prange では， irank (計算担当ノード番号)， nprocs （計算ノード総数）より maxcas_me を算出する．その後，各ノード毎に maxcas_me の数だけイベントの計算を行い，イベントループを抜けた後，neutno(1 バッチあたりの neutron の数) などの物理量をノード間で足し合わせて，全体での値を求める．このイベントループ内の計算を，必要とするバッチ数分繰り返す．

次に， Fig. 2.2中の (1) から (3) までのラインの説明を以下に述べる．

(1) prange をコールして，各ノードが担当するイベント数 maxcas_me を求める．

(2) 並列計算するイベントループの中で，総和をとる変数（しかもイベントのループ終了後ゼロクリアしている変数）であるため，次のバッチ計算に入る前に，ノード間で全体の数値を算出する．同様な処理を行った変数を Table 2.4に示す．

(3) (1) で求めた maxcas_me よりイベント数が少なければ文番号 25 に飛ぶ．

並列化計算用に新規に設定したコモン文【mpi.para】の内容を， Table 2.5に示す．

2.3.2.4 計算結果出力部の並列化

代表ノード (ノード 0) から計算結果を出力する為に，各ノード間の総和処理を行なう変数名と修正したルーチン名を Table 2.6に示す．また，出力装置番号と出力処理内容を Table 2.7に示す．モンテカルロ計算部と計算結果出力部の妥当性を確認する為，各ノードの初期乱数を固定して，検証計算を実施した．検証計算より，各ノードにおける計算結果が設計仕様通りに出力され，モンテカルロ計算部と計算結果出力部の妥当性を確認した．

2.4 並列化の効果

Paragon用に並列化したＮＭＴＣコードを用いて，関西研Paragonで１ノードから５１２ノードまでの経過時間を測定した．測定結果をTable 2.8に示す． Table 2.8より速度向上率の最大値は，２５６ノードで１４６倍となった．

2.5 まとめ

本作業では，高エネルギー核子・中間子輸送コードＮＭＴＣに対して，ＭＰＩによるインテル製スカラー並列計算機Paragon向き並列化整備を実施した．ＮＭＴＣコードのParagonでの速度向上率は，２５６ノードで１４６倍となった．更に，計算ノードを５１２ノードにすると，計算時間の減少割合よりデータ通信時間の増加割合の方が大きくなり，速度向上率は９７倍と減少した．

Table 2.1 Modification of original code (compile,link).

ファイル名	ルーチン名	修正内容
dbpnt.f	dbpnt	FORMAT文修正（文番号:6, 32）
ioinit.f	ioinit	automatic指定文をコメント
		ルーチン名変更（ioinit→ioinitp） *
main.f	main	ルーチン名変更（ioinit→ioinitp） *

* ” ioinit”がParagon System Lib.のファイル名と同名の為．

Table 2.2 Modification of original code (execution).

ファイル名	ルーチン名	修正内容
main.f	main	システムルーチンidateの引数を変更
ovly12.f	ovly12	システムルーチンidateの引数を変更

Table 2.3 Modified routine and files.

ルーチン名	ファイル名
main	main.f
ovly11	ovly11.f
jomin1	marslib.f
jomin2	marslib.f
gtvlin	marslib.f
azip	marslib.f
sazar	marslib.f
sors	sors.f
engbin	engbin.f
nmtin2	nmtin2.f
nmtin3	nmtin3.f
y0read	y0read.f
readnm	readnm.f
gthsig	gthsig.f
geni	geni.f

Table 2.4 Variable which are handled for summation between nodes (Monte Carlo calculation part).

変数名	備　考
neutno	
nocas	
npipt	
npizt	
npint	
nfiss	
ncall	ncall は，バッチを通じての累積変数なので，ダミー変数 ncalldu を使用して，累積計算.
an2(nobch)	
apip(nobch)	
api0(nobch)	
apin(nobch)	
negeng(1～7)	warning 判定用変数
ncasfl(1～3)	warning 判定用変数
nazero	warning 判定用変数
nzzero	warning 判定用変数
nwsfis	warning 判定用変数
nwsfis	warning 判定用変数

Table 2.5 Include file 'mpi.para'.

変数名	データ型	内　容
mynode	I*4	計算ノード番号
nodes	I*4	計算ノード総数
nodew	I*4	テスト用変数（通常時 =0）
ndim	I*4	ワーク配列 ddim の大きさ
ddim(ndim)	R*8	ワーク配列（通信関数で使用）

Table 2.6 Variable which are handled for summation between nodes(output part).

ルーチン名	変数名	出力装置番号
analy3	revts pevts rhevts phevys pdevts wtxreg peiv	6
regnpr	talreg	32
surfpr	talspc talsfl talscr	36～38
heatpr	talhet	51～56 61，62
bsrfpr	talbc talbct talbf talbft	71，72
yildpr	talyld	40
analyz	jtneut wtneut	6

Table 2.7 Unit specifier and output processing.

出力装置番号	出力処理内容
6	データ集約して，ノード0より出力
12	各ノードより出力 file012m/file012.###
13	各ノードより出力 file013m/file013.###
15	各ノードより出力 file015m/file015.###
32	データ集約して，ノード0より出力
36～38	データ集約して，ノード0より出力
40	データ集約して，ノード0より出力
41	各ノードより出力 file041m/file041.###
51～56	データ集約して，ノード0より出力
61，62	データ集約して，ノード0より出力
71，72	データ集約して，ノード0より出力

Table 2.8 Speed up ratio.

計算ノード数	経過時間（min）	速度向上率
1	585	1
2	296	1.98
4	148	3.95
8	75	7.80
16	38	15.39
32	19	30.79
64	10	58.50
128	5	117.00
256	4	146.25
512	6	97.50

計算データ　：ユーザー提示テストデータ【sample2.in】
ヒストリ数　：1000 * 300　(イベント数＊バッチ数)

```
                    subroutine randmc
c -------------------------------------------------------------------
c     controls for the pseudo-random number sequence.
c     ==mcnp4a:random==
c -------------------------------------------------------------------
      implicit double precision (a-h,o-z)
CYS
      include '../include2/mpi.para'
CYE
      parameter (fb=13008944d0,fs=170125d0,gb=1136868d0,gs=6328637d0,
     1 p=2d0**24,q=2d0**(-24),rm=5d0**19)
c
      common /randm4/ rnfb,rnfs,rngb,rngs,rnmult,rijk,ranj,rani,rans,
     &                ranb,rnrtc,
     &                nstrid,inif
      common /iradkk/ randkk
      dimension dbcn(14)
      data dbcn/14*0.d0/
     &     iuo/6/
c -------------------------------------------------------------------
c
      nstrid=152917
      rnfb=fb
      rnfs=fs
      rngb=gb
      rngs=gs
      rnmult=rm
  ---------------- (途中省略) ----------------
   30 rijk=rnmult
      if(dbcn(1).gt.0.)rijk=dbcn(1)
CYS
      rijk = rijk + 10000*mynode                        ------ (1)
      write(6,*) 'PE=',mynode,'first randum number =',rijk
CYE
      if(dbcn(1)+dbcn(8).ne.0.)inif=1
      rani=aint(rijk*q)
      ranj=rijk-rani*p
      do 40 i=1,int(dbcn(8))
   40 call advijk
      if(dbcn(1)+dbcn(8).ne.0.)write(iuo,50)rijk
   50 format(25h starting random number =,2x,f16.0,t11,1h )
      dbcn(1)=0.
      dbcn(8)=0.
      return
      end
```

Fig. 2.1 Modification of subroutine randmc.

```
      subroutine ovly12
c -------------------------------------------------------------------
      implicit real*8 (a-h,o-z)
      include '../include2/param'
      include '../include2/comon'
CYS
      include '../include2/mpi.para'
CYE
c
  ---------------- (途中省略) ------------------
CYS
      call prange(1,maxcas,nodes,mynode,ista,iend,maxcas_me) -- (1)
CYE
   25 nocas=nocas+1
  ---------------- (途中省略) ------------------
      if (no.le.nomax) go to 40
CYS
C      if (nocas.lt.maxcas) go to 25
      if (nocas.lt.maxcas_me) go to 25      ---------- (3)
CYE
   65 nobch=nobch+1
CYS ---------------------------------------------------------------
C      call mpi_allreduce(nocas,idummy,1,mpi_integer,mpi_sum,
C     1                                    mpi_comm_world,ierr)
      call mpi_reduce(nocas,idummy,1,mpi_integer,mpi_sum,
     1                                   0,mpi_comm_world,ierr)
      nocas = idummy                                          (2)
C      call mpi_allreduce(neutno,idummy,1,mpi_integer,mpi_sum,
C     1                                     mpi_comm_world,ierr)
      call mpi_reduce(neutno,idummy,1,mpi_integer,mpi_sum,
     1                                   0,mpi_comm_world,ierr)
      neutno = idummy
CYE-----------------------------------------------------------------
       ncas=ncas+nocas
CYS
```

Fig. 2.2 Modification of subroutine ovly12(1/3).

```
      ncalldu = ncall
C      call mpi_allreduce(ncall,idummy,1,mpi_integer,mpi_sum,
C     1                                    mpi_comm_world,ierr)
      call mpi_reduce(ncall,idummy,1,mpi_integer,mpi_sum,
     1                                   0,mpi_comm_world,ierr)
      ncall = idummy
C      call mpi_allreduce(an2(nobch),ddummy,1,mpi_real8,mpi_sum,
C     1                                    mpi_comm_world,ierr)
      call mpi_reduce(an2(nobch),ddummy,1,mpi_real8,mpi_sum,
     1                                   0,mpi_comm_world,ierr)
      an2(nobch) = ddummy
CYE
   70 if (ncall.ge.1000000000) then
CYS
C         ncallk=ncallk+1
C         ncall=ncall-1000000000
          ncallk = ncall/1000000000
          ncall = ncall-ncallk*1000000000
CYE
      endif
      hhcall=' '
      if (ncallk.gt.0) then
         write(hhcall,'(i4,i9.9)') ncallk,ncall
      else
         write(hhcall(5:13),'(i9)') ncall
      endif
CYS
C      write(io,75) nobch,neutno,ncas,rijk,hhcall
      if (mynode .eq. 0) then
        write(io,75) nobch,neutno,ncas,rijk,hhcall
      end if
CYE
```

Fig. 2.2 Modification of subroutine ovly12(2/3).

```
   75 format(' batch ',i3,'  cutoff-neutrons=',i8,
     &       ', cumulative cascades=',i8,', random=',f16.0,', calls=',a)
      if (iwt.eq.1) write(io,74) an2(nobch)
   74 format('            cutoff-n weight=',1pe14.7)
      call timex
CYS
      ncall = ncalldu
C
C      call mpi_allreduce(negeng,idumdim,7,mpi_integer,mpi_sum,
C     1                                    mpi_comm_world,ierr)
      call mpi_reduce(negeng,idumdim,7,mpi_integer,mpi_sum,
     1                                    0,mpi_comm_world,ierr)
      do i = 1,7
       negeng(i) = idumdim(i)
      end do
c
C      call mpi_allreduce(ncasfl,idumdim,3,mpi_integer,mpi_sum,
C     1                                    mpi_comm_world,ierr)
      call mpi_reduce(ncasfl,idumdim,3,mpi_integer,mpi_sum,
     1                                    0,mpi_comm_world,ierr)
      do i =1,3
        ncasfl(i) = idumdim(i)
      end do
c
C      call mpi_allreduce(nazero,idummy,1,mpi_integer,mpi_sum,
C     1                                    mpi_comm_world,ierr)
      call mpi_reduce(nazero,idummy,1,mpi_integer,mpi_sum,
     1                                    0,mpi_comm_world,ierr)
      nazero = idummy
  ---------------- (途中省略) ----------------
      end
```

Fig. 2.2 Modification of subroutine ovly12(3/3).

```
      subroutine prange(n1,n2,nprocs,irank,ista,iend,iii)
c
      ii = n2 - n1 + 1
      if (nprocs .gt. ii) then
        if(irank .eq. 0) then
          write(6,*) '********** prange  error*********'
          write(6,*) 'nodes n1 n2=',nprocs,n1,n2
        end if
        stop 800
      end if
      iwork1 = (n2-n1+1)/nprocs
      iwork2 = mod(n2-n1+1,nprocs)
      ista = irank*iwork1 + n1 + min(irank,iwork2)
      iend = ista + iwork1 - 1
      if (iwork2 .gt. irank ) iend = iend + 1
      iii = iend-ista+1
c
      if (irank .eq. 0) then
C        write(6,*) '********** prange *********'
      end if
C      write(6,*) 'myrank ista iend iii=',irank,ista,iend,iii
c
      return
      end
```

Fig. 2.3 Subroutine prange.

参 考 文 献

[1] 中原　康明・筒井　恒夫：高エネルギー核反応および核子・中間子輸送シミュレーション・コードシステム NMTC/JAERI, JAERI-M 82-198, 1982年12月.

[2] 高田　弘・義澤　宣明・小迫　和明・石橋　健二：核子・中間子輸送コードの改良版 NMTC/JAERI97, JAERI-Data/Code 98-005, 1998年2月.

3. DA-VLASOV コードの並列化

3.1 コード概要

本作業の目的は，プラズマシミュレーションコードDA-VLASOV (Differential Algebraic Vlasov code) のスカラー並列による高速化である．本章では，インテル製スカラー並列計算機Paragonに対する，上記DA-VLASOVコードの並列化作業(MPI版)について述べる．DA-VLASOVコードは，1次元Vlasov-Poison方程式を位相空間で解いて計算を行っている．数値解法にはDA-CIP (Differential Algebraic Cubic Interpolated Propagation) 法を用いている．DA-VLASOVコードの並列化は，数値解法に微分代数的ＣＩＰ法 (DA-CIP法) [1] 及び差分形式で計算を実行している部分について，隣接ノードからの通信データを使用するネイバリング通信を使用した．また，ルーチンADVINDXでは，各計算ノードが計算を担当する計算領域での計算結果を0ノードに集めて全体での値を作成し，その値を各計算ノードに送信する方法を実施した．

3.2 Paragonへの移植

並列化作業の前作業として，DA-VLASOVコード(VPP500ベース)のParagon (那珂研) 1ノードへの移植作業を実施した．DA-VLASOVコードを下記の条件にて，Paragon 1ノードへ移植した．

(a) 移植先計算機　：Paragon S2
(b) 最適化　　：-O0 (コンパイル時最適化なし)
(c) 計算体系　　：-127 ～ 128

移植時に発生したwarning及びerrorメッセージの内容をFig. 3.1に示す．これらのメッセージを出力した箇所をFig. 3.2に示す様に変更し，Paragon上での動作確認を実施した．その時使用したテスト入力データの内容をFig. 3.3に示す．VPP500とParagonの計算結果は一致し，DA-VLASOVコードのParagon 1ノードへの移植作業の妥当性を確認した．

3.3 並列化

3.3.1 並列化方針

Fig. 3.4にDA-VLASOVコードのソースツリー図を示す(図中の数値は省略した子ルーチンの対応を示す)．また，並列化前のソースプログラムを用いた各サブルーチンの経過時間及び呼び出し回数をTable 3.1に示す．Table 3.1内のMAINルーチンの経過時間には，初期化及び出力ルーチン(INIT,OUTPUT)の経過時間は含んでいない．Table 3.1より，本コードは，計算量が顕著に多くなるホットスポットがなく，各ルーチンに計算負荷が分散しているルーチン構成であるため，以下の方針に基づいて並列化作業を実施することとした．

(1) データ初期化及び出力ルーチン(INIT,OUTPUT)の並列化はしない．

(2) タイムステップループの中にあるループは，基本的に全て並列化を実施する．

(3) タイムステップループが反復する間は，各ノードは自分が担当するデータのみをもち，通信が必要な場合も必要最小限の通信のみを実施する（ネイバリング通信）．

(4) ルーチン ADVINDX では，各計算ノードが計算を担当する計算領域での計算結果をノード０に集めて全体での値を作成し，その値を各計算ノードに送信する方法を実施する（後述の'ルーチン ADVINDX の並列化' の項を参照）．

(5) 出力は，ノード０のみで実行する．

3.3.2 並列化手法

この節では，本並列化作業において実施した，具体的な並列化箇所及び手法を説明する．

3.3.2.1 並列化用基本定数作成

並列化用の基本定数を MAIN ルーチンで作成し， include file である'mpi.para.inc' にコモン文で設定した．'mpi.para.inc' の内容は， Fig. 3.6を参照のこと．以下， MAIN の，並列化用基本定数作成部分 (Fig. 3.5) を参照しながら，内容を説明する．以下の番号は， Fig. 3.5に示した番号と対応している．

(1) 計算を担当するノードＩＤを設定.

(2) 計算に使用する総ノード数を設定.

(3) 計算を担当するノードＩＤ＋１（ネイバリング通信に使用）を設定.

(4) 計算を担当するノードＩＤ－１（ネイバリング通信に使用）を設定.

(5) ルーチン ADVINDX で２分割した演算範囲の境界となるノード番号を設定.

(6) 各ノードが担当するメッシュ No. の初期値 (JFIR) を設定．各ノードが担当するメッシュ No. の終値 (JLAS) を設定（演算範囲： -jm+2 ～ jm-1）．（サブルーチン PINIT はオリジナルルーチン）

(7) 各ノードが担当する各配列の要素数を設定（演算範囲： -jm+2 ～ jm-1）．各ノードが担当する各配列の要素数を設定（演算範囲： -jm+1 ～ jm）．各ノードが担当するメッシュ No. の初期値 (JFIR1) を設定．各ノードが担当するメッシュ No. の終値 (JLAS1) を設定．（演算範囲： -jm+1 ～ jm）

(8) ルーチン ADVINDX における各ノードが担当するメッシュ No. の初期値 (JS) を設定．各ノードが担当するメッシュ No. の終値 (JE) を設定．（演算範囲： -jm+1 ～ jm-1）

(9) ルーチン ADVINDX での計算ノードの演算順序を設定.

3.3.2.2　ルーチン ADVINDX の並列化

DA-VLASOV コードは，基本的に１次元方向 -jm+1 ～ jm （又は，-jm+2 ～ jm-1）のデイメンジョン範囲をもつ配列を，各ノード毎に分割して並列化を実施している．ルーチン ADVINDX のソースリストを Fig. 3.7に示す．また，ADVINDX から CALL されている HUNT,INNER,NEAREST のプログラム構造は，逐次計算構造なので，これらのルーチンを直接並列化することはできない．よって，上位ルーチンである ADVINDX 内のＤＯループを分割して並列化を実施することにする．

以下に，ADVINDX のプログラム構造及び並列化方針を説明する．以下に示した番号は，Fig. 3.7に示した番号と対応している．

(1) 並列化を実施するため，このＤＯループ（文番号：１００）を分割する．ただし，(6) のＤＯループ分割と合わせて，１次元方向 -jm+1 ～ jm-1 の範囲を分割することになるので，ＤＯループの初期値，終値を ADVINDX 用に設定する．また，(1 ～ jm-1),(0 ～ -jm+1) と２つのループに分割できる様に，各ノードのループ分割範囲を設定する．

(2) ＤＯループ（文番号：１１０）の初期値 (jh,ih)，終値 (jhmax,ihmax) を設定する．

(3) (2) で設定した初期値 (jh,ih)，終値 (jhmax,ihmax) をもつＤＯループである．ここで設定した初期値，終値は，(1) で使用されている各ノードに演算範囲を分割したＤＯループの初期値，終値とは異なるため，このＤＯループ内で計算したデータ（ibase 等）は後で再構築する必要がある．ADVINDX のプログラム構成は，Fig. 3.8に示す様に，多重ループ内の各ノードの演算範囲の境界がオーバーラップしている．

(4) (5) で iold() に数値〔nodef 以外 nodef(=-10000) は初期値〕が設定されている場合は，データ（ibase 等）の設定は実施しない．

(5) データ (ibase,nno,iold： ADVINDX での出力値）の設定．

(6) 並列化を実施するため，このＤＯループ（文番号：２００）を分割する．ただし，ＤＯ文の増分パラメーターが”－１”であるので，ＤＯループの初期値，終値を通常の設定（単調増分）とは逆にして設定する．

次に，ADVINDX の並列化設計 PAD(Problem Analysis Diagram) を，Fig. 3.9に示す．また，修正を実施した ADVINDX のソースリストを Fig. 3.10に示す．以下に，ADVINDX の並列化プログラムの構成を説明する．以下に示した番号は，Fig. 3.10に示した番号と対応している．

(1) 配列 sadxu のＩ方向±１メッシュのデータをネイバリング通信により設定する．

(2) ＤＯループの範囲とその範囲の演算担当ノードを設定する．

演算範囲	担当ノード
1 ～ jm-1	node2 ～ nodes-1
0 ～ -jm+1	node1 ～ 0

(3) 並列化を実施するため，このＤＯループ（文番号：１００）を分割する．各ノード (node2 ～ nodes-1) が担当する範囲の初期値 (JS)，終値 (JE) を設定する．

(4) ノード０に送信するデータの範囲 (JHBMIN ～ JHBMAX) を決定する．

(5) 各ノードで計算した演算範囲１～ jm-1 のデータ (ibase,nno,iold) を，ノード０に送信する．

(6) 各ノードで計算した演算範囲０～ -jm+1 のデータ (ibase,nno,iold) を，ノード０に送信する．

(7) 各ノードから送信されたデータ (ibase,nno,iold) を，ノード０のダミーバッファ IBUF に受信する．

(8) ダミーバッファ IBUF から全演算範囲 -jm+1 ～ jm-1 のデータ (ibase,nno,iold) を作成する．

(9) (8) で作成したデータ (ibase,nno,iold) を，各ノードに送信する．

3.3.2.3　ネイバリング通信

今回並列化した DA-VLASOV コードには，差分計算式が含まれているため，Fig. 3.11に示した様に，各ノードは，隣接するノードより隣接メッシュのデータを受けとらなければならない．Fig. 3.11のノード０の左側及びノード２の右側は，固定境界のため，通信は行わない．ネイバリング通信により並列化したソースリストを，Fig. 3.12に示す．以下，その内容について説明する．

(1) 左側のノードから，右側に隣接しているノードの境界からひとつ外側のメッシュへの通信（例：Fig. 3.11で，ノード０からノード１への通信）．

(2) 右側のノードから，左側に隣接しているノードの境界からひとつ外側のメッシュへの通信（例：Fig. 3.11で，ノード１からノード０への通信）．

3.4　並列化の効果

本コードの８５%程度を並列化した．並列化した実行ファイルを用いて，実行ノード数と経過時間及び速度向上率の関係を Table 3.2に示す．また，Table 3.2の４～６４ノードの計算結果 (out.emode その他) と１ノード（最適化レベル：-O0）との計算結果を比較し，計算結果が一致している事を確認した．

(計算条件)

(1) 並列計算機　：　那珂研 Paragon

(2) 計算体系　：　-127 ～ 128

(3) 最適化レベル　：　－Ｏ４

経過時間の測定結果より，Paragon の３２ノードでの速度向上率は，約８．３倍となった．

3.5 まとめ

本作業では，まず，DA-VLASOV コードが Paragon 上で正常に動作する様に修正を行い，計算結果が，VPP500 と同等であることを確認した．次に，Paragon 向き並列化作業を実施した．並列化作業では，差分式の並列化には「ネイバリング通信」，ルーチン ADVINDX では，各計算ノードが計算を担当する計算領域での計算結果を，ノード０に集めて全体での値を作成し，その値を各計算ノードに送信する方法を用いた．並列化前のオリジナルコードでの経過時間が，全体の約４０％を占める ADVINDX の並列化効率が低かったため，コード全体の並列化効率は，Paragon ３２ノードで約８．３倍となった．

Table 3.1 Distribution of computational costs of original version.

No	ルーチン名	経過時間 (sec)*1	呼び出し回数
1	INIT	2.48E+00	1
2	EMFIELD	4.89E-01	12
3	OUTPUT	2.10E+00	3
4	MISC	1.67E+00	3
5	TDMA1D	7.47E-03	12
6	SETDT	3.14E-02	2
7	LAGRANGE	5.02E+01	2
8	DERIVS	4.90E+01	8
9	DERIV	4.90E+01	8
10	ARTSTATE	6.05E+00	22
11	EXTRAP	7.91E-03	22
12	PERBOUND	3.52E-02	22
13	FXUCAL	8.34E+00	31
14	PERBOXU	1.60E-02	31
15	COLTERM	3.63E+00	8
16	CINT1D6	1.42E+00	130048
17	CINT16	7.07E+00	146304
18	CINT1D2P	2.50E+00	114688
19	FFIELD	4.60E+01	9
20	ADVINDX	2.68E+01	9
21	HUNT	3.66E+00	525440
22	INNER	1.09E+01	1942231
23	NEAREST	9.43E-01	147329
24	WHEREIS	3.02E+00	7
25	RK4	4.28E+01	2
26	EULER	1.16E+01	2
27	MAIN	6.18E+01	1

*1 経過時間は，そのルーチンから呼び出している子ルーチンの経過時間も含んでいる (Fig. 3.4参照).

Table 3.2 Speed up ratio.

計算ノード数	経過時間 (sec)	速度向上率
1	55.00	1.0
4	18.10	3.0
8	10.80	5.1
16	7.50	7.3
32	6.60	8.3
48	6.88	8.0
64	8.10	6.8

速度向上率測定（那珂研 Paragon）

解析条件

(1) 計算体系　：　-127 ～ 128

(2) 計算時間には入出力，初期化ルーチンの経過時間は含めない

(3) 最適化レベル：－O 4

(4) 1ノードで使用した実行ファイルは，並列化プログラミング前のもの

```
(warning)
  PGFTN-W-0164-Overlapping data initializations of eps (vla.f)
    0 inform,   1 warnings,   0 severes, 0 fatal for lagrange
(error)
  PGFTN-S-0034-Syntax error at or near ) (vla.f: 2668)
    0 inform,   0 warnings,   1 severes, 0 fatal for output
```

Fig. 3.1 Warning and error message.

```
(1) warning (lagrange)
    CS
    C       data eps / 1.e-3 /, hmin / 0.0001/
    C       eps data used in "com.h"
    C
            data  hmin / 0.0001/
    CE


(2) error (output)
    CS
    C           write(27,1000)(xp(i),char(9)),(ar(i),char(9)),
    C     &             (au(i),char(9)),(ap(i),char(9))
          write(27,1000) xp(i),char(9),ar(i),char(9),
          &             au(i),char(9),ap(i),char(9)
    CE
```

Fig. 3.2 Modification of subroutine lagrange and output.

```
<<< bump-on-tail instability >>>
  xL = 6.*Pi/0.3
  xK0 = 0.3
  xV = 8.0
  xCol = 0.
  endtime  =0.2
  pr_time  = -0.0001
  pr_dtime = 0.05
  fl1_time  = -0.0001
  fl1_dtime = 0.0
```

Fig. 3.3 Input data for porting test.

```
MAIN---INIT---EMFIELD---TDMA1D
       |     |--ARTSTATE   (3)
       |-OUTPUT(1)-MISC---EMFIELD--TDMA1D
       |-SETDT
       |
       |--LAGRANGE-DERIVS-DERIV---ARTSTATE --|--EXTRAP
       |           |   (2)          |   (3)      | — PERBOUND
       |           |                |            | — FXUCAL
       |           |                |            |--PERBOUNDXU
       |           |                |--EMFIELD---TDMA1D
       |           |                |
       |           |                |--COLTERM---|--CINT16C
       |           |                |            |--CINT1D6
       |           |                |            |--DENSITY
       |           |                |            |--CINT16
       |           |                |            |--CINT1D2P
       |           |                |            |--FKCOL---CINT1D6
       |           |                |
       |           |                |--FFIELD---|--ADVINDX--|--HUNT
       |           |                |   (4)      |           |--INNER
       |           |                |            |           |--NEAREST
       |           |                |            |--FXUCAL
       |           |                |            |--PERBOUNDXU
       |           |                |            |--CINT16
       |           |                |            |--ARTSTATE-(3)
       |           |                |
       |           |                |--WHEREIS--HUNT
       |           |                |--CINT1D2P
       |           |
       |           |--RK4-DERIVS-(2)
       |
       |--EULER--|--ARTSTATE----(3)
       |         |--FFIELD   ----(4)
       |---OUTPUT--- (1)
```

Fig. 3.4 Tree structure of DA-VLASOV code.

```
      call mpi_comm_rank(mpi_comm_world,IAM,ierr)    ---(1)
      call mpi_comm_size(mpi_comm_world,NODES,ierr) ---(2)
      IUP = IAM + 1                                  ---(3)
      IDOWN = IAM - 1                                ---(4)
      NODE2 = NODES/2      --+(5)
      NODE1 = NODE2-1      --+
      IMLN = im + 3
      NJ = jm - 1
      IF (IAM.GE.0.AND.IAM.LE.NODE1) THEN               --+
           IAM1 = NODE1 - IAM                            |
           CALL PINIT(NJ,JF1,JL1,IAM1,NODE2)
           JFIR = (JL1-1)*(-1)
           JLAS = (JF1-1)*(-1)                          (6)
      ELSE IF(IAM.GE.NODE2.AND.IAM.LE.(NODES-1)) THEN
           IAM2 = IAM - NODE2
           CALL PINIT(NJ,JFIR,JLAS,IAM2,NODE2)           |
      END IF                                           --+

      DO 9020 IPR=0,NODE1                               --+
           IAM1 = NODE1 - IPR                            |
           CALL PINIT(NJ,JF1,JL1,IAM1,NODE2)
           JFIRZ = (JL1-1)*(-1)
           JLASZ = (JF1-1)*(-1)
           JFIR1C(IPR)=JFIRZ
           IF(IPR.EQ.0)THEN
             JFIR1C(IPR)=JFIRZ-1
           END IF
           JJLEN(IPR)=IMLN*IIABS(JLASZ-JFIRZ+1)          (7)
           IF(IPR.NE.0) THEN
             JJLEN1(IPR)=JJLEN(IPR)
           ELSE
             JJLEN1(IPR)=IMLN+JJLEN(IPR)
           END IF                                        |
9020  CONTINUE
```

Fig. 3.5 Definition of constants for parallelization(1/2).

```
      DO 9021 IPR=NODE2,NODES-1
           IAM2 = IPR - NODE2
          CALL PINIT(NJ,JFIRZ,JLASZ,IAM2,NODE2)
           JFIR1C(IPR)=JFIRZ
           JJLEN(IPR)=IMLN*IIABS(JLASZ-JFIRZ+1)
           IF(IPR.NE.(NODES-1)) THEN
             JJLEN1(IPR)=JJLEN(IPR)
           ELSE
             JJLEN1(IPR)=IMLN+JJLEN(IPR)
           END IF
 9021 CONTINUE
      JFIR1 = JFIR
      JLAS1 = JLAS
      IF (IAM.GE.0.AND.IAM.LE.NODE1) THEN
         JS    = JLAS
         JE    = JFIR
      ELSE IF(IAM.GE.NODE2.AND.IAM.LE.(NODES-1)) THEN
         JS    = JFIR
         JE    = JLAS
      END IF
      IF (IAM.EQ.0) THEN
         JFIR1 = JFIR1 - 1
         JE    = JE - 1
      END IF
      IF (IAM.EQ.(NODES-1)) THEN
         JLAS1 = JLAS1 + 1
      END IF
C
      IC = 1
      DO 9100 I=1,NODE1
         NORDER(I) = IC
         IC = IC + 1
 9100 CONTINUE
      IC = 1
      DO 9200 J=NODE2,NODES-1
         NORDER(J) = NODES - IC
         IC = IC + 1
 9200 CONTINUE
```

(7)

(8)

(9)

Fig. 3.5 Definition of constants for parallelization(2/2).

```
        PARAMETER(NCPU=256)
        COMMON /PARA/ IAM,NODES,JFIR,JLAS,JFIR1,JLAS1,
      *               JJLEN(0:NCPU-1),
      *               JJLEN1(0:NCPU-1),
      *               IUP,IDOWN,
      *               JS,JE,NORDER(NCPU),
      *               NODE1,NODE2,IMLN,JFIR1C(0:NCPU-1) ,
      *               JPON1(0:NCPU-1)
            integer istatus(mpi_status_size)

NCPU     :  計算ノード数最大値
IAM      :  計算を担当するノードＩＤ
NODES    :  計算に使用するノード総数
JFIR     :  各ノードが担当するメッシュの初期値
                〔演算範囲：-jm+2 ～ jm-1〕
JLAS     :  各ノードが担当するメッシュの終値
                〔演算範囲：-jm+2 ～ jm-1〕
JFIR1    :  各ノードが担当するメッシュの初期値
                〔演算範囲：-jm+1 ～ jm〕
JLAS1    :  各ノードが担当するメッシュの終値
                〔演算範囲：-jm+1 ～ jm〕
JJLEN(0:NCPU-1)  ：各ノードが担当する各配列の要素数
                〔演算範囲：-jm+2 ～ jm-1〕
JJLEN1(0:NCPU-1)：各ノードが担当する各配列の要素数
                〔演算範囲：-jm+1 ～ jm〕
IUP      :  計算を担当するノードＩＤ＋１（ネイバリング通信に使用）
IDOWN    :  計算を担当するノードＩＤ－１（ネイバリング通信に使用）
JS       :  各ノードが担当するメッシュの初期値（ADVINDX）
JE       :  各ノードが担当するメッシュの終値（ADVINDX）
                〔演算範囲：-jm+1 ～ jm-1〕
NORDER(NCPU)：ルーチン ADVINDX での計算ノードの演算順序
NODE1,NODE2 ：ルーチン ADVINDX で２分割した演算範囲の境界となるノード番号
IMLN     ：＝ｉｍ＋３
JFIR1C(0:NCPU-1)：各ノードの JFIR1 計算値を格納
JPON1(0:NCPU-1) ：受信データの位置（MPI_ALLGATHERV 用）
```

Fig. 3.6 Include file 'mpi.para.inc'.

```
      subroutine advindx(sadxu)
           (途中省略)
      do 100 j=1,jm-1        (1)
      do 100 i=0,im-1
          nx  = (sadxu(i,j,1)-eps)/xL
            (x1,u1,x2,u2,x3,u3,x4,u4 の計算)
        call hunt(xph,nxh,x1,ilo)    --+
        call hunt(uph,nuh,u1,jlo)      |
         ih = ilo -1                  (2)
         jh = jlo -jm-1
         ihmax = min(ih+3,im+1)       |
         jhmax = min(jh+3,jm)       --+
          do 110 jk=jh,jhmax   --+(3)
          do 110 ik=ih,ihmax   --+
            if( iold(ik,jk,1) .eq. nodef ) then        (4)
              in = inner(x1,u1,x2,u2,x3,u3,x4,u4,xp(ik),up(jk))
              if( in .eq. 1 ) then
                  near = nearest(x1,u1,x2,u2,x3,u3,x4,u4,xp(ik),up(jk))
                  ibase(ik,jk,1) = i       --+
                  ibase(ik,jk,2) = j         |
                  nadj(ik,jk)    = nx
                  nno(ik,jk)     = near      (5)
                if( near .eq. 1) then
                    iold(ik,jk,1) = i
                    iold(ik,jk,2) = j
                       (途中省略)             |
                endif                      --+
              endif
            endif
  110     continue
  100 continue
      do 200 j=0,-jm+1,-1    (6)
      do 200 i=-1,im-1
           (上部　文番号 100 の DO ループと同等処理)
200   CONTINUE
```

Fig. 3.7　Calculational part before modification (ADVINDX).

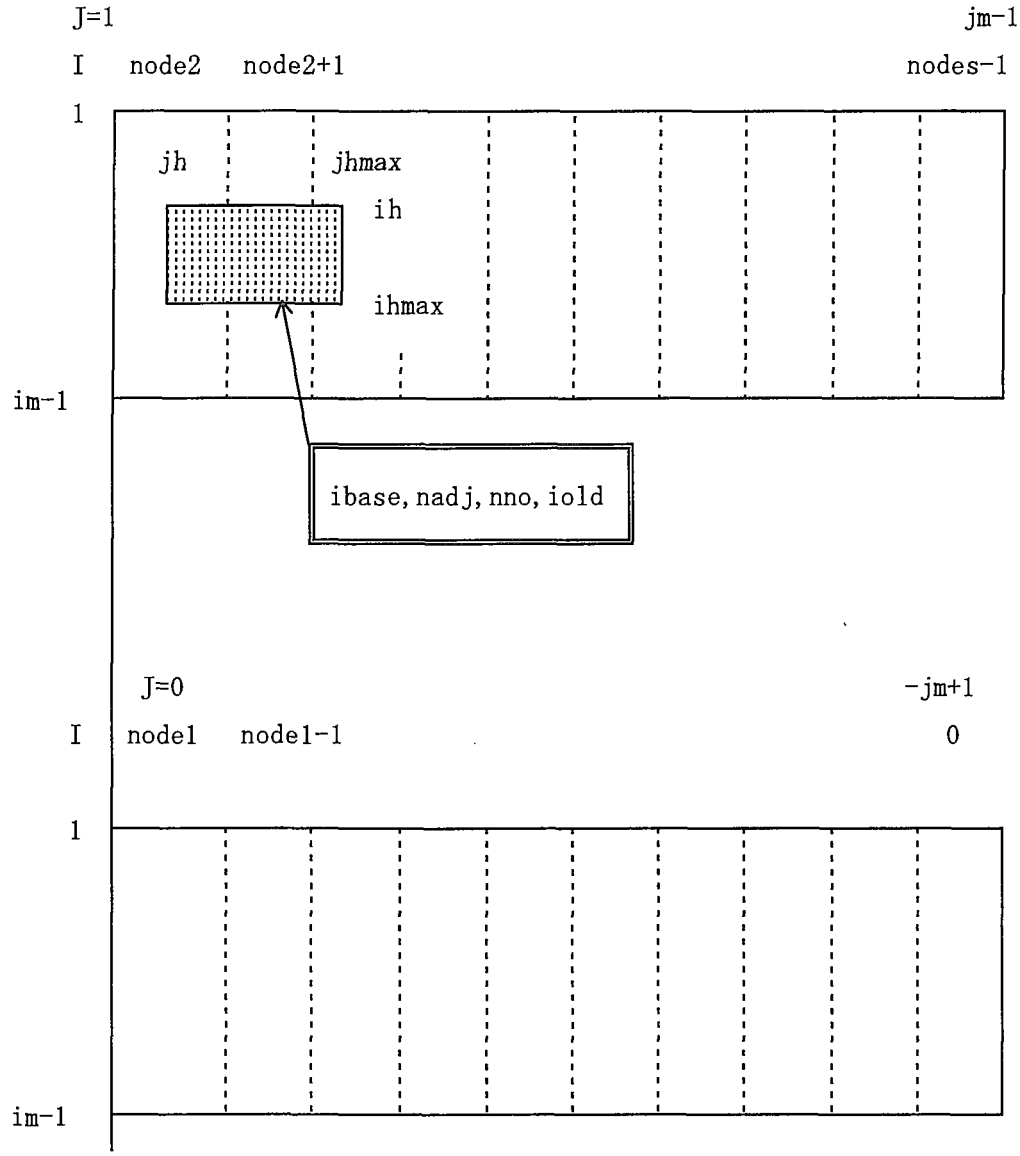


Fig. 3.8 Programming structure of subroutine ADVINDX.

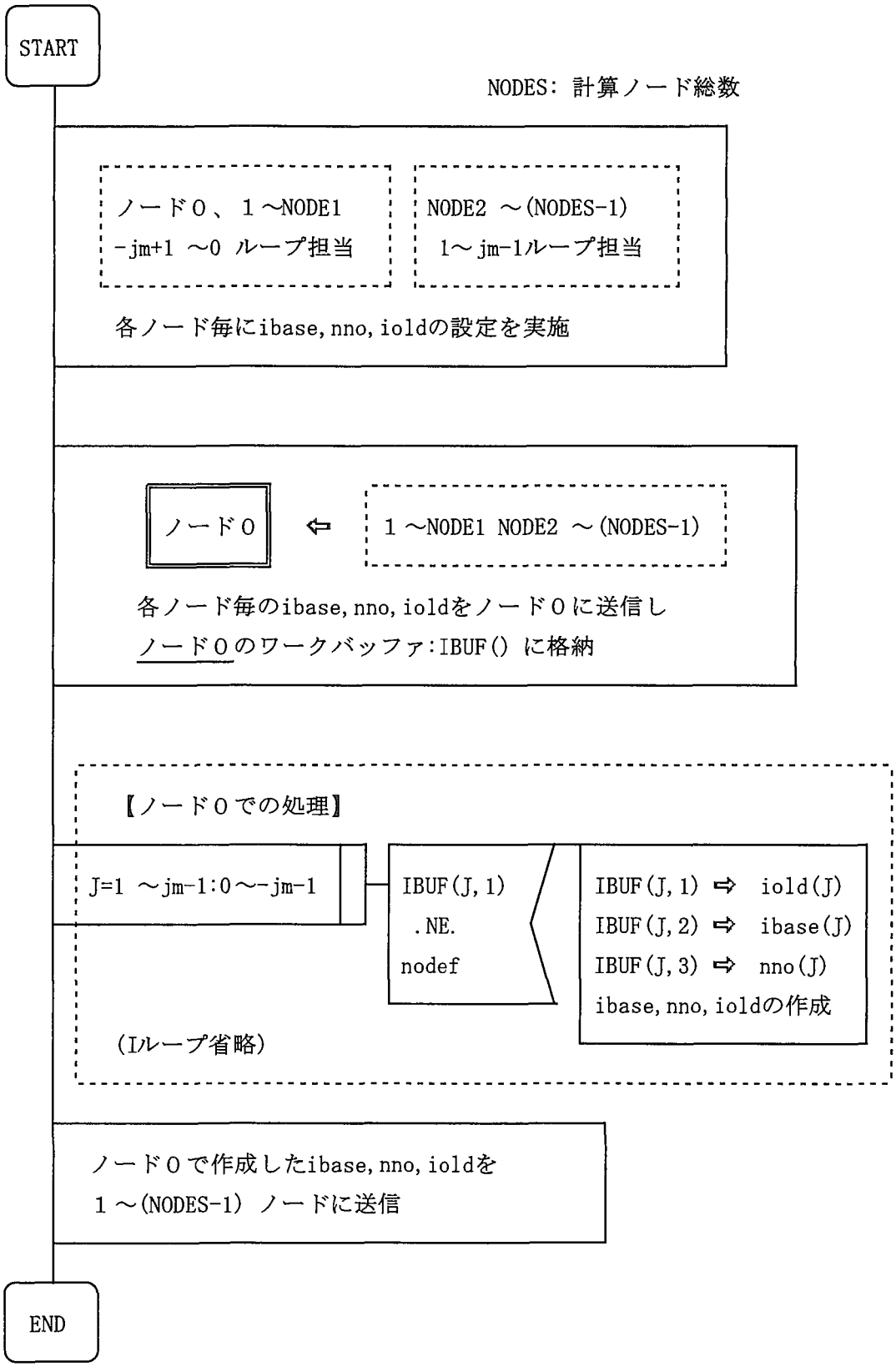


Fig. 3.9 PAD of modified subroutine ADVINDX.

```
      subroutine advindx(sadxu)
          ( 途中省略)
      IF(IAM.GT.0.AND.IAM.LE.(NODES-1)) THEN                    --+
        call mpi_isend(SADXU(-1,JFIR1,1),IMLN,MPI_REAL8,
     1               IDOWN,1000,mpi_comm_world,MSG1,ierr)       |
        call mpi_isend(SADXU(-1,JFIR1,2),IMLN,MPI_REAL8,
     1               IDOWN,2000,mpi_comm_world,MSG2,ierr)
      END IF
      IF(IAM.GE.0.AND.IAM.LT.(NODES-1)) THEN
        call mpi_irecv(SADXU(-1,JLAS1+1,1),IMLN,MPI_REAL8,
     1               IUP,1000,mpi_comm_world,MSGR1,ierr)
        call mpi_irecv(SADXU(-1,JLAS1+1,2),IMLN,MPI_REAL8,
     1               IUP,2000,mpi_comm_world,MSGR2,ierr)
      END IF                                                    (1)
      IF(IAM.GT.0.AND.IAM.LE.(NODES-1)) THEN
        call mpi_wait(MSG1,istatus,ierr)
        call mpi_wait(MSG2,istatus,ierr)
      END IF
      IF(IAM.GE.0.AND.IAM.LT.(NODES-1))THEN
        call mpi_wait(MSGR1,istatus,ierr)                        |
        call mpi_wait(MSGR2,istatus,ierr)                       --+
      END IF
      JHBMIN =  10000
      JHBMAX = -10000
      nxh = im+3
      nuh = jm+jm
CYS
      IF(IAM.GE.NODE2) THEN                ------(2)
CYE
          ilo = 0
          jlo = jm
CYS
C         do 100 j=1,jm-1
          do 100 j=JS,JE                   ------(3)
CYE
          do 100 i=0,im-1
          nx  = (sadxu(i,j,1)-eps)/xL
          x1  = sadxu(i,j,1)-nx*xL
```

Fig. 3.10 Modification in subroutine ADVINDX(1/6).

```
          u1  = sadxu(i,j,2)
          x2  = sadxu(i+1,j,1)-nx*xL
          u2  = sadxu(i+1,j,2)
          x3  = sadxu(i+1,j+1,1)-nx*xL
          u3  = sadxu(i+1,j+1,2)
          x4  = sadxu(i,j+1,1)-nx*xL
          u4  = sadxu(i,j+1,2)
          call hunt(xph,nxh,x1,ilo)
          call hunt(uph,nuh,u1,jlo)
          ih = ilo -1
          jh = jlo -jm-1
          ihmax = min(ih+3,im+1)
          jhmax = min(jh+3,jm)
          do 110 jk=jh,jhmax
          do 110 ik=ih,ihmax
          if( iold(ik,jk,1) .eq. nodef ) then
            in = inner(x1,u1,x2,u2,x3,u3,x4,u4,xp(ik),up(jk))
            if( in .eq. 1 ) then
                   near = nearest(x1,u1,x2,u2,x3,u3,x4,u4,xp(ik),up(jk))
                        ibase(ik,jk,1) = i
                        ibase(ik,jk,2) = j
                        nadj(ik,jk)    = nx
                        nno(ik,jk)     = near
                if( near .eq. 1) then
                  iold(ik,jk,1) = i
                  iold(ik,jk,2) = j
                     (途中省略)
               endif
            endif
          endif
110 continue
```

Fig. 3.10 Modification in subroutine ADVINDX(2/6).

```
CYS
      IF (JH.LT.JHBMIN) JHBMIN = JH          (4)
      IF (JHMAX.GT.JHBMAX) JHBMAX = JHMAX
CYE
  100 continue
CYS
      LSLEN = 4*(im+3)*(JHBMAX-JHBMIN+1)
C
      call mpi_isend(LSLEN,1,MPI_INTEGER,                    --+
     1               0,500+IAM,mpi_comm_world,MSG0,ierr)      |
      call mpi_isend(JHBMIN,1,MPI_INTEGER,
     1               0,1000+IAM,mpi_comm_world,MSG1,ierr)
      call mpi_isend(JHBMAX,1,MPI_INTEGER,
     1               0,2000+IAM,mpi_comm_world,MSG2,ierr)
      call mpi_isend(IOLD(-1,JHBMIN,1),LSLEN,MPI_INTEGER,
     1               0,3000+IAM,mpi_comm_world,MSG3,ierr)      (5)
      call mpi_isend(IOLD(-1,JHBMIN,2),LSLEN,MPI_INTEGER,
     1               0,4000+IAM,mpi_comm_world,MSG4,ierr)
      call mpi_isend(IBASE(-1,JHBMIN,1),LSLEN,MPI_INTEGER,
     1               0,5000+IAM,mpi_comm_world,MSG5,ierr)
      call mpi_isend(IBASE(-1,JHBMIN,2),LSLEN,MPI_INTEGER,
     1               0,6000+IAM,mpi_comm_world,MSG6,ierr)
      call mpi_isend(NNO(-1,JHBMIN),LSLEN,MPI_INTEGER,        |
     1               0,7000+IAM,mpi_comm_world,MSG7,ierr) --+
C
      call mpi_wait(MSG0,istatus,ierr)
      call mpi_wait(MSG1,istatus,ierr)
      call mpi_wait(MSG2,istatus,ierr)
      call mpi_wait(MSG3,istatus,ierr)
      call mpi_wait(MSG4,istatus,ierr)
      call mpi_wait(MSG5,istatus,ierr)
      call mpi_wait(MSG6,istatus,ierr)
      call mpi_wait(MSG7,istatus,ierr)
      END IF
CYE
```

Fig. 3.10 Modification in subroutine ADVINDX(3/6).

```
      IF(IAM.LE.NODE1) THEN
       ilo = 0
       jlo = jm
CYS
C      do 200 j=0,-jm+1,-1
       do 200 j=JS,JE,-1
CYE
       do 200 i=-1,im-1
            (上部, 文番号１００のループと同等計算)
       IF (IAM.NE.0) THEN
         LSLEN = 4*(im+3)*(JHBMAX-JHBMIN+1)                    --+
       call mpi_isend(LSLEN,1,MPI_INTEGER,                      |
     1                0,500+IAM,mpi_comm_world,MSG0,ierr)
       call mpi_isend(JHBMIN,1,MPI_INTEGER,
     1                0,1000+IAM,mpi_comm_world,MSG1,ierr)
       call mpi_isend(JHBMAX,1,MPI_INTEGER,
     1                0,2000+IAM,mpi_comm_world,MSG2,ierr)
       call mpi_isend(IOLD(-1,JHBMIN,1),LSLEN,MPI_INTEGER,
     1                0,3000+IAM,mpi_comm_world,MSG3,ierr)       (6)
       call mpi_isend(IOLD(-1,JHBMIN,2),LSLEN,MPI_INTEGER,
     1                0,4000+IAM,mpi_comm_world,MSG4,ierr)
       call mpi_isend(IBASE(-1,JHBMIN,1),LSLEN,MPI_INTEGER,
     1                0,5000+IAM,mpi_comm_world,MSG5,ierr)
       call mpi_isend(IBASE(-1,JHBMIN,2),LSLEN,MPI_INTEGER,
     1                0,6000+IAM,mpi_comm_world,MSG6,ierr)
       call mpi_isend(NNO(-1,JHBMIN),LSLEN,MPI_INTEGER,
     1                0,7000+IAM,mpi_comm_world,MSG7,ierr)      |
                                                              --+
       call mpi_wait(MSG0,istatus,ierr)
       call mpi_wait(MSG1,istatus,ierr)
       call mpi_wait(MSG2,istatus,ierr)
       call mpi_wait(MSG3,istatus,ierr)
       call mpi_wait(MSG4,istatus,ierr)
       call mpi_wait(MSG5,istatus,ierr)
       call mpi_wait(MSG6,istatus,ierr)
       call mpi_wait(MSG7,istatus,ierr)
     END IF
200 CONTINUE
```

Fig. 3.10 Modification in subroutine ADVINDX(4/6).

```
      IF(IAM.EQ.0) THEN                                              --+
        DO 9000 I=1,NODES-1                                           |
         J = NORDER(I)
        call mpi_irecv(LSLEN,1,MPI_INTEGER,MPI_ANY_SOURCE
     1                    ,500+j,mpi_comm_world,MSGR0,ierr)
        call mpi_irecv(JS1,1,MPI_INTEGER,MPI_ANY_SOURCE
     1                    ,1000+j,mpi_comm_world,MSGR1,ierr)
        call mpi_irecv(JE1,1,MPI_INTEGER,MPI_ANY_SOURCE
     1                    ,2000+j,mpi_comm_world,MSGR2,ierr)
        call mpi_wait(MSGR0,istatus,ierr)
        call mpi_wait(MSGR1,istatus,ierr)
        call mpi_wait(MSGR2,istatus,ierr)
        call mpi_irecv(IBUF(-1,JS1,1),LSLEN,MPI_INTEGER,MPI_ANY_      (7)
       SOURCE,3000+j,mpi_comm_world,MSGR3,ierr)
       call mpi_irecv(IBUF(-1,JS1,2),LSLEN,MPI_INTEGER,MPI_ANY_
       SOURCE,4000+j,mpi_comm_world,MSGR4,ierr)
       call mpi_irecv(IBUF(-1,JS1,3),LSLEN,MPI_INTEGER,MPI_ANY_
       SOURCE,5000+j,mpi_comm_world,MSGR5,ierr)
       call mpi_irecv(IBUF(-1,JS1,4),LSLEN,MPI_INTEGER,MPI_ANY_
       SOURCE,6000+j,mpi_comm_world,MSGR6,ierr)
       call mpi_irecv(IBUF(-1,JS1,5),LSLEN,MPI_INTEGER,MPI_ANY_
       SOURCE,7000+j,mpi_comm_world,MSGR7,ierr)
         CALL MPI_WAIT(MSGR3,istatus,ierr)
         CALL MPI_WAIT(MSGR4,istatus,ierr)
         CALL MPI_WAIT(MSGR5,istatus,ierr)
         CALL MPI_WAIT(MSGR6,istatus,ierr)                            |
         CALL MPI_WAIT(MSGR7,istatus,ierr)                          --+
          DO 9010 JJ = JS1,JE1
          DO 9010 II = -1,im+1
          IF(IBUF(II,JJ,1).NE.nodef) THEN                           --+
               IOLD(II,JJ,1)  = IBUF(II,JJ,1)                        |
               IOLD(II,JJ,2)  = IBUF(II,JJ,2)
               IBASE(II,JJ,1) = IBUF(II,JJ,3)
               IBASE(II,JJ,2) = IBUF(II,JJ,4)                           (8)
               NNO(II,JJ)     = IBUF(II,JJ,5)
          END IF
9010    CONTINUE                                                     |
9000    CONTINUE                                                   --+
```

Fig. 3.10 Modification in subroutine ADVINDX(5/6).

```
      LLEN = 4*(im+3)*(2*jm)
C
        call mpi_bcast(IOLD(-1,-jm+1,1),LLEN,MPI_INTEGER,0,    --+
     1                 mpi_comm_world,ierr)                      |
        call mpi_bcast(IOLD(-1,-jm+1,2),LLEN,MPI_INTEGER,0,
     1                 mpi_comm_world,ierr)
        call mpi_bcast(IBASE(-1,-jm+1,1),LLEN,MPI_INTEGER,0,       (9)
     1                 mpi_comm_world,ierr)
        call mpi_bcast(IBASE(-1,-jm+1,2),LLEN,MPI_INTEGER,0,
     1                 mpi_comm_world,ierr)
        call mpi_bcast(NNO(-1,-jm+1),LLEN,MPI_INTEGER,0,          |
     1                 mpi_comm_world,ierr)                    --+
CYE
        RETURN
      END
```

Fig. 3.10 Modification in subroutine ADVINDX(6/6).

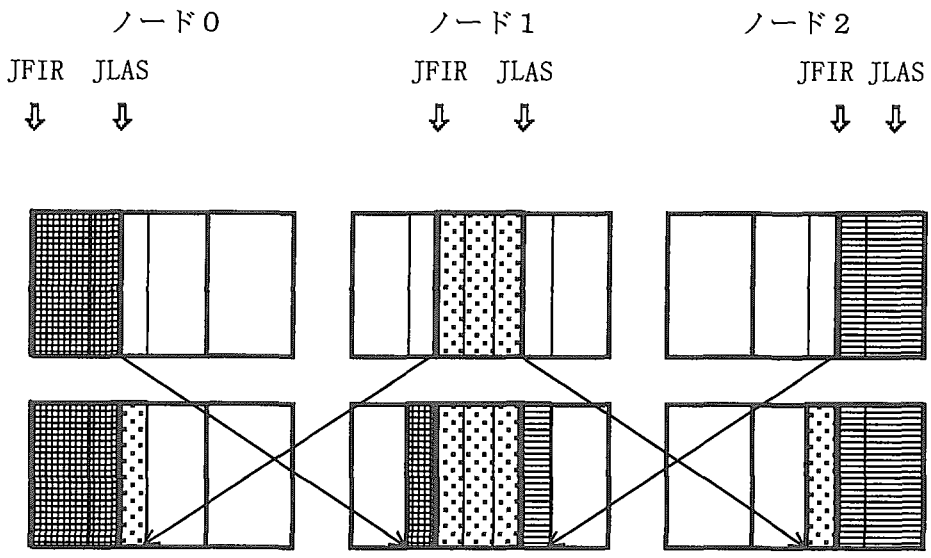


Fig. 3.11 Cnceptional of neighboring communication.

```
      IF(IAM.GT.0.AND.IAM.LE.(NODES-1)) THEN              --+
        call mpi_isend(SSF(-1,JLAS1),IMLN,MPI_REAL8,        |
     1              IUP,1000,mpi_comm_world,MSG11,ierr)
      END IF
      IF(IAM.GE.0.AND.IAM.LT.(NODES-1)) THEN
     call mpi_irecv(SSF(-1,JFIR1-1),IMLN,MPI_REAL8,IDOWN
     1             ,1000,mpi_comm_world,MSGR11,ierr)           (1)
      END IF
      IF(IAM.GT.0.AND.IAM.LE.(NODES-1)) THEN
        call mpi_wait(MSG11,istatus,ierr)
      END IF
      IF(IAM.GE.0.AND.IAM.LT.(NODES-1)) THEN                |
        call mpi_wait(MSGR11,istatus,ierr)                --+
      END IF
      IF(IAM.GE.0.AND.IAM.LT.(NODES-1)) THEN               --+
        call mpi_isend(SSF(-1,JFIR1),IMLN,MPI_REAL8,        |
     1              IDOWN,1200,mpi_comm_world,MSG21,ierr)
      END IF
      IF(IAM.GT.0.AND.IAM.LE.(NODES-1)) THEN
        call mpi_irecv(SSF(-1,JLAS1+1),IMLN,MPI_REAL8,IUP
     1             ,1200,mpi_comm_world,MSGR21,ierr)           (2)
      END IF
      IF(IAM.GE.0.AND.IAM.LT.(NODES-1)) THEN
        call mpi_wait(MSG21,istatus,ierr)
      END IF
      IF(IAM.GT.0.AND.IAM.LE.(NODES-1)) THEN
        call mpi_wait(MSGR21,istatus,ierr)                  |
      END IF                                              --+
```

Fig. 3.12 Neighboring communication part.

参 考 文 献

[1] Takayuki UTSUMI, Tomoaki KUNUGI, James KOGA :A Numerical Method for Solving the One-Dimensional Vlasov-Poisson Equation in Phase Space , 私信, May 1997.

4. MCNP4B2コードの並列化

4.1 概要

本章では，連続エネルギ粒子輸送モンテカルロコードＭＣＮＰ４Ｂ２ (Monte Carlo N-Particle Transport code version 4B2) に対して，ＭＰＩによるインテル製スカラー並列計算機 Paragon 向きの並列化整備作業について述べる．ＭＰＩでの並列化は，平成１０年度作業で実施済であるが，基本ソースの整合性を合わせる為， AP3000 に導入されたソースを基に並列化を実施した [1]．更に，データ通信の効率化を計る為，ノード０から他のノードへのデータ分配通信には，対角送信法を導入した．

ＭＣＮＰコードは，米国のロスアラモス国立研究所で開発された連続エネルギ時間依存の中性子・光子結合モンテカルロ輸送計算コードであり，任意の１～３次元空間を扱うことができる．１９９８年１０月現在，ＭＣＮＰコードは，本作業で並列化したＭＣＮＰ４Ｂ２まで公開されており，中性子，２次ガンマ線，ガンマ線及び電子の輸送計算を行なうことができる [2]．

4.2 並列化

4.2.1 ＭＰＩによる並列化

ＭＣＮＰ４Ｂ２コードのＭＰＩによる並列化作業では，通信バッファ容量の最適化を行うため，ＭＰＩ及び通信バッファの設定にＣ言語を使用し，既存のＰＶＭ版をＭＰＩ版に修正した．ＰＶＭからＭＰＩへの変換には，ＭＣＮＰ４Ａコードを Paragon に整備する時に使用した clib.c ファイル（ＰＶＭから Paragon 用のデータ通信関数ＮＸへの変換ファイル）を，ＰＶＭからＭＰＩに変換する様に修正して使用した．ＰＶＭからＭＰＩに変換する様に修正した clib.c ファイルを， Fig. 4.1に示す．また， List 形式の通信バッファを構成する MsgBuffer， CtlBuffer の関係を Fig. 4.2に示す． CtlBuffer では， MsgBuffer のバッファ領域のポインタのポインタを格納し，バッファ領域の使用，未使用を示すデータも格納する．一方， MsgBuffer では，実際にデータを格納するバッファ領域のポインタ，パック / アンパックの位置を格納する．通信バッファの削除，追加を行うルーチンである delmsgbuffer， addmsgbuffer の概念図を Fig. 4.3に示す． delmsgbuffer では，構造体 MsgBuffer 内の指定部分を削除し， addmsgbuffer では，構造体 MsgBuffer 内の指定部分にバッファ領域を追加する．対角送信法では， Fig. 4.4に示す様に計算ノードが順次データ通信を実施し，データ通信時の経過時間を従来方法より短縮することができる．対角送信法のルーチンである mydistint， mydist ルーチンを Fig. 4.5に示す．

4.2.2 サンプルデータによるテスト計算

ＭＣＮＰ４Ｂ２コードの並列化パッケージに含まれていたサンプルデータを用いて，テスト計算を実施した．テストを実施したサンプルデータは，以下に示す２９種類を計算ノード数５で実施した
(一部，５ノード以外のケースも有り)．

```
 1  simple neutron problem to test some basic operations of mcnp.
 2  three different tallies of the same physical quantity.
 3  many features of the general source.
 4  photons.
 5  toroidal tokamak.
 6  cutoffs, flagging, and variance reduction features.
 7  generate surface source for test No.8 .
 8  use surface source from test No.7
 9  kcode in complicated cells and sdef.
10  general test problem.
11  intertwined super pretzels with s(a,b), mode n p.
12  porosity tool model.
13  check of the volume calculator, rotational symmetry case.
14  test general source in repeated structures.
15  test filled lattice and skewed lattice.
16  test general source in a lattice.
17  kcode in a rectangular finite lattice.
18  kcode in a hexagonal prism lattice.
19  multigroup boltzman-fokker-planck ver.of test No.20.
20  continuous energy electron version of test No.19.
21  electron-photon -generates surface source for test No.22.
22  electron-photon ssr from test No.21
23  forward 80 group electron-photon detector chip problem
24  reflecting lattice.  15x15 at 3.75 w/o u-235 enrichment.
25  test No.24(restart)
26  test No.25(restart)
27  fission surface source from test No.09
28  Coupled Neutron-Photon Adjoint Problem
29  ssr from test No.07; copy of inp08 to test auger production
```

No.1～No.29 のサンプル計算の内，No.1，2は，マルチタスク用のデータを若干修正することにより実行可能となった．また，No.8，29についてはシングルタスクは実行可能で精度も保たれている．しかし，マルチタスクは実行可能であるが，サンプル結果との誤差は大きい（提供されたサンプル実行シェル内に，マルチタスクはサンプル結果との誤差は大きく，検討中とのコメント有り）．No.23，27も，サンプル結果との誤差が若干大きい．その他のサンプル計算は，実行可能であり，これらの計算結果は，ＭＣＮＰ４Ｂ２コードの並列パッケージに含まれていたサンプル計算結果とほぼ一致している．

4.3 並列化の効果

修正したＭＣＮＰ４Ｂ２を用いて，那珂研 Paragon での１ノードから１２８ノードまでの経過時間を測定した．測定結果を Table 4.1に示す． Table 4.1より速度向上率は，６４ノードで３５倍となった．

計算条件

(1) 計算データ　：ユーザー提示テストデータ No.1

(2) ヒストリ数　：100000

(3) 最適化レベル：－Ｏ４

4.4 まとめ

本作業では，連続エネルギ粒子輸送モンテカルロコードＭＣＮＰ４Ｂ２に対して，ＭＰＩによるインテル製スカラー並列計算機 Paragon 向き並列化整備作業を実施した．ＭＰＩの並列化は，既存のＰＶＭ版を基に修正した clib.c ファイルと対角送信法を利用して実施した．那珂研 Paragon での速度向上率は，６４ノードで３５倍となった．更に，計算ノードを１２８ノードにすると，計算時間の減少割合よりデータ通信時間の増加割合の方が大きくなり，速度向上率は３１倍と減少した．サンプル計算 No.8, 29 でのマルチタスクの計算結果の精度の問題は，今後のＭＣＮＰコード整備計画を踏まえて，新たな情報を入手しだい検討する．

Table 4.1 Speed up ratio.

計算ノード数	経過時間（sec）	速度向上率
1	39838	1.0
2	20311	2.0
4	10521	3.8
8	5481	7.3
16	3119	12.8
32	1748	22.8
64	1139	35.0
128	1270	31.0

速度向上率測定（那珂研 Paragon）

解析条件 (1) 計算データ：ユーザー提示テストデータＮｏ．１（ヒストリ数＝100000）

```
/*
 * allPid[0] はマスタープロセス ID
 * allPid[1] から allPid[numSlaves] はスレーブプロセス ID
 */
static pid_t allPid[MAXCPU]; /* 全プロセス ID */
static int        numSlaves; /* 全スレーブプロセスの個数 */
/* 使用中リストヘッダ */
static MsgBuffer uHead = { NULL, 0, 0, 0, InvalidBufID, NULL };
/* 未使用リストヘッダ */
static MsgBuffer fHead = { NULL, 0, 0, 0, InvalidBufID, NULL };
static CtlBuffer ctlBuffer = { /* バッファ制御構造体 */
NULL, 0, 0, InvalidBufID, InvalidBufID, &uHead, &fHead
};
/* データサイズ表 */
static size_t dataSize[] = {
  sizeof(char),
  sizeof(unsigned char),
  sizeof(short),
  sizeof(int),
  sizeof(float),
  2 * sizeof(float),
  sizeof(double),
  2 * sizeof(double)
};

/* アロケート関数 */
static void *alloc(void *ptr, size_t size)
                  【ptr: 確保したメモリの先頭アドレス】
{
  if (ptr == NULL)   【初期設定】
    return malloc(size);
  else
    return realloc(ptr, size);
}
```

Fig. 4.1 Modified file clib.c (1/11).

```
/*
 * ポインタの配列を大きくする.
 * 戻り値: 成功 0
 *         失敗 1
 */
static int enlarge_array(void)
{
  /* ys */
  int mynode;
  /* ye */
  int rv = 0;
  const size_t initsize = 512;    【初期設定値】
  size_t size = ctlBuffer.size == 0 ? initsize : ctlBuffer.size * 2;
  void *p;

  if ((p = alloc(ctlBuffer.array, sizeof(MsgBuffer *) * size)) != NULL) {
    ctlBuffer.array = p;
    ctlBuffer.size = size;
  } else {
    /*  ys  */     【メモリ不足】
    /* fprintf(stderr, "(%ld): not enough memory.\n", mynode()); */
    MPI_Comm_rank(MPI_COMM_WORLD,&mynode);
    fprintf(stderr, "(%ld): not enough memory.\n", mynode);
    /*  ye  */
    rv = 0;
  }

  return rv;
}

/*
 * ポインタの配列長をチェックする.
 * 戻り値: 成功 1
 *         失敗 0
 */
static int check_array(void)
{
  int rv = 1;

  if (ctlBuffer.indx >= ctlBuffer.size && enlarge_array()) rv = 0;

  return rv;
}
```

Fig. 4.1 Modified file clib.c (2/11).

```
/*
 * メッセージバッファ領域を作成する.
 */
static MsgBuffer *create_msgbuffer(void)
{
  MsgBuffer *mb = NULL;
  /* ys */
  int mynode;
  /* ye */
  if (check_array()) {       【Msgbuffer の大きさ】
    if ((mb = alloc(NULL, sizeof(MsgBuffer))) != NULL) {
      mb->buffer = NULL;
      mb->size = 0;
      mb->indx = 0;
      mb->leng = 0;
      mb->bufid = ctlBuffer.indx++;      【代入して加算】
      mb->next = NULL;
    } else {
    /*  ys  */
    /*  fprintf(stderr, "(%ld): not enough memory.\n", mynode()); */
      MPI_Comm_rank(MPI_COMM_WORLD,&mynode);
      fprintf(stderr, "(%ld): not enough memory.\n", mynode);
    /*  ye  */
    }
  }

  return mb;
}

/*
 * リストから p->next を削除する.
 */
static void del_msgbuffer(MsgBuffer *p)
{
  if (p->next->next != NULL)
 *(ctlBuffer.array + p->next->next->bufid) = p;
  p->next = p->next->next;
}
```

Fig. 4.1 Modified file clib.c (3/11).

```
/*
 * リストの p の直後に q を挿入する.
 */
static void add_msgbuffer(MsgBuffer *p, MsgBuffer *q)
{
  if (p->next != NULL) *(ctlBuffer.array + p->next->bufid) = q;
  q->next = p->next;
  *(ctlBuffer.array + q->bufid) = p;
  p->next = q;
}

/*
 * p が指すメッセージバッファを初期化する.
 */
static void clear_msgbuffer(MsgBuffer *p)
{
  p->indx = 0;
  p->leng = 0;
}

/*
 * 新しいメッセージバッファを得る.
 */
static MsgBuffer *get_msgbuffer(void)
{
  MsgBuffer *mb;

  if ((mb = ctlBuffer.freep->next) != NULL) {
    del_msgbuffer(ctlBuffer.freep);
    add_msgbuffer(ctlBuffer.usedp, mb);
    clear_msgbuffer(mb);
  } else if ((mb = create_msgbuffer()) != NULL) {
    add_msgbuffer(ctlBuffer.usedp, mb);
  }

  return mb;
}
```

Fig. 4.1 Modified file clib.c (4/11).

```
/*
 * バッファ領域を大きさをチェックし,
 * 必要なら領域を拡大する.
 * 戻り値: 成功 1
 *         失敗 0
 */
static int enlarge_buffer(MsgBuffer *mb, size_t request)
{
  /* ys */
  int mynode;
  /* ye */
  int rv = 1;   【データが格納されている上限】
  size_t size = mb->indx + request;
  void *p;

  if (mb->size < size) {    【必要ならば領域拡大】
    if ((p = alloc(mb->buffer, size)) != NULL) {
      mb->buffer = p;
      mb->size = size;
    } else {
    /*  ys  */
    /*  fprintf(stderr, "(%ld): not enough memory.\n", mynode()); */
      MPI_Comm_rank(MPI_COMM_WORLD,&mynode);
      fprintf(stderr, "(%ld): not enough memory.\n", mynode);
    /*  ye  */
      rv = 0;
    }
  }

  return rv;
}
```

Fig. 4.1 Modified file clib.c (5/11).

```
/*
 * バッファ領域をアロケートする.
 * 戻り値: 成功 1
 *         失敗 0
 */
static int alloc_buffer(MsgBuffer *mb, size_t size)
{
  /* ys */
  int mynode;
  /* ye */
  int rv = 1;
  void *p;

  if (mb->size < size) {
    if ((p = alloc(mb->buffer, size)) != NULL) {
      mb->buffer = p;
      mb->size = size;
    } else {
    /*  ys  */
    /*  fprintf(stderr, "(%ld): not enough memory.\n", mynode()); */
      MPI_Comm_rank(MPI_COMM_WORLD,&mynode);
      fprintf(stderr, "(%ld): not enough memory.\n", mynode);
    /*  ye  */
      rv = 0;
    }
  }

  return rv;
}

/*
 * 活性受信バッファを解放する.
 */
static void free_rcvbuffer(void)
{
  MsgBuffer *mb;

  if (ctlBuffer.rcvid != InvalidBufID) {
    mb = (*(ctlBuffer.array + ctlBuffer.rcvid))->next;
    del_msgbuffer(*(ctlBuffer.array + mb->bufid));
    add_msgbuffer(ctlBuffer.freep, mb);
    ctlBuffer.rcvid = InvalidBufID;
  }
}
```

Fig. 4.1 Modified file clib.c (6/11).

```
/*
 * 活性バッファがあればそれをクリアし,
 * 新しく活性バッファを用意する.
 */
static int ready_msgbuffer(int *active)
{
  MsgBuffer *mb;
  if (*active == InvalidBufID) {  【真なら活性バッファなし】
    if ((mb = get_msgbuffer()) != NULL) *active = mb->bufid;
  } else {
    clear_msgbuffer((*(ctlBuffer.array + *active))->next);
  }
  return *active + 1;
}
/*
 * 活性受信バッファがあればそれをクリアし,
 * 新しく活性受信バッファを用意する.
 */
static void pvmfinitrecv(int *bufid)
{
  *bufid = ready_msgbuffer(&ctlBuffer.rcvid);
}

/*
 * 活性送信バッファがあればそれをクリアし,
 * 新しく活性送信バッファを用意する.
 *
 * ※ encoding 引数が省略されていることに注意せよ.
 */
void pvmfinitsend_(int *bufid)
{
  *bufid = ready_msgbuffer(&ctlBuffer.sndid);
}
/*
 * 活性送信バッファへデータをパックする.
 *
 * ※ stride 引数が省略されていることに注意せよ.
 */
void pvmfpack_(int *what, void *xp, int *nitem, int *info) 【what: 種類】
{                                                          【xp: 先頭ポインタ】
  size_t bytes = dataSize[*what] * *nitem;                 【nitem: データ数】
  MsgBuffer *mb = (*(ctlBuffer.array + ctlBuffer.sndid))->next;
  if (enlarge_buffer(mb, bytes)) {
    memcpy(mb->buffer + mb->indx, xp, bytes);
    mb->indx += bytes;   【Indx を bytes 増分】
  } else {
    *info = InfoError;
  }
}
```

Fig. 4.1 Modified file clib.c (7/11).

```
/*
 * 活性送信バッファのデータを送信する.
 */
void pvmfsend_(int *tid, int *msgtag, int *info)
{
  MsgBuffer *mb = (*(ctlBuffer.array + ctlBuffer.sndid))->next;

  /* ys  */
  /* csend(*msgtag, mb->buffer, mb->indx, *tid, myptype()); */
  MPI_Send(mb->buffer, mb->indx, MPI_BYTE, *tid, *msgtag,
           MPI_COMM_WORLD);
  /* ye  */
}

/*
 * 活性送信バッファのデータを全スレーブに送信する.
 *
 * ※ ntask 引数, task id 引数が省略されていることに注意せよ.
 *    スレーブの番号は 1 から numSlaves と仮定している.
 */
void pvmfmcast_(int *msgtag, int *info)
{
  static int first = 1;
  static long allnode[MAXCPU];
  long idx;
  MsgBuffer *mb = (*(ctlBuffer.array + ctlBuffer.sndid))->next;

  if (first) {
    for (idx = 0; idx <= numSlaves; ++idx) allnode[idx] = idx;
    first = 0;
  }
  /* ys */
  /* gsendx(*msgtag, mb->buffer, mb->indx, allnode + 1, numSlaves); */
  for (idx=1;idx<=numSlaves; ++idx) {
  MPI_Send(mb->buffer, mb->indx, MPI_BYTE, idx, *msgtag,
           MPI_COMM_WORLD);
  }
  /* ye */
}
```

Fig. 4.1 Modified file clib.c (8/11).

```
/*
 * *msgtag のメッセージを受信する.
 *
 * ※ task id 引数が省略されていることに注意せよ.
 */
/* ys */
void pvmfrecv_(int *msgtag, int *bufid, int *tid )
/* void pvmfrecv_(int *msgtag, int *bufid) */
/* ye */
{
  MsgBuffer *mb;
  long count;
  /* ys */
  MPI_Status stat;
  int count1;
  /* ye */

  /* ys */
  /* cprobe(*msgtag); */
  MPI_Probe(MPI_ANY_SOURCE, *msgtag, MPI_COMM_WORLD, &stat);
  /* ye */
  pvmfinitrecv(bufid); /* 活性受信バッファを用意する. */
  if (*bufid > 0) {
    mb = (*(ctlBuffer.array + *bufid - 1))->next;
    /* ys */
    /* count = (size_t)infocount(); */
    MPI_Get_count(&stat, MPI_BYTE, &count1 );   【受信要素数をカウント】
    count = (size_t)count1;
    /* ye */
    if (alloc_buffer(mb, (size_t)count)) { /* 受信する. */
      /* ys */
      /* crecv(*msgtag, mb->buffer, count); */
      MPI_Recv(mb->buffer, count, MPI_BYTE, MPI_ANY_SOURCE,
               *msgtag, MPI_COMM_WORLD, &stat );
        *tid = stat.MPI_SOURCE;
       /*  printf("tid=%d\n", *tid ); */
      /* ye */
      mb->leng = (size_t)count;
#ifdef SLIM_MEMORY
      if (count == 0) free_rcvbuffer();
#endif
    } else { /* 活性受信バッファを無効にする. */
      free_rcvbuffer();
    }
  }
}
```

Fig. 4.1 Modified file clib.c (9/11).

```
/*
 * 活性受信バッファからデータをアンパックする.
 *
 * ※ stride 引数が省略されていることに注意せよ.
 */
void pvmfunpack_(int *what, void *xp, int *nitem, int *info)
{
  size_t bytes = dataSize[*what] * *nitem;
  MsgBuffer *mb = (*(ctlBuffer.array + ctlBuffer.rcvid))->next;

  memcpy(xp, mb->buffer + mb->indx, bytes);   【*xp にデータコピー】
  mb->indx += bytes;
#ifdef SLIM_MEMORY
  if (mb->indx >= mb->leng) free_rcvbuffer();
#endif /* SLIM_MEMORY */
}

/*
 * メッセージが到着しているかどうか調べる.
 *
 * ※ task id 引数が省略されていることに注意せよ.
 * ※ bufid は負ならばエラー, ゼロなら未到着, 正なら到着を示すが,
 *    正の場合でも返される値はバッファ ID ではないことに注意せよ.
 */
void pvmfprobe_(int *msgtag, int *bufid)
{
  /* ys */
  int flag;
  MPI_Status stat;
  /* *bufid = (int)iprobe(*msgtag); */
  MPI_Iprobe(MPI_ANY_SOURCE, *msgtag, MPI_COMM_WORLD,
             &flag, &stat );
  *bufid = flag;
  /* ye */
}

/*
 * メッセージがどこから来たか調べる.
 *
 * ※ bufid 引数, bytes 引数, msgtag 引数が省略されていることに注意せよ
 * ※ pvmfrecv の呼出し直後に呼び出されることを仮定している.
 */
void pvmfbufinfo_(int *tid)
{
  /* ys */
  MPI_Status stat;
  /*
  *tid = infonode();
  */
  *tid = stat.MPI_SOURCE;
  /* yds */
  /*  printf("(tid=%d\n)", *tid ); */
  /* yde */
  /* ye */
}
```

Fig. 4.1 Modified file clib.c (10/11).

```
/*
 * 活性受信バッファを切り替える.
 */                                                    【データ保持】
void pvmfsetrbuf_(int *bufid, int *oldbuf)   【bufid=0,rcvid → oldbuf】
{                                                  【Invalidid → rcvid】
  *oldbuf = ctlBuffer.rcvid + 1;
  if (*bufid > 0) {
    ctlBuffer.rcvid = *bufid - 1;
  } else {
    ctlBuffer.rcvid = InvalidBufID;
  }
}
/*
 * スレーブ数をマスターから受信する.
 */
void recvnsub_(void)
{
  /* ys */
  int myid;
  MPI_Status stat;
  /* crecv(NsubTag, &numSlaves, sizeof numSlaves); */
  MPI_Bcast(&numSlaves, 1, MPI_INT, MasterID, MPI_COMM_WORLD);
  /* if (mynode() > numSlaves) exit(0); */
  MPI_Comm_rank(MPI_COMM_WORLD, &myid );
  if ( myid  > numSlaves) exit(0);
  /* ye */
}

/*
 * スレーブ数を全スレーブに送信する.
 * numnodes() - 1 個送る.
 *
 * ※ numnodes() が MAXCPU を超えないかどうか検査していないことに注意せよ.
 */
void sendnsub_(int *nsub)
{
  /* ys */
  /* long allnode[MAXCPU];
  long num = numnodes();
  int idx;

  for (idx = 1; idx < num; ++idx) allnode[idx] = idx;
  gsendx(NsubTag, nsub, sizeof *nsub, allnode + 1, num - 1);
  */
  MPI_Bcast(nsub, 1, MPI_INT, MasterID, MPI_COMM_WORLD);
  /* ye */
  numSlaves = *nsub;
}
```

Fig. 4.1 Modified file clib.c (11/11).

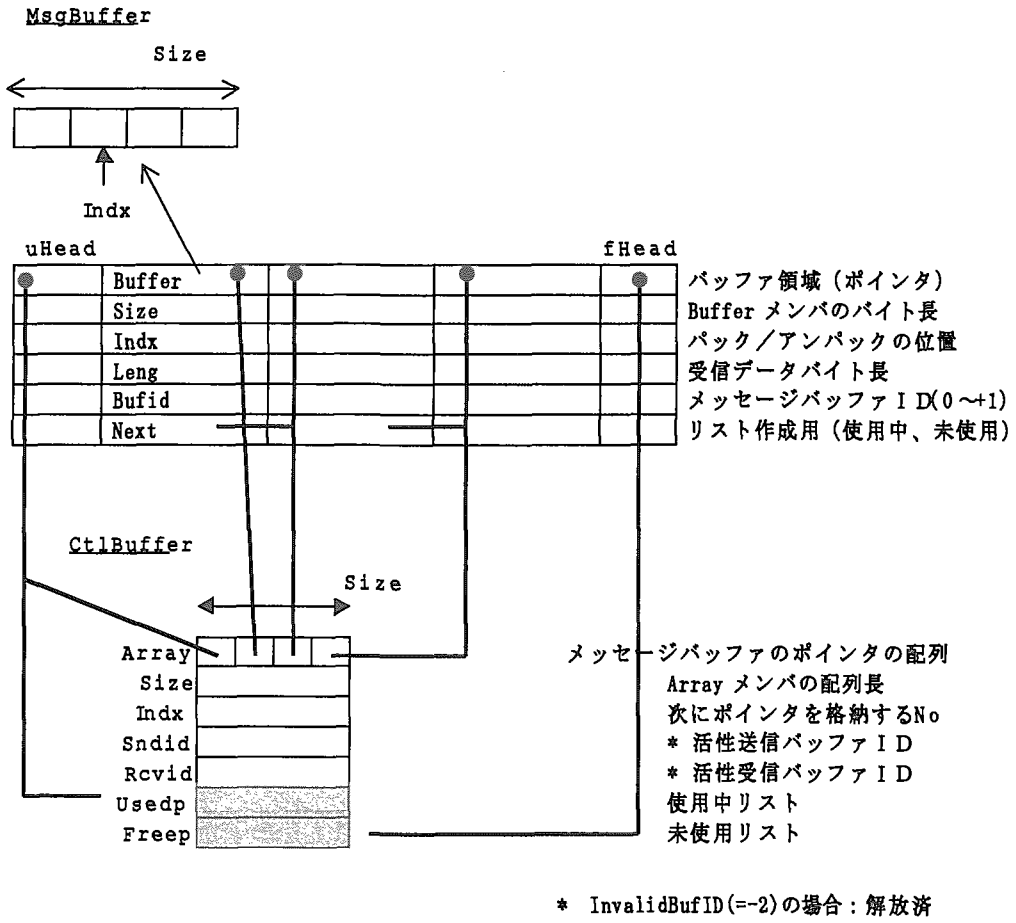


Fig. 4.2 Relation between MsgBuffer and CtlBuffer .

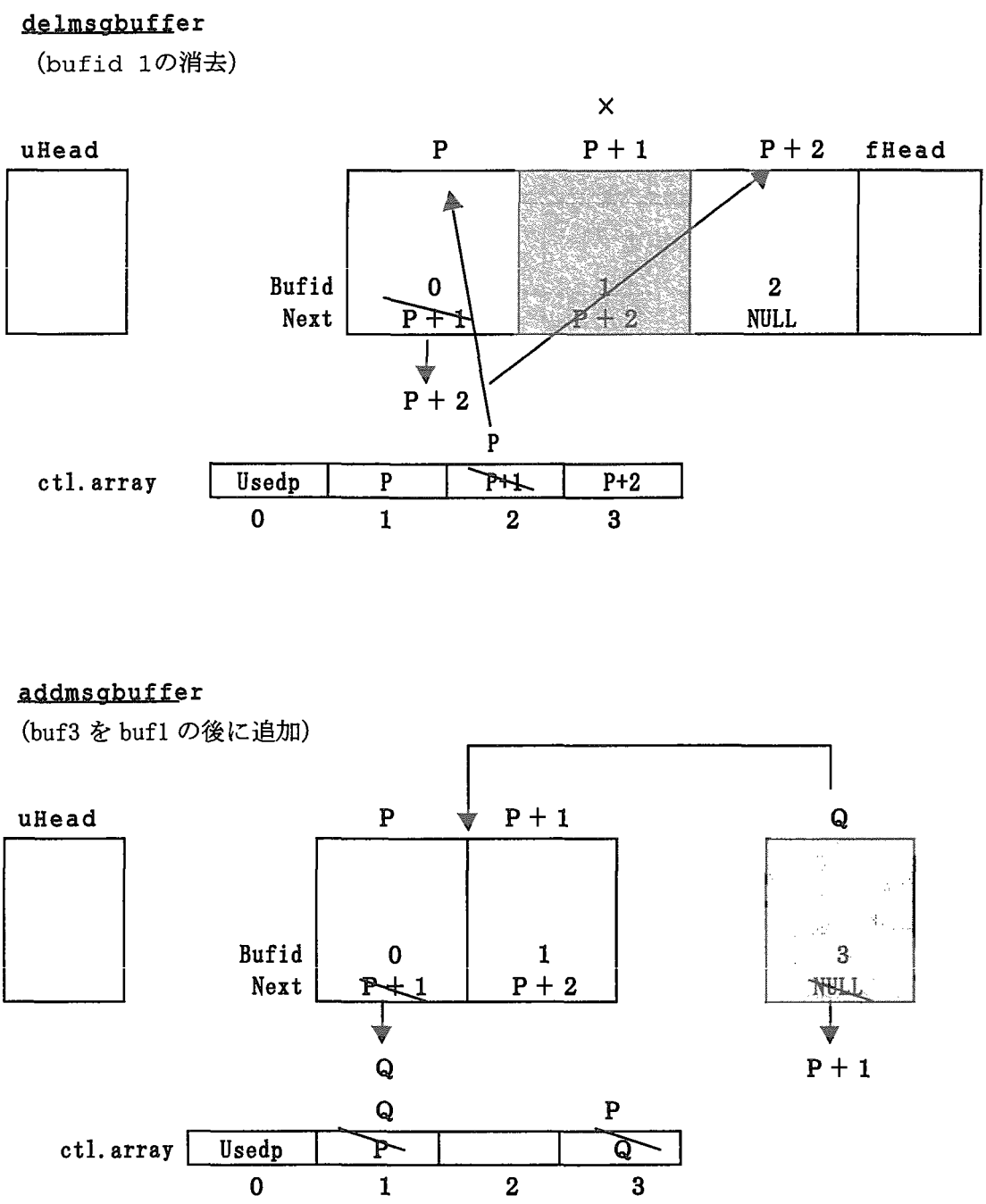
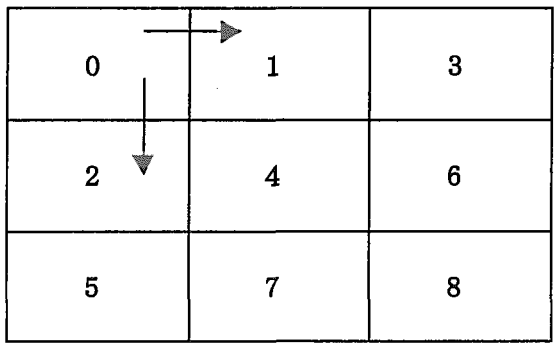


Fig. 4.3 Cnception of delmsgbuffer and addmsgbuffer.

計算ノード数９（３×３）におけるデータ通信



*枠内の数値は計算ノード番号

（１）ＤＯループによる０ノードからのデータ分配(従来方法)

ステップ１：０ノードから１ノードへのデータ通信
ステップ２：０ノードから２ノードへのデータ通信
ステップ３：０ノードから３ノードへのデータ通信
⋮
ステップ８：０ノードから８ノードへのデータ通信

（２）対角送信法によるデータ分配（今回、採用方法）

ステップ１：０ノードから１、２ノードへのデータ通信
ステップ２：１ノードから３、４ノードへのデータ通信
　　２ノードから５ノードへのデータ通信
ステップ３：４ノードから６、７ノードへのデータ通信
ステップ４：６ノードから８ノードへのデータ通信

テスト計測

転送データ　：8000(byte)
計算ノード数：１２８
経過時間　　：データ分配法（１）　９．２４－２(sec)
　　：データ分配法（２）　２．８６－３(sec)

Fig. 4.4　Diagonal communication method.

```
/*
 * プロセッサ数をえる.
 */
void mnump_(int *np, int *in)
{
  /* ys */
  int size;
  /* *np = numnodes(); */
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  *np = size;
  /* ye */
  *in = 0;
}
/**~~~~~~~~~~~~~~~~ 対角送信法 ~~~~~~~~~~~~~~~~**/
/**
 ** 対角送信法
 ** ノード 0 から他のノードへデータを分配する.
 **/

struct Partition {
  long rows;    /* 行数 */
  long cols;    /* 列数 */
  long mynd;    /* ノード番号 */
  long myrw;    /* 行番号 */
  long mycl;    /* 列番号 */
};

static struct Partition Partition;

void mydistinit_(void)
{
  long ndnm, rows, cols, qut, rem;
  /* ys */
  int myid, size;
  /*Partition.mynd = mynode();
  ndnm = numnodes();
  */
  MPI_Comm_rank(MPI_COMM_WORLD, &myid );
  Partition.mynd = myid;
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  ndnm = size;
  nx_app_rect(&rows, &cols);
```

Fig. 4.5 Subroutine mydistint and mydist(1/3).

```
  Partition.myrw = Partition.mynd / cols;
  Partition.mycl = Partition.mynd % cols;
  qut = ndnm / cols;
  rem = ndnm % cols;
  if (!qut) {
    Partition.rows = 1;
    Partition.cols = rem;
  } else if (!rem) {
    Partition.rows = qut;
    Partition.cols = cols;
  } else if (Partition.mycl < rem && Partition.myrw < qut) {
    Partition.rows = qut + 1;
    Partition.cols = cols;
  } else if (Partition.mycl < rem && Partition.myrw == qut) {
    Partition.rows = qut + 1;
    Partition.cols = rem;
  } else {
    Partition.rows = qut;
    Partition.cols = cols;
  }
}

void mydist_(int *msgtag, int *bufid)
{
  int tid;
  int count;
  MsgBuffer *mb;
  /* ys */
  int *idy;
  /* ye */
  if (Partition.myrw || Partition.mycl) {
    /* ys */
    /* pvmfrecv_(msgtag, bufid); */
    pvmfrecv_(msgtag, bufid, idy);
    /* ye */

    mb = (*(ctlBuffer.array + ctlBuffer.rcvid))->next;
```

Fig. 4.5 Subroutine mydistint and mydist(2/3).

```
    if (!Partition.myrw && Partition.mycl < Partition.cols - 1) {
      /* ys */
      /*
      csend(*msgtag, mb->buffer, (long)mb->leng,
            Partition.mynd + 1, myptype());
      */
      MPI_Send(mb->buffer, (long)mb->leng, MPI_BYTE, Partition.mynd + 1,
               *msgtag, MPI_COMM_WORLD);
      /* ye */
    }
    if (Partition.myrw < Partition.rows - 1) {
      /* ys */
      /*
      csend(*msgtag, mb->buffer, (long)mb->leng,
Partition.mynd + Partition.cols, myptype());
      */
      MPI_Send(mb->buffer, (long)mb->leng, MPI_BYTE,
         Partition.mynd + Partition.cols, *msgtag, MPI_COMM_WORLD);
      /* ye */
    }
  } else {
    if (!Partition.myrw && Partition.mycl < Partition.cols - 1) {
      tid = Partition.mynd + 1;
      pvmfsend_(&tid, msgtag, bufid);
    }
    if (Partition.myrw < Partition.rows - 1) {
      tid = Partition.mynd + Partition.cols;
      pvmfsend_(&tid, msgtag, bufid);
    }
  }
}
/**-------------------- 対角送信法 ----------------------**/
```

Fig. 4.5 Subroutine mydistint and mydist(3/3).

参 考 文 献

[1] 原子力コードの高速化（スカラ並列化編）平成10年度作業報告書， JAERI-Data/Code 2000-016， March 2000.

[2] Version 4B Manual 「MCNP-A General Monte Carlo N-Particle Transport Code Version 4B」， LA-12625-M， March 1997.

5. おわりに

計算科学技術推進センター情報システム管理課で実施している原子力コードの高速化作業は，平成11年度に18件の作業を完了した．平成12年度前期にも13件の作業が計画されている．これらの作業は，計算時間の大きい原子力コードを原研が保有する各種スーパーコンピュータ向けに最適なベクトル化，並列化を施す高速化チューニングを行うものであり，コード実行時間の大幅な短縮に寄与している．さらに，単一プロセッサ上ではメモリ不足から実行できないようなジョブを並列化により実行可能にするなど，計算機のスループット向上，ターンアラウンドタイム短縮，それによるユーザの仕事の効率化，計算可能なジョブの範囲の拡大など，計算機の効率的な運用と計算機資源の有効利用に大いに貢献するものと考えている．

本報告書では，高エネルギー核子・中間子輸送計算コードNMTC，ブラソフプラズマシミュレーションコードDA-VLASOV及び中性子・光子結合モンテカルロ輸送計算コードMCNP4B2を対象に実施したParagon向けスカラ並列化作業について記述した．

原研では，平成12年度末に東海研，那珂研及び関西研の各地区のスーパーコンピュータの更新を予定しており，今回の高速化作業では，次期機種への移植性を考慮し，汎用性が高い並列化通信ライブラリMPIを利用した．この結果，各コードとも他の計算機への移植性を有しながらParagon上での高速化効果を期待通り得ることが出来た．

本報告書が原子力コードの高速化に携わる人々に多少なりとも参考になれば幸いである．

謝　辞

本作業を行う上で，作業を依頼された粒子線工学研究室 高田弘氏（2章），光量子シミュレーション研究グループ 内海隆行氏（3章）及び炉心プラズマ第一実験室 西谷健夫氏（4章）には，コード内容の把握に際し御協力頂きました．また，本報告書の作成に当り数値実験グループの渡辺正氏には御指導と御助言をいただきました．さらに，本作業を円滑に遂行するための各種事務処理については山田圭子氏に御協力を頂きました．ここにこれらの方々に感謝の意を表します．最後に，本報告書を執筆する機会を与えて下さいました計算科学技術推進センター長秋元正幸氏，情報システム管理課長藤井実氏，(株) 日立製作所公共情報営業本部部長代理高田尚紀氏に感謝致します．

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

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

（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）

原子力コードの高速化（スカラ並列化編）—平成11年度作業報告書—