JAERI-Data/Code
2000-016

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

2000年3月

箭竹陽一*・足立将晶・久米悦雄・川井　渉*・川崎信夫
根本俊行*・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 2000－016

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 －

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

(2000年2月1日受理)

本報告書は，平成10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10年度に12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，連続エネルギー粒子輸送モンテカルロコードMCNP4B2，連続エネルギー及び多群モデルモンテカルロコードMVP/GMVP及び光量子による固体溶融蒸発シミュレーションコードPHCIP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，汎用トカマク回路シミュレーションプログラムGTCSPを対象に実施したVPP500向けベクトル化作業について，イオン性融体分子動力学計算コードMSP2，渦電流解析コードEDDYCAL，受動的冷却システム試験解析コードTHANPACST2及びMHD平衡コードSELENEJを対象に実施したVPP500向けベクトル／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，連続エネルギー粒子輸送モンテカルロコードMCNP4B2及び軽水炉安全解析コードRELAP5(RELAP5/MOD2/C36-05, RELAP5/MOD3.2.1.2)のAP3000への移植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
* 富士通株式会社
** 株式会社日立製作所

JAERI－Data/Code 2000－016

Vectorization, Parallelization and Porting of Nuclear Codes (Parallelization on Scalar Processors) - Progress Report Fiscal 1998 -

Yo-ichi YATAKE**, Masaaki ADACHI※, Etsuo KUME,
Wataru KAWAI*, Nobuo KAWASAKI※, Toshiyuki NEMOTO*,
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 February 1, 2000)

Several computer codes in the nuclear field have been vectorized, parallelized and transported on the FUJITSU VPP500 system, the AP3000 system and the Paragon system at Center for Promotion of Computational Science and Engineering in Japan Atomic Energy Research Institute. We dealt with 12 codes in fiscal 1998. These results are reported in 3 parts, i.e., the vectorization and parallelization on vector processors part, the parallelization on scalar processors part and the porting part. In this report, we describe the parallelization on scalar processors.

In this parallelization on scalar processors part, the parallelization of Monte Carlo N-Particle Transport code MCNP4B2, Plasma Hydrodynamics code using Cubic Interpolated Propagation Method PHCIP and Vectorized Monte Carlo code (continuous energy model / multi-group model) MVP/GMVP on the Paragon are described.

In the vectorization and parallelization on vector processors part, the vectorization of General Tokamak Circuit Simulation Program code GTCSP, the vectorization and parallelization of Molecular Dynamics Ntv Simulation code MSP2, Eddy Current Analysis code EDDYCAL, Thermal Analysis Code for Test of Passive Cooling System by HENDEL T2 code THANPACST2 and MHD Equilibrium code SELENEJ on the VPP500 are described. In the porting part, the porting of Monte Carlo N-Particle Transport code MCNP4B2 and Reactor Safety Analysis code RELAP5 on the AP3000 are described.

Keywords : MCNP4B2, MVP/GMVP, PHCIP, Parallelization, Paragon, Nuclear Codes

---

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

目　　次

Contents

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の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10年度に12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．なお、例年分冊としていた「ベクトル化編」と「並列化編」については，近年，ベクトル化のみでチューニングを終えるコードが減少し，更なる高速化のため，ベクトル化のみならず並列化をも施すチューニングが増大しており，今年度からこれら2つを合わせて1分冊とすることにした．また，本年度から新たにParagonでの高速化作業成果を加え，編成を前述のように見直している．

本報告書の「スカラ並列化編」では，連続エネルギー粒子輸送モンテカルロコードMCNP4B2，連続エネルギー及び多群モデルモンテカルロコードMVP/GMVP及び光量子による固体溶融蒸発シミュレーションコードPHCIP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，汎用トカマク回路シミュレーションプログラムGTCSPを対象に実施したVPP500向けベクトル化作業について，イオン性融体分子動力学計算コードMSP2，渦電流解析コードEDDYCAL，受動的冷却システム試験解析コードTHANPACST2及びMHD平衡コードSELENEJを対象に実施したVPP500向けベクトル／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，連続エネルギー粒子輸送モンテカルロコードMCNP4B2及び軽水炉安全解析コードRELAP5(RELAP5/MOD2/C36-05, RELAP5/MOD3.2.1.2)のAP3000への移植作業について記述している．なお，平成10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，連続エネルギー粒子輸送モンテカルロコードMCNP4B2を対象に実施したParagonにおける並列化作業について述べる．本作業ではMPIによる並列版の整備及びユーザが独自にカスタマイズしたルーチンのMCNP4B2コードへの組み込みを実施した．この結果，並列化効果はシングル実行時に比較して最大64ノードの並列実行で，約32.5倍の速度向上が得られた．

3章では，連続エネルギー及び多群モデルモンテカルロコードMVP/GMVPを対象に実施したParagonにおける整備作業について述べる．本作業では，既存のMPIによる並列版の不具合である，リスタートファイルの入出力部の改善を実施した．この不具合はリスタートファイル出力時に，3ノード以上の計算でデッドロックを起してしまうというもので，今回の改善により200ノードまでの計算が可能となった．

4章では，光量子による固体溶融蒸発シミュレーションコードPHCIPを対象に実施したParagonにおける並列化作業について述べる．本作業では，差分式の並列化にネイバリング通信を用い，MPIによる並列化を施した．しかしながら、通信回数が多く，並列化の効果は4ノー

ドで約3.4倍にとどまった.

なお，本報告書の作業は箭竹が担当した.

2. MCNP4B2コードの並列化

2.1 概要

本章では，連続エネルギー粒子輸送モンテカルロコードＭＣＮＰ４Ｂ２に対して，ＭＰＩ (Message Passing Interface) によるインテル製スカラ並列計算機 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，既存のＰＶＭ (Parallel Virtual Machine) 版を基に実施する．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はＣ言語を使用し，ＭＰＩ版ＭＣＮＰ４Ｂ２の Paragon での整備を実施した．更に，ユーザが独自にカスタマイズした線源ルーチン (source.f) の組み込みも実施した．

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ー時間依存の中性子・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 [1].

2.2 並列化

2.2.1 並列化整備の内容

本作業では，ＭＰＩを用いたＭＣＮＰ４Ｂ２コードの並列化整備及びユーザが独自にカスタマイズした線源ルーチンの組み込みを実施し，実行シェルスクリプト等の実行環境を整えた．

2.2.2 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.2.1に示す．

2.2.3 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．

2.2.4 線源ルーチンの組み込み

2.2.4.1 線源ルーチンの構造

ユーザが独自にカスタマイズして使用していたＮＸ版の線源ルーチン (source.f) を Fig.2.2に示す．このルーチンは，ＮＸ版のＭＣＮＰ４Ｂコードで使用されていたものであり，ルーチン内

で使用される初期データも，ルーチン内の代入文で設定されている．また，線源ルーチン内の計算で必要とされるコモンデータも，ＭＣＮＰ４ＢとＭＣＮＰ４Ｂ２とでは差異がないため，ＭＣＮＰ４Ｂ２コードの線源ルーチン (source.f) に，ユーザが独自にカスタマイズした部分を組み込み，修正を行った．修正した線源ルーチン (source.f) を Fig.2.3に示す．Fig.2.3はソース上 Fig.2.2と同一であり，ユーザカスタマイズ部をコメント文（CYS-CYE）で囲んである．

2.2.4.2　ユーザデータによるテスト計算

ＭＣＮＰ４Ｂ（ユーザが独自にカスタマイズした線源ルーチンを組み込んだ）で使用されていた計算データを用いてテスト計算を実施した．テストデータは，ユーザより提供された４種類のデータを使用した．ＭＣＮＰ４Ｂ用のデータは，ＭＣＮＰ４Ｂ２にそのまま使用することができ，ＭＣＮＰ４Ｂ２の計算モデルは基本的にＭＣＮＰ４Ｂ２と同一であるため，ＭＣＮＰ４Ｂの計算結果とＭＣＮＰ４Ｂ２の計算結果とを比較した．計算結果を比較検討した結果，両者の計算結果は，ほぼ一致し，有意な差はなかった．

2.3　効果

線源ルーチンを修正した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 Table2.1に示す．Table2.1より速度向上率は，６４ノードで３２．５倍となった．

```
+-----------------------------------------------+
|計算条件                                       |
|   (1) 計算データ   :ユーザ提示テストデータＮｏ．１  |
|   (2) ヒストリ数   :100000                    |
|   (3) 最適化レベル: －04                      |
+-----------------------------------------------+
```

2.4　まとめ

本作業では，連続エネルギー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 ファイルを利用して実施した．更に，ユーザが独自にカスタマイズした線源ルーチンの組み込みを実施した．那珂研 Paragon での速度向上率は，６４ノードで３２．５倍となった．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 2.1 Speed up ratio.

計算ノード数	経過時間（sec）	速度向上率
1	39838	1.0
2	20186	2.0
4	10408	3.8
8	5398	7.4
16	3076	13.0
32	1719	23.2
64	1226	32.5
128	1386	28.7

速度向上率測定（那珂研 Paragon） 並列化コード：ＭＣＮＰ４Ｂ２ (MPI)
解析条件 (1) 計算データ：ユーザ提示テストデータ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)
{
  if (ptr == NULL)
    return malloc(size);
  else
    return realloc(ptr, size);
}
```

Fig. 2.1 Modified file clib.c (1/14).

```
/*
 * 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;
  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  */
    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. 2.1 Modified file clib.c (2/14).

```
/*
 * 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()) {
    if ((mb = alloc(NULL, sizeof(MsgBuffer))) != NULL) {
      mb->buffer = NULL;
      mb->size = 0;
      mb->indx = 0;
      mb->leng = 0;
      mb->bufid = ctlBuffer.indx++;
      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. 2.1 Modified file clib.c (3/14).

```
/*
 * 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. 2.1 Modified file clib.c (4/14).

```
/*
 * 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;
  size_t size = mb->indx + request;
  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;
}
```

Fig. 2.1 Modified file clib.c (5/14).

```
/*
 * 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. 2.1 Modified file clib.c (6/14).

```
/*
 * 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) {
    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)
{
  size_t bytes = dataSize[*what] * *nitem;
  MsgBuffer *mb = (*(ctlBuffer.array + ctlBuffer.sndid))->next;

  if (enlarge_buffer(mb, bytes)) {
    memcpy(mb->buffer + mb->indx, xp, bytes);
    mb->indx += bytes;
  } else {
    *info = InfoError;
  }
}
```

Fig. 2.1 Modified file clib.c (7/14).

```
/*
 * 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. 2.1 Modified file clib.c (8/14).

```
/*
 * *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 );
    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. 2.1 Modified file clib.c (9/14).

```
/*
 * 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);
  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. 2.1 Modified file clib.c (10/14).

```
/*
 * 活性受信バッファを切り替える。
 */
void pvmfsetrbuf_(int *bufid, int *oldbuf)
{
  *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. 2.1 Modified file clib.c (11/14).

```
/*
 * 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. 2.1 Modified file clib.c (12/14).

```
  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. 2.1 Modified file clib.c (13/14).

```
    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. 2.1 Modified file clib.c (14/14).

```
        SUBROUTINE SOURCE
                (途中省略)
c..by crc
      COMMON/PFDE/ PDP(100,100)
C
 1010  IF (J11-00104) 201,200,201
  201 J11=00104
c..by crc
c       R0=SRC(1)
c       R =SRC(2)
c       A =SRC(3)
c       Z0=SRC(4)
c       M0=SRC(5)
c       ENS=SRC(6)
c       TA=SRC(7)
c       TB=SRC(8)
       R0=rdum(1)
       R =rdum(2)
       A =rdum(3)
       Z0=rdum(4)
       M0=rdum(5)
       ENS=rdum(6)
       TA=rdum(7)
       TB=rdum(8)
c..by crc
C        THETA1=-3.1415926
C        THETA2= 3.1415926
        THETA1=-0.3141592
        THETA2= 0.3141592
C        THETA1=-0.1570796
C        THETA2= 0.1570796
C        THETA1=-0.0785398
C        THETA2= 0.0785398
C** SOURCE PARAMETER**

              (途中省略)
  100 R      = RANG(X)
      ERG    = ENRG(1)
      IF(R.LE.PROB(1))              GO  TO  125
      DO  110  I=1,IMAX
      IF(R.GE.PROB(I).AND.R.LE.PROB(I+1))   GO  TO  120
  110 CONTINUE
      I      = IMAX
  120 P1     = PROB(I)
      P2     = PROB(I+1)
      E1     = ENRG(I)
      E2     = ENRG(I+1)
      ERG    = E1+(R-P1)*(E2-E1)/(P2-P1)
  125 CONTINUE
CRC   WRITE(6,620)  I,R,P1,P2,E1,E2,ERG
  620 FORMAT(1X,I5,1P6E12.3)
      RETURN
      END
```

Fig. 2.2 source.f (used in MCNP4B).

```
  subroutine source
                (途中省略)
CYS
      COMMON/PFDE/ PDP(100,100)
C
 1010  IF (J11-00104) 201,200,201
  201 J11=00104
c..by crc
c       R0=SRC(1)
c       R =SRC(2)
c       A =SRC(3)
c       Z0=SRC(4)
c       M0=SRC(5)
c       ENS=SRC(6)
c       TA=SRC(7)
c       TB=SRC(8)
       R0=rdum(1)
       R =rdum(2)
       A =rdum(3)
       Z0=rdum(4)
       M0=rdum(5)
       ENS=rdum(6)
       TA=rdum(7)
       TB=rdum(8)
c..by crc
C        THETA1=-3.1415926
C        THETA2= 3.1415926
        THETA1=-0.3141592
        THETA2= 0.3141592
C        THETA1=-0.1570796
C        THETA2= 0.1570796
C        THETA1=-0.0785398
C        THETA2= 0.0785398
C** SOURCE PARAMETER**
                    (途中省略)

  100 R      = RANG(X)
      ERG    = ENRG(1)
      IF(R.LE.PROB(1))              GO  TO  125
      DO  110  I=1,IMAX
      IF(R.GE.PROB(I).AND.R.LE.PROB(I+1))   GO  TO  120
  110 CONTINUE
      I      = IMAX
  120 P1     = PROB(I)
      P2     = PROB(I+1)
      E1     = ENRG(I)
      E2     = ENRG(I+1)
      ERG    = E1+(R-P1)*(E2-E1)/(P2-P1)
  125 CONTINUE
CRC   WRITE(6,620)  I,R,P1,P2,E1,E2,ERG
  620 FORMAT(1X,I5,1P6E12.3)
CYE
      RETURN
      END
```

Fig. 2.3 source.f (modified for MCNP4B2).

参 考 文 献

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

3. MVP/GMVPコードの整備

3.1 概要

本章では，連続エネルギー及び多群モデルモンテカルロコードMVP/GMVPに対して，ＭＰＩによるインテル製スカラ並列計算機Paragon向きの並列化整備作業について述べる．

ＭＰＩによる並列化は，関西研Paragon上で既存のＰＶＭ版を基に実施されている．よって，本作業では，並列化版の不具合であるリスタートファイルの出力部分の改善を検討した．

MVP/GMVPは，中性子及び光子を対象とした輸送計算を行なうコードである．数値解法は，ベクトル化モンテカルロ法であり，仮想粒子を各バッチ(世代)で処理する．各バッチの仮想粒子の計算は各計算ノードに分割して実施され，最終的な統計量及び各種計算結果は，ひとつのホストノードが，ホスト以外の各ノードから計算データを集めて処理する．各ノードが使用する初期乱数は，ホストノードで作成した後，ホストノード以外の各ノードに転送される．

3.2 リスタートファイルの修正

3.2.1 現行のファイル設定の問題点

現行のMVP/GMVPコードのリスタートファイルの出力は，ホスト以外のノードで計算されたデータをホストノードに転送し，ホストノードがまとめて，リスタートファイルを出力する方法をとっている．Paragonには，並列ファイル入出力を高速に行なうpfsが整備されているが，MVP/GMVPコードは，並列入出力機能が整備されていない他の並列計算機との互換性を考慮し，pfsの採用を見送っている．

Fig.3.1に，リスタートファイルの入出力部及びモンテカルロ計算部の上位ルーチンである，actmppの一部（リスタートファイル出力部）を示す．Fig.3.1内のa)のifブロックでは，ホストノードが実行され，1の部分では，ホストノード自身で計算されたデータのファイル出力を行なっている．また，2の部分では，3の部分でホスト以外のノードから送信されたデータを受信してファイル出力を行なっている．データの送受信には，ブロック型送受信関数を使用している．現行のリスタートファイル出力部のロジックでは，ホストノード自身が，自分自身のデータの書き込み，及び他ノードからの送信データの受信書き込みを連続的に実施しなければならず，ホストノードの負担が大きい．ユーザが提示した基本的なサンプルデータを使用して，リスタートファイル出力を実施したところ，計算ノード数2ノードまでは正常終了したが，それ以上のノード数でリスタートファイル出力を実行すると，リスタートファイル出力部でデッドロックを起こしてしまうことが確認された．

3.2.2　ファイル設定方法の変更

Fig.3.2に，デッドロックを防ぐ為の改良を施したactmppの一部（リスタートファイル出力部）を示す．Fig.3.2の１の部分では，ホストノードが，自分自身のデータのファイル出力を実施している．ホスト以外のノードは，ホストノードがMPI＿BARRIERをcallするまで，MPI＿BARRIERで実行をロックされている．Fig.3.2の２の部分では，ホストノードとそれ以外のノードとの１対１のデータ通信及びファイル出力が，ノード番号の順番通りに実施され，リスタートファイル出力部でのデッドロックを回避している．

3.2.3　サンプルデータによるテスト計算

改良を施したactmppルーチンを使用して，サンプルデータによるテスト計算を実施した．サンプルデータの内容を下記に示す．

```
PWR FULL CORE pin power cal.
仮想粒子数          128000
スキップバッチ数    10
計算バッチ数        200
ファイル名          pwr05pe004.inp
計算ノード数        128
```

テストは，下記の２ケースを実行し，計算結果を比較した．

テストケース　A　：　計算バッチ数100 でリスタートファイルを出力し
　　計算バッチ数101 からリスタート計算．
テストケース　B　：　リスタート計算はせずに，計算バッチ数200 まで
　　計算．

テストケースA，Bの計算結果を，それぞれFig.3.3，Fig.3.4に示す．Fig.3.3，Fig.3.4より，テストケースA，Bの計算結果は一致し，リスタートファイルの入出力が正確に行なわれていることが検証された．しかしながら，計算ノード数を２５６にすると，デッドロックが生じ，リスタートファイルの出力ができなくなる．計算ノード数をパラメータとしたテスト計算を実施したところ，計算ノード数２００までは，リスタートファイルの入出力が，正確に行われていることを確認している．

3.3　まとめ

本作業では，連続エネルギー及び多群モデルモンテカルロコードMVP/GMVPに対して，ＭＰＩによるインテル製スカラ並列計算機Paragon向きの並列化整備作業を実施した．

ＭＰＩによる並列化は，関西研Paragon上で，既存のＰＶＭ版を基に実施されている．よって，本作業では，並列化版の不具合であるリスタートファイルの出力部分の改善を実施し，計算ノード数１２８までのリスタート計算を可能にした．

```
C     ... output data body ...
C
C     ... task 1 outputs data for himself and get data from other task.
C
C
        if ( IDTASK.eq.1 ) then                                    --+
           call RESTOT( IOW, H, 1, IDTASK, TITLE )     ---    1
           do 110 ITSK = 2, NTASK
              call RESTOT( IOW, H, 1, ITSK, TITLE )    ---    2  a)
 110       continue
            call RWIND( IROT )
        else                                                       --+
           call RESTOT( IOW, H, 1, IDTASK, TITLE )     ---    3
        end if
```

Fig. 3.1 Output part for restart file in subroutine ACTMPP.

```
C     ... output data body ...
C
C     ... task 1 outputs data for himself and get data from other task.
C
C
CYDS
         if ( IDTASK.eq.1 ) then                              --+
            call RESTOT( IOW, H, 1, IDTASK, TITLE )             1
         end if
         call MPI_BARRIER(MPI_COMM_WORLD,ierr)               --+
  C
         do 110 ITSK = 2, NTASK                              --+
           if ( IDTASK.eq.1 ) then
              call RESTOT( IOW, H, 1, ITSK, TITLE )
           end if
           if ( IDTASK.eq.ITSK ) then                           2
              call RESTOT( IOW, H, 1, IDTASK, TITLE )
           end if
           call MPI_BARRIER(MPI_COMM_WORLD,ierr)
  110    continue                                              --+
           call RWIND( IROT )
CYDE
```

Fig. 3.2 Modified output part in subroutine ACTMPP.

```
*********************************
*      EVENTS OF NEUTRONS       *
*********************************


                                                COUNT      WEIGHT SUM
SOURCE PARTICLES                              25600000.    0.256000D+08
FISSION NEUTRONS (WHEN JEIGN=0)                     0.    0.000000D+00
NEUTRONS INCREASED BY (N,MN) REACTION           28291.    0.216487D+05
FISSION REACTION PREVENTED (JEIGN=0)                0.    0.000000D+00
(N,GAMMA+X) REACTION                                0.    0.000000D+00
COLLISION                                   937990883.    0.795684D+09
SPLITTING (IMPORTANCE OR WEIGHT WINDOW)             0.    0.000000D+00
SPLITTING PREVENTED                                 0.    0.000000D+00
LEAKAGE                                          4320.    0.368210D+04
ENERGY CUTOFF                                       0.    0.000000D+00
KILLED (IMPORTANCE OR WEIGHT WINDOW)                0.    0.000000D+00
SURVIVED (IMPORTANCE OR WEIGHT WINDOW)              0.    0.000000D+00
KILLED (WEIGHT CUTOFF)                       12850398.    0.695244D+06
SURVIVED (WEIGHT CUTOFF)                       778206.    0.778206D+06
KILLED AT FISGEN AND PHTGEN                         0.    0.000000D+00
ANALOG ABSORPTION                            12773573.    0.984942D+07

NUMBER OF FREE FLIGHT                       4448415047.
NUMBER OF BOUNDARY CROSSING                 3510552164.
NUMBER OF REFLECTION                                0.
```

Fig. 3.3 Output of test case A(include restart calculation).

```
*********************************
*      EVENTS OF NEUTRONS       *
*********************************


                                                COUNT      WEIGHT SUM

SOURCE PARTICLES                              25600000.    0.256000D+08
FISSION NEUTRONS (WHEN JEIGN=0)                     0.    0.000000D+00
NEUTRONS INCREASED BY (N,MN) REACTION           28291.    0.216487D+05
FISSION REACTION PREVENTED (JEIGN=0)                0.    0.000000D+00
(N,GAMMA+X) REACTION                                0.    0.000000D+00
COLLISION                                   937990883.    0.795684D+09
SPLITTING (IMPORTANCE OR WEIGHT WINDOW)             0.    0.000000D+00
SPLITTING PREVENTED                                 0.    0.000000D+00
LEAKAGE                                          4320.    0.368210D+04
ENERGY CUTOFF                                       0.    0.000000D+00
KILLED (IMPORTANCE OR WEIGHT WINDOW)                0.    0.000000D+00
SURVIVED (IMPORTANCE OR WEIGHT WINDOW)              0.    0.000000D+00
KILLED (WEIGHT CUTOFF)                       12850398.    0.695244D+06
SURVIVED (WEIGHT CUTOFF)                       778206.    0.778206D+06
KILLED AT FISGEN AND PHTGEN                         0.    0.000000D+00
ANALOG ABSORPTION                            12773573.    0.984942D+07

NUMBER OF FREE FLIGHT                       4448415047.
NUMBER OF BOUNDARY CROSSING                 3510552164.
NUMBER OF REFLECTION                                0.
```

Fig. 3.4 Output of test case B(no restart calculation).

参 考 文 献

[1] 森　貴正, 中川　正幸： MVP-GMVP 連続エネルギー法及び多群法に基づく汎用中性子・光子輸送計算モンテカルロコード, JAERI-Data/Code 94-007,1994 年 8 月.

[2] 森　貴正, 中川　正幸, 佐々木　誠：汎用モンテカルロコード MVP-GMVP の改良, 私信,1996 年 5 月.

4. PHCIP コードの並列化

4.1 コード概要

本作業の目的は，光量子による固体溶融蒸発シミュレーションコードＰＨＣＩＰのスカラー並列による高速化である．ＰＨＣＩＰコードは，連続体運動方程式を用いて計算を行っている．移流項の数値解法にはＣＩＰ (Cubic Interpolated Propagation) 法及びＣ－ＣＵＰ法を，圧力ポアソン方程式の解法にはＩＣＣＧ (Incomplete CHOLESKY decomposition Conjugated Gradient) 法を用いている．また，本コードの解析体系は２次元となっている．

本章では，インテル製スカラ並列計算機 Paragon に対する，上記ＰＨＣＩＰコードの並列化作業について記述する．ＰＨＣＩＰコードの並列化は，数値解法にＣＩＰ法 [1]，Ｃ－ＣＵＰ法及びその他の差分法で計算を実行している部分については，隣接ノードからのデータを受送信するネイバリング通信を使用した．なお通信ライブラリにはＭＰＩを使用している．

4.2 Paragon への移植

並列化作業の前作業として，ＰＨＣＩＰコードの Paragon （那珂研） １ノードへの移植作業を実施し，下記条件にてテスト計算を実施した．ＰＨＣＩＰコードは，VPP500 でベクトル化されているが，ソースプログラムに特殊なコーディングは施されていない．

(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　　：50*50*1(X-Y-Z)
(d) 計算条件　　：time step iteration ５回

移植時に発生した Warning 及び error を Fig.4.1に示す．また，移植時のソースプログラムの変更箇所 (変更ルーチンは IFORM，CIP3D) を Fig.4.2に示す．次に，ソースプログラムを Fig.4.2に示した通りに変更して，Paragon 上での動作確認を実施した．その時使用した入力データの内容と計算結果の一部 (計算結果の妥当性を評価するために参照した部分) を，それぞれ Fig.4.3，Fig.4.4，Fig.4.5に示す．Fig.4.4，Fig.4.5より，部分的に表記の相違 ((*) 部分) はあるが，これは，WRITE 文の書式指定子にフリーフォーマットを用いている結果であり，数値そのものの計算結果については一致している．これより，ＰＨＣＩＰコードの Paragon １ノードへの移植作業の妥当性を確認した．

4.3 並列化

4.3.1 並列化方針

Fig.4.6に，ＰＨＣＩＰコードのソースツリー図を示す．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 Table4.1に示す．Table4.1内のメイン

プログラム (MAIN) の経過時間には，初期化及び入出力ルーチン (INITLZ, RFORM, IFORM) の経過時間は含んでいない．Table4.1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，計算コストが，各ルーチンに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) データ初期化及び入出力ルーチン (INITLZ,RFORM,IFORM) の並列化はしない．
(2) タイムステップの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) タイムステップの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) 多くの計算ノードを使用した場合の計算パフォーマンスを向上させる為，２次元２方向（ｉ，ｊ方向）に領域分割して並列化を実施する．多ノード計算では，１方向に分割するより，２方向分割の方が各ノード当たりの通信データ量は少なくなる（ただし，通信時の立ち上げ回数は２倍となる）．

4.3.2 並列化手法

4.3.2.1 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.4.8を参照のこと．以下，Fig.4.7を参照しながら，内容を説明する．以下の番号は，Fig.4.7に示した番号と対応している．

1. カルテシアントポロジーにより計算を担当するノードに隣接する上下左右方向のノード番号を設定する．

   idown ： -i 方向のノード番号を設定
   iup ： +i 方向のノード番号を設定
   jdown ： -j 方向のノード番号を設定
   jup ： +j 方向のノード番号を設定

2. 各ノードが担当する i 方向メッシュの初期値 (ISTA) を設定．
   各ノードが担当する i 方向メッシュの終値 (IEND) を設定．
   各ノードが担当する j 方向メッシュの初期値 (JSTA) を設定．
   各ノードが担当する j 方向メッシュの終値 (JEND) を設定．
   【(IM+1)*(JM+1)*(KM+1) 体系 (最外周境界メッシュを含む)】
   各ノードが担当する i 方向メッシュの初期値 (ISTA1) を設定．
   各ノードが担当する i 方向メッシュの終値 (IEND1) を設定．
   各ノードが担当する j 方向メッシュの初期値 (JSTA1) を設定．
   各ノードが担当する j 方向メッシュの終値 (JEND1) を設定．
   【NX*NY*NZ 体系 (最外周境界メッシュは含まない)】

3. itype1()　：　テスト計算用ルーチンの為の派生データの設定.

   【(IM+1)*(JM+1)*(KM+1) 体系 (最外周境界メッシュを含む)】

   itype2()　：　テスト計算用ルーチンの為の派生データの設定.

   【NX*NY*NZ 体系 (最外周境界メッシュは含まない)】

4. cart __ stridei ：ネイバリング通信用の i 方向の派生データの設定.

   cart __ stridej ：ネイバリング通信用の j 方向の派生データの設定.

4.3.2.2　カルテシアントポロジーの設定

ＭＰＩには，カルテシアントポロジーという各プロセスを直角座標系に割り当てる機能がある．このカルテシアントポロジーで設定されたプロセスは自分自身の直角座標系での座標位置の他にも，隣接するプロセスの有無，周期境界条件の設定を自分自身で判断して正確にデータの通信を実施することができる．実際の設定方法を，Fig.4.7 (1/2) 内の 1 に示す．Fig.4.7内に示されている idown,iup,jdown,jup には下記に示す様な計算を担当するノードに隣接する上下左右方向のノード番号が設定される．

```
idown ： -i 方向のノード番号を設定
iup   ： +i 方向のノード番号を設定
jdown ： -j 方向のノード番号を設定
jup   ： +j 方向のノード番号を設定
```

Fig.4.9に簡単な例を示す．枠内の番号はノード番号を示す．周期境界条件が設定されていないとすると，ノード番号 1 に対する下記の変数は

```
idown=  MPI PROC NULL
iup  =  5
jdown=  0
jup  =  2
```

となる．ノード番号に MPI PROC NULL が設定されると，ＭＰＩは通信を行なわず，送受信バッファ内のデータは変更されることなく動作が終了する．

4.3.2.3　ネイバリング通信

今回並列化したＰＨＣＩＰコードは，差分計算式全体の大部分を占めているため，担当のノードは，隣接した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．ＭＰＩの派生データにより並列化したソースリストを，Fig.4.10に示す．以下，Fig.4.10の内容について説明する．

1. － j 方向側のノードから，＋ j 側に隣接しているノードの－ j 側の境界ひとつ外側のメッシュへの通信.

2. ＋ j 方向側のノードから，－ j 側に隣接しているノードの＋ j 側の境界ひとつ外側のメッシュへの通信.

3. －ｉ方向側のノードから，＋ｉ側に隣接しているノードの－ｉ側の境界ひとつ外側のメッシュへの通信.

4. ＋ｉ方向側のノードから，－ｉ側に隣接しているノードの＋ｉ側の境界ひとつ外側のメッシュへの通信.

4.4 並列化の効果

本コードの約９５％の処理を並列化した．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，通信時間及び速度向上率の関係をTable4.2に示す.

（計算条件）

（1）並列計算機　：　関西研 Paragon
（2）計算体系　：　50*50*1　(X*Y*Z)
（3）iteration time　：　１回
（4）最適化レベル　：　－Ｏ４

ＭＰＩを使用して，１ノードあたりの計算領域を，ｉｊ方向に２次元分割して並列化を試みたところ，基本的に通信回数が多過ぎて並列化の効果がでない結果となり，速度向上率は，関西研Paragonの４ノードで約３．４倍となった.

２方向に分割すると，計算ノードが多い場合，１方向のみ分割する場合より１ノードあたりの通信データ量は減少するが，通信関数の立ち上げ回数が２倍となってしまう．　Table 4.2の”通信時間 (sec) *1” が示す通り，２次元分割の並列化により計算ノード数の増加に対する通信時間の増加は抑えられる傾向は示されたが，通信関数の立ち上げ回数の増加による通信時間への影響の方が大きい為，並列化効果は低い結果となった.

4.5 まとめ

本作業では，まず，ＰＨＣＩＰ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」を用い，ｉｊ方向に２次元分割して並列化を実施した．２次元分割の並列化により，計算ノード数の増加に対する通信時間の増加は抑えられる傾向は示されたが，通信関数の立ち上げ回数の増加による通信時間の影響の方が大きい為，並列化効果は低い結果となった.

Table 4.1 Execution time of original PHCIP code.

No	ルーチン名	経過時間 (sec) *1	呼び出し回数
1	INITLZ	8.42E+00	1
2	ARTVIS	1.91E-01	1
3	POISN	2.95E+00	1
4	VELVIS	3.14E-01	1
5	ADVVEL	1.77E-01	1
6	GRDNAV	1.60E+00	6
7	CIP3D	9.08E+00	6
8	FTAN	2.05E-01	1
9	BOUND	5.35E-01	2
10	FARTAN	3.11E-01	1
11	SHIFT	2.14E-01	11
12	RFORM	2.20E+01	10
13	BOUNDR	3.18E-01	5
14	BOUNDU	9.41E-02	5
15	BOUNDV	7.22E-02	5
16	BOUNDW	4.10E-02	5
17	MICCG	5.84E-01	1
18	BOUNDT	1.67E-01	2
19	BOUNDF	3.34E-01	4
20	BOUNDG	1.02E+00	12
21	IFORM	7.49E+00	4
22	ICCG	4.24E-01	1
23	MAIN	1.60E+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．

(測定条件)

(a) 移植先計算機 : Paragon S5MP (関西研 2CPU/1node)

(b) 最適化 : -O4

(c) 計算体系 : 150*150*1(X-Y-Z)

(d) 計算条件 : time step iteration 1回

Table 4.2 Speed up ratio.

計算ノード	経過時間（sec）	通信時間（sec）*1	通信時間（sec）*2	速度向上率
1	11.70			1.0
4	3.40	0.93	0.014	3.4
16	3.80	2.20	0.028	3.1
36	4.80	3.58	0.093	2.4
64	4.80	3.60	0.120	2.4
121	5.10	3.57	0.390	2.3
256	6.60	4.01	1.590	1.8
484	9.80	4.17	4.510	1.2

＊1　差分式、ネイバリング通信部

＊2　ＩＣＣＧ法、ノード0にデータ集約

速度向上率測定（関西研 Paragon）　並列化コード：ＰＨＣＩＰ（ｉｊ方向分割版）

解析条件

(1) 計算体系　：　50*50*1

(2)iteration time ： 1回

(3) 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

(4) 最適化レベル：－Ｏ４

(5) ノード1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-0093-Type conversion of expression performed
 PGFTN-W-0093-Type conversion of expression performed
 PGFTN-W-0093-Type conversion of expression performed
     0 inform,   3 warnings,   0 severes, 0 fatal for cip3d
 (error)
 PGFTN-S-0210-Exponent width not used in the Ew.dEe or Gw.dEe
 (CCUP3D.50*50.f: 2763)
 PGFTN-S-0210-Exponent width not used in the Ew.dEe or Gw.dEe
 (CCUP3D.50*50.f: 2772)
    0 inform,   0 warnings,   2 severes, 0 fatal for iform
```

Fig. 4.1 Warning and error messege.

```
(1) warning (CIP3D)
     C********* S
     C       X1 = -SIGN(1.0,CX)
     C       Y1 = -SIGN(1.0,CY)
     C       Z1 = -SIGN(1.0,CZ)
             X1 = -DSIGN(1.0D0,CX)
             Y1 = -DSIGN(1.0D0,CY)
             Z1 = -DSIGN(1.0D0,CZ)
     C********* E

(2) error (IFORM)
     C********* S
     C6210          FORMAT(1H ,2X,I3,11E12)
     6210          FORMAT(1H ,2X,I3,11E12.4)
     C********* E

     C********* S
     C6230          FORMAT(1H ,2X,I3,10E12)
     6230          FORMAT(1H ,2X,I3,10E12.4)
     C********* E
```

Fig. 4.2 Modification of original code source (1-node).

```
REYNLS=1000.D0,
 UWALL=1.0D0,
 IDIGIT=1,
 GASCON=1.0D0,
 GAMMA=1.4D0,
 NXIN=50,
 NYIN=50,
 NZIN=1,
 DT=0.005D0,
 MXSTEP=5,
 NSTEP=100,
 IPLANE=1,
 NOPRT=1,
 ITMAXP=500,
 EPSP=1.0D-6,
 P0=1.0D0,
 DEN0=1.4D0,
 U0=3.0D00,
 V0=0.0D0,
 W0=0.0D0,
 XLAM=0.0D0
```

Fig. 4.3 Input data for test calculation.

```
 TANGENT TRANFORMATION (IDIGIT=1...ON / 0...OFF) =           1
 GAS CONSTANT (287.1J/KG/K FOR AIR AT 1ATM.) =   1.000000000000000
 RATIO OF SPECIFIC HEAT (1.402 FOR AIR) =   1.400000000000000
 NUMBER OF MESH IN X-DIRECTION (REAL CELL ONLY) =          50
 NUMBER OF MESH IN Y-DIRECTION (REAL CELL ONLY) =          50
 NUMBER OF MESH IN Z-DIRECTION (REAL CELL ONLY) =           1
 MESH WIDTH : DX  (AUTOMATICALY DETERMINED)=  2.0000000000000000E-02 (*)
 MESH WIDTH : DY  (AUTOMATICALY DETERMINED)=  2.0000000000000000E-02 (*)
 MESH WIDTH : DZ  (AUTOMATICALY DETERMINED)=  2.0000000000000000E-02 (*)
 (途中省略)
 ISTEP=           1 PRESSURE ITERATION COUNT=          4
 ISTEP=           2 PRESSURE ITERATION COUNT=          4
 ISTEP=           3 PRESSURE ITERATION COUNT=          4
 ISTEP=           4 PRESSURE ITERATION COUNT=          4
 ISTEP=           5 PRESSURE ITERATION COUNT=          4


 ========  C-CUP : ICYC=    5  ========
 *****  YUN : U-VELOCITY
J=      I=  0           1           2           3           4
    6           7           8           9          10
    51   0.000E+00   2.124E-01   5.880E-01   9.735E-01   1.372E+00
2.487E+00   2.746E+00   2.769E+00   2.768E+00   2.768E+00
    50   0.000E+00   2.124E-01   5.880E-01   9.735E-01   1.372E+00
2.487E+00   2.746E+00   2.769E+00   2.768E+00   2.768E+00
    49   0.000E+00   1.690E-01   5.517E-01   9.554E-01   1.374E+00
2.652E+00   2.966E+00   2.994E+00   2.994E+00   2.994E+00
    48   0.000E+00   1.699E-01   5.526E-01   9.558E-01   1.374E+00
2.656E+00   2.971E+00   3.000E+00   3.000E+00   3.000E+00
    47   0.000E+00   1.700E-01   5.528E-01   9.559E-01   1.374E+00
2.656E+00   2.971E+00   3.000E+00   3.000E+00   3.000E+00
 (以下省略)
```

Fig. 4.4 Result of test calculation on VPP500.

```
 TANGENT TRANFORMATION (IDIGIT=1...ON / 0...OFF) =           1
 GAS CONSTANT (287.1J/KG/K FOR AIR AT 1ATM.) =   1.000000000000000
 RATIO OF SPECIFIC HEAT (1.402 FOR AIR) =   1.400000000000000
 NUMBER OF MESH IN X-DIRECTION (REAL CELL ONLY) =          50
 NUMBER OF MESH IN Y-DIRECTION (REAL CELL ONLY) =          50
 NUMBER OF MESH IN Z-DIRECTION (REAL CELL ONLY) =           1
 MESH WIDTH : DX  (AUTOMATICALY DETERMINED)=  2.0000000000000000E-002 (*)
 MESH WIDTH : DY  (AUTOMATICALY DETERMINED)=  2.0000000000000000E-002 (*)
 MESH WIDTH : DZ  (AUTOMATICALY DETERMINED)=  2.0000000000000000E-002 (*)
 (途中省略)
 ISTEP=           1 PRESSURE ITERATION COUNT=           4
 ISTEP=           2 PRESSURE ITERATION COUNT=           4
 ISTEP=           3 PRESSURE ITERATION COUNT=           4
 ISTEP=           4 PRESSURE ITERATION COUNT=           4
 ISTEP=           5 PRESSURE ITERATION COUNT=           4


 ========  C-CUP : ICYC=    5  ========
 *****  YUN : U-VELOCITY            K=   1 *****
 J=      I=  0           1           2           3           4
    6           7           8           9          10
    51   0.000E+00   2.124E-01   5.880E-01   9.735E-01   1.372E+00
2.487E+00   2.746E+00   2.769E+00   2.768E+00   2.768E+00
    50   0.000E+00   2.124E-01   5.880E-01   9.735E-01   1.372E+00
2.487E+00   2.746E+00   2.769E+00   2.768E+00   2.768E+00
    49   0.000E+00   1.690E-01   5.517E-01   9.554E-01   1.374E+00
2.652E+00   2.966E+00   2.994E+00   2.994E+00   2.994E+00
    48   0.000E+00   1.699E-01   5.526E-01   9.558E-01   1.374E+00
2.656E+00   2.971E+00   3.000E+00   3.000E+00   3.000E+00
    47   0.000E+00   1.700E-01   5.528E-01   9.559E-01   1.374E+00
2.656E+00   2.971E+00   3.000E+00   3.000E+00   3.000E+00
 (以下省略)
```

Fig. 4.5 Result of test calculation on Paragon.

```
MAIN － | － INITLZ   － IFORM
        | － ARTVIS
        | － POISN    － BOUNDR  BOUNDU  BOUNDV BOUNDW MICCG ICCG
        | － VELVIS
        | － ADVVEL
        | － GRDNAV   － BOUNDG
        | － CIP3D    － BOUNDG
        | － FTAN
        | － BOUND    － BOUNDR  BOUNDU  BOUNDV BOUNDW BOUNDT BOUNDF
        | － FARTAN
        | － SHIFT
        | － RFORM
```

Fig. 4.6 Tree structure of PHCIP code.

```
         dims(1) = iprocs
         dims(2) = jprocs
         periods(1) = .false.
         periods(2) = .false.
         ndim = 2
         reorder = .false.
c
      call MPI_CART_CREATE( comm_calc, ndim, dims, periods,
     &      reorder, cart_comm, ierr )
      call MPI_CART_GET( cart_comm, ndim, cart_dims,
     &      cart_periods, cart_coords, ierr )
      call MPI_CART_RANK(cart_comm, cart_coords, cart_rank,
     &        ierr )
      call MPI_COMM_SIZE( cart_comm, cart_size, ierr )
c                                                    +=====+
      call MPI_CART_SHIFT( cart_comm, 0, 1,                 +
     &      idown, iup, ierr )                              1
      call MPI_CART_SHIFT( cart_comm, 1, 1,
     &      jdown, jup, ierr )                             +
c------ set type vector of arrays ------                 +======+
      call PARA_RANGE(0, im, iprocs, cart_coords(1), ista, iend )
      call PARA_RANGE(0, jm, jprocs, cart_coords(2), jsta, jend )
C
      ista1 = ista                                        +=====+
      iend1 = iend                                              +
      jsta1 = jsta
      jend1 = jend
      if (cart_coords(2) .eq. 0 ) then
         jsta1 = 1
      end if                                                    2
      if (cart_coords(1) .eq. 0 ) then
         ista1 = 1
      end if
      if (cart_coords(2) .eq. jprocs-1 ) then
         jend1 = JM - 1
      end if
```

Fig. 4.7 Initialization process for parallel execution (1/2).

```
      if (cart_coords(1) .eq. iprocs-1 ) then
         iend1 = IM - 1
      end if                                                  +
c                                                           +======+
      do irank = 0,isize-1
        icoords(1) = irank/jprocs
        icoords(2) = JMOD(irank,jprocs)
c
        call PARA_RANGE(0, im, iprocs, icoords(1),
     &                  istax, iendx )
        call PARA_RANGE(0, jm, jprocs, icoords(2),
     &                  jstax, jendx )
c
        istay = istax
        iendy = iendx
        jstay = jstax
        jendy = jendx
c
      if (icoords(2) .eq. 0 ) then
         jstax = 1
      end if
      if (icoords(1) .eq. 0 ) then
         istax = 1
      end if
      if (icoords(2) .eq. jprocs-1 ) then
         jendx = JM - 1
      end if
      if (icoords(1) .eq. iprocs-1 ) then
         iendx = IM - 1
      end if
        call PARA_TYPE_BLOCK2(0, im, 0, istay, iendy, jstay, jendy,
     &                  MPI_DOUBLE_PRECISION, itype1(irank))
        call PARA_TYPE_BLOCK2(1, im-1, 1, istax, iendx, jstax, jendx,
     &                  MPI_DOUBLE_PRECISION, itype2(irank))
      end do
                                                  ------   3

      call PARA_TYPE_BLOCK2A(0, im, 1, jlen, MPI_DOUBLE_PRECISION,
     &                       cart_stridej )
      call PARA_TYPE_BLOCK2A(0, jm, ilen, 1, MPI_DOUBLE_PRECISION,
     &                       cart_stridei )
                                                  ------   4
```

Fig. 4.7 Initialization process for parallel execution (2/2).

```
+-----------------------------------------------------------+
 parameter (iprocs=11,jprocs=11)
 parameter (isize=iprocs*jprocs)
 common /para/ group_world,comm_world,group_calc, comm_calc,
&              group_write,comm_write,
&              comm_ctow,comm_wtoc,cart_comm,
&              cart_dims(2),cart_rank,cart_size,
&              cart_coords(2),cart_periods(2),
&              idown,iup,jdown,jup,
&              ista,iend,jsta,jend,
&              ista1,iend1,jsta1,jend1,
&              cart_stridei,cart_stridej,
&              nprocs,myrank,
&              itype1(0:isize-1),itype2(0:isize-1)
+-----------------------------------------------------------+
          iprocs   :   i 方向計算ノード数
          jprocs   :   j 方向計算ノード数
          isize    :   計算ノード総数
          group_world ～ cart_periods(2)
             コミュニケータ、カルテシアントポロジー用変数
          idown    :   計算を担当するノードの－ i 方向の隣接ノード番号
                          (ネイバリング通信に使用)
          iup      :   計算を担当するノードの＋ i 方向の隣接ノード番号
          jdown    :   計算を担当するノードの－ j 方向の隣接ノード番号
          jup      :   計算を担当するノードの＋ j 方向の隣接ノード番号
          ista1    :   各ノードが担当する i 方向メッシュの初期値
                          〔NX*NY*NZ 体系 ( 最外周境界メッシュは含まない) 〕
          iend1    :   各ノードが担当する i 方向メッシュの終値
                          〔NX*NY*NZ 体系 ( 最外周境界メッシュは含まない) 〕
          jsta1    :   各ノードが担当する j 方向メッシュの初期値
                          〔NX*NY*NZ 体系 ( 最外周境界メッシュは含まない) 〕
          jend1    :   各ノードが担当する j 方向メッシュの終値
                          〔NX*NY*NZ 体系 ( 最外周境界メッシュは含まない) 〕
          ista     :   各ノードが担当する i 方向メッシュの初期値
                       〔 (IM+1)*(JM+1)*(KM+1) 体系 ( 最外周境界メッシュを含む) 〕

          iend     :   各ノードが担当する i 方向メッシュの終値
                       〔 (IM+1)*(JM+1)*(KM+1) 体系 ( 最外周境界メッシュを含む) 〕

          jsta     :   各ノードが担当する j 方向メッシュの初期値
                       〔 (IM+1)*(JM+1)*(KM+1) 体系 ( 最外周境界メッシュを含む) 〕

          jend     :   各ノードが担当する J 方向メッシュの終値
                       〔 (IM+1)*(JM+1)*(KM+1) 体系 ( 最外周境界メッシュを含む) 〕

          cart_stridei  : i 方向のネイバリング通信用派生データ
          cart_stridej  : j 方向のネイバリング通信用派生データ
          nprocs   :   Paragon で使用できるノード総数
          myrank   :   計算を担当するノードＩＤ
          itype1(0:isize-1)  :グローバル通信用派生データ
                              〔NX*NY*NZ 体系 ( 最外周境界メッシュは含まない) 〕
          itype2(0:isize-1)  :グローバル通信用派生データ
                       〔 (IM+1)*(JM+1)*(KM+1) 体系 ( 最外周境界メッシュを含む) 〕
```

Fig. 4.8 Include file 'mpi.para.inc'.

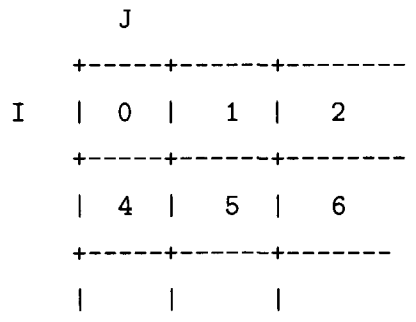

Fig. 4.9 Cartesian Topology.

```
      call MPI_IRECV(A(ista,jsta-1),1,cart_stridei,jdown,1, ===
     &               cart_comm,irecv1,ierr)                        1
      call MPI_ISEND(A(ista,jend),1,cart_stridei,jup,1,
     &               cart_comm,isend1,ierr)                    ===

      call MPI_IRECV(A(ista,jend+1),1,cart_stridei,jup,1,    ===
     &               cart_comm,irecv2,ierr)                        2
      call MPI_ISEND(A(ista,jsta),1,cart_stridei,jdown,1,
     &               cart_comm,isend2,ierr)                    ===

      call MPI_IRECV(A(ista-1,jsta),1,cart_stridej,idown,1, ===
     &               cart_comm,jrecv1,ierr)                        3
      call MPI_ISEND(A(iend,jsta),1,cart_stridej,iup,1,
     &               cart_comm,jsend1,ierr)                    ===

      call MPI_IRECV(A(iend+1,jsta),1,cart_stridej,iup,1,    ===
     &               cart_comm,jrecv2,ierr)                        4
      call MPI_ISEND(A(ista,jsta),1,cart_stridej,idown,1,
     &               cart_comm,jsend2,ierr)                    ===
c
      call MPI_WAIT(irecv1,istatus,ierr)
      call MPI_WAIT(isend1,istatus,ierr)
      call MPI_WAIT(irecv2,istatus,ierr)
      call MPI_WAIT(isend2,istatus,ierr)
      call MPI_WAIT(jrecv1,istatus,ierr)
      call MPI_WAIT(jsend1,istatus,ierr)
      call MPI_WAIT(jrecv2,istatus,ierr)
      call MPI_WAIT(jsend2,istatus,ierr)
```

Fig. 4.10 Communication part of subroutine COMDIM.

参 考 文 献

[1] T.Yabe et al:A universal solver for hyperbolic equations by cubic-polynomial interpolation,Computer Physics Communications ,66(1991)219-242.

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，毎年10数件を順調にこなし，平成10年度に12件の作業を完了，平成11年度にも14件の作業が計画されている．これら作業は，ユーザからの依頼に応じ，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効果により可能にするなど，計算機のスループットの向上，ターンアラウンドタイムの短縮，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，連続エネルギー粒子輸送モンテカルロコードMCNP4B2，連続エネルギー及び多群モデルモンテカルロコードMVP-GMVP及び光量子による固体溶融蒸発シミュレーションコードPHCIPを対象に実施したスカラ並列化作業について記述した．

今回は，スカラ並列による高速化作業，並びにファイルI/Oの不具合修正作業を実施したが，高速化効果が不十分な計算コードもあり，並列化を考慮した計算アルゴリズムの見直し，並列I/Oの導入等が，今後の課題になるものと思われる．また，今後の並列化による高速化作業では，汎用性が高い通信ライブラリとしてMPIの利用を推進する．

最後に，本報告書がこれらの仕事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，作業を依頼された炉心プラズマ第1実験室西谷健夫氏（2章）、炉物理研究グループ長家康展氏（3章）、光量子シミュレーション研究グループ内海隆行氏（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．

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

$1\ \mathrm{eV} = 1.60218 \times 10^{-19}\ \mathrm{J}$

$1\ \mathrm{u} = 1.66054 \times 10^{-27}\ \mathrm{kg}$

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	$N \cdot m$
工率，放射束	ワット	W	J/s
電気量，電荷	クーロン	C	$A \cdot s$
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	$V \cdot s$
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	$cd \cdot sr$
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

$1\ \text{Å} = 0.1\ \mathrm{nm} = 10^{-10}\ \mathrm{m}$

$1\ \mathrm{b} = 100\ \mathrm{fm}^2 = 10^{-28}\ \mathrm{m}^2$

$1\ \mathrm{bar} = 0.1\ \mathrm{MPa} = 10^{5}\ \mathrm{Pa}$

$1\ \mathrm{Gal} = 1\ \mathrm{cm/s}^2 = 10^{-2}\ \mathrm{m/s}^2$

$1\ \mathrm{Ci} = 3.7 \times 10^{10}\ \mathrm{Bq}$

$1\ \mathrm{R} = 2.58 \times 10^{-4}\ \mathrm{C/kg}$

$1\ \mathrm{rad} = 1\ \mathrm{cGy} = 10^{-2}\ \mathrm{Gy}$

$1\ \mathrm{rem} = 1\ \mathrm{cSv} = 10^{-2}\ \mathrm{Sv}$

（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/m²)=10 P(ポアズ)(g/(cm·s))

動粘度　1 m²/s=10⁴St(ストークス)(cm²/s)

圧力	MPa(=10 bar)	kgf/cm²	atm	mmHg(Torr)	lbf/in²(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10年度作業報告書