2010 年度並列処理システム資料

情報処理演習室での PVM3 使用方法

2010 年 10 月 21 日 山崎 勝弘

１　PVM とは

PVM(Parallel Virtual Machine)は，ネットワークで接続された計算機群を仮想的な並列計算機として利用するためのソフトウェアシステムである．対応する計算機には，パーソナルコンピュータから並列計算機／スーパーコンピュータまで，多くの種類のものがある[1]．

PVM では異なったアーキテクチャのホスト間で整数や浮動小数のデータを交換するため，送信されるデータは XDR(eXternal Data Representation) Standard によりエンコードされる．送信側はデータをバッファにパックした後，送信を行い，受信側はバッファにデータを受信した後，アンパックを行いデータを取り出す．基本的に送受信は非同期で，送信と受信が一致しない場合でも送信がブロックされることはない．なお，このデータのパック／アンパックには，データをまとめて送受信することによりオーバーヘッドを減少させる目的もある．

現在、PVM3.4 がインストールされており、使用可能である。文献[2]は PVM3.1 の日本語マニュアルであり、参照すること。また、PVM3.3 のマニュアルが PVM のホームページ[6]にある。最新情報もホームページをアクセスして調べること。

２　環境設定

PVM はコラーニングハウスⅠの情報処理演習室 日立 FLORA 上で使用可能である.PVM の使用に先立って，一度だけ環境設定を行う必要がある．

まず，~/.cshrc に次の 4 行を追加する．ただし，追加する場所は，パス設定”set path =...”の行より後とする．

```
setenv PVM_ROOT /usr/share/pvm3
setenv PVM_DPATH $PVM_ROOT/lib/pvmd
setenv PVM_ARCH ‘$PVM_ROOT/lib/pvmgetarch ‘
set path=( $path $PVM_ROOT/lib $PVM_ROOT/bin/$PVM_ARCH )
```

PVM プログラムを格納するディレクトリ　~/pvm3/bin/LINUX　を作成する．

```
% mkdirhier ~/pvm3/bin/LINUX
```

これまでの　~/.cshrc　の変更を直ちに反映するには，次のようにする．

```
% source ~/.cshrc
```

次に,他のマシンを slave に追加できる様にするために,~/.rhosts　を作成する．情報処理演習室 C32 (coi4s001～coi4s140) の全てのマシンが使用できるように，全てのマシン名を記述する．

```
Coi4s001
Coi4s002
  :
Coi4s140
```

PVM 関連で用意されているマニュアルは、pvm_initsend, pvm_pack, pvm_unpack,pvm_send, pvm_recv, pvm_pkmesg の６つである。% man pvm_pack のように実行すると、マニュアルを見ることができる。

３　PVM の起動と終了

PVM の起動は，コマンドプロンプトから，次のようにする．

```
% pvm
```

正常に起動すれば PVM のプロンプト (pvm>) が表示される．次に，ホストの追加を行う．例えば，現在 coi4s001 上で PVM を起動していて，coi4s002,coi4s003, coi4s004 の 3 台を追加する場合は，PVM のプロンプトで次のようにする．

```
pvm> add coi4s002 coi4s003 coi4s004
```

現在登録されているホストを確認するには，次のようにする．

```
pvm> conf
```

ホストの登録が完了すれば，

```
pvm> quit
```

として一旦 PVM の制御を終了し (PVM 自体は動き続ける)，PVM プログラムの実行を行う．PVM プログラムの実行は，普通のプログラムと同じようにマスター・プログラム名を入力して実行すれば良い．ただし，実行ファイルは通常，ディレクトリ~/pvm3/bin/LINUX に生成されるので，次のようになる．

```
% ~/pvm3/bin/LINUX/<マスター・プログラム名>
```

なお，**実験を終了する際には，必ず PVM のプロンプトで halt を実行して，PVM を停止させる．**

```
pvm> halt
```

ここまでの実行例を示す.

```
% pvm     (PVM を起動)
pvm> conf     (現在登録されているホストを確認)
1 host, 1 data format
                    HOST     DTID     ARCH   SPEED
                coi4s001    40000    LINUX    1000
pvm> add coi4s002 coi4s003 coi4s004     (coi4s002,coi4s003,coi4s004 を追加)
3 successful
                    HOST     DTID
                coi4s002    80000
                coi4s003    c0000
                coi4s004   100000
pvm> conf     (現在登録されているホストを確認)
4 hosts, 1 data format
                    HOST     DTID     ARCH   SPEED
                coi4s001    40000    LINUX    1000
                coi4s002    80000    LINUX    1000
                coi4s003    c0000    LINUX    1000
                coi4s004   100000    LINUX    1000
pvm> quit     (ここで一旦 PVM の制御を終了する. PVM は動き続けている.)

pvmd still running.
%

 (ここで自由に PVM プログラムを実行できる.)

% pvm
pvmd already running.
pvm> halt     (PVM を停止する.)
```

4　プログラム例

PVM 上での並列処理では, まず最初に起動されるマスター・プログラムと, それから呼び出されるスレーブ・プログラムが必要である. さらに, コンパイルを容易にするため, Makefile.aimk を用意する.

1 から 10000 までの和の計算を PVM 上で並列処理するサンプルプログラムを http://www.hpc.se.ritsumei.ac.jp/pvm/pvm.html の上に用意している. このプログラムでは, 例えば 1 から 20 までの和を計算する場合, 図 1 に示すように区間を 4 分割して各スレーブに部分和を計算させ, それをマスターが回収して合計を計算する. なお, プログラム中には端数の処理を行う部分がある. この端数とは, 図 2 に網掛けで示すような, プロセッサに等分割した後の余りのことである. sum_master_s.c がマスター・プログラムで, sum_slave_s.c がスレーブ・プログラムである.

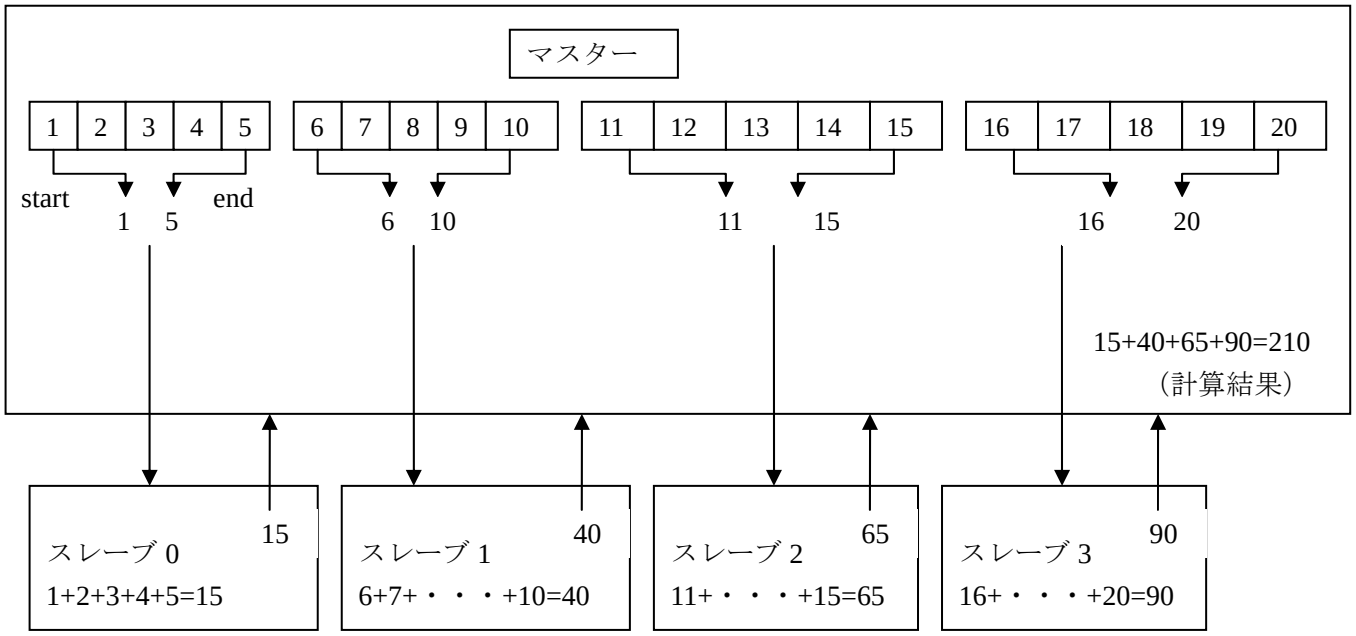

図 1：1 から 20 までの和の並列計算

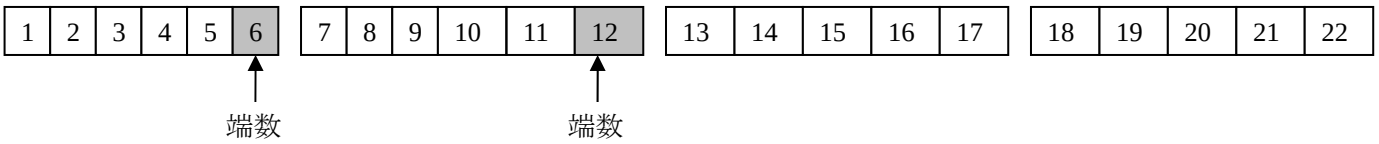

図 2：端数の処理（1 から 22 までの和の計算の場合）

これらのソースファイルと Makefile.aimk と second.c を，pvm3 とは別のディレクトリにセーブし，

```
% aimk
```

を実行すれば，マスター・プログラムとスレーブ・プログラムのコンパイルができ，ディレクトリ~/pvm3/bin/LINUX の中にバイナリファイルが生成される．この LINUX はアーキテクチャ名で，コンパイルしたマシンに応じた名前になる．

万一，ディレクトリ~/pvm3/bin/LINUX の中にバイナリファイルが生成されなかった場合は，次のようにしてコンパイルをやり直すと良い．

```
% aimk clean;aimk
```

実行は，あらかじめ PVM の起動とホストの登録を済ませた後，必ずホームで次のように入力する．

```
% ~/pvm3/bin/LINUX/sum_master_s
```

結果が画面に表示される．実行例を示す．

```
   1 -  2500:  3126250   (TID:80001)
2501 -  5000:  9376250   (TID:c0001)
5001 -  7500: 15626250   (TID:100001)
7501 - 10000: 21876250   (TID:40003)
       Total: 50005000
```

1行目は，1から2500までの部分和が3126250で，TID（タスクID）が80001であるスレーブにより計算されたことを意味する．このTIDは実行毎に変化する．最終的な結果は部分和の合計の50005000となる．

なお，このプログラムは，処理を等分して各ホストに割り当てる静的負荷分散を用いている．しかし，一般にPVMでは処理速度が異なる複数の計算機を使用するため，実行時に処理が空いているホストに次々に処理を割り当てる動的負荷分散が必要となる．動的負荷分散を行うサンプルはhttp://www.hpc.se.ritsumei.ac.jp/pvm/pvm.htmlに用意している．sum_master_d.cがマスター・プログラムで，sum_slave_d.cがスレーブ・プログラムである．

５　並列プログラミング

５．１　コンパイル

PVM上のプログラムのコンパイルには普通のコンパイラ（ccやgcc）を用いる．ただし，PVMのヘッダファイルをインクルードするとともに，ライブラリをリンクする必要がある．通常はaimkにより，Makefile.aimkを用いてコンパイルを行う．

５．２　スレーブ・プログラムの生成

マスター・プログラムから次のようにしてスレーブ・プログラムを生成する．

```
pvm_spawn("slave", (char**)0, 0, "", 4, tids);
```

ここで，slaveは生成するスレーブ・プログラム名で，4は生成する数，tidsは，生成したスレーブ・プログラムのタスクIDが格納されるint型の配列である．

５．３　メッセージの送受信

メッセージの送信は，初期化，パック，送信という手順で行う．メッセージの受信は，受信，アンパックという手順で行う．ただし，パック，アンパックの順序と内容は対応している必要がある．
int型の変数iとfloat型の配列f（要素数4）を送信する手順は，次のようになる．

```
pvm_initsend(PvmDataDefault);   /* 初期化 */
pvm_pkint(&i, 1, 1);            /* iをパック */
pvm_pkfloat(f, 4, 1);           /* f[0]～f[3]をパック */
pvm_send(tid1, 101);            /* 送信 */
```

tid１は受信側のタスクIDである．これを受信する手順は，次のようになる．

```
pvm_recv(tid2, 101);      /* 受信 */
pvm_upkint(&i, 1, 1);     /* iをアンパック */
pvm_upkfloat(f, 4, 1);    /* f[0]～f[3]をアンパック */
```

tid2は送信側のタスクIDである．マスターではpvm_spawnで得たtidsから各スレーブのタスクIDを知ることができ，スレーブではpvm_parent()関数によりマスターのタスクIDを知ることができる．101はメッセージのラベルで任意の値を指定できる．通信はラベルが一致するもの同士で行われる．なお，通信には他にも一度に複数の相手に送信を行う関数なども用意されている．詳細は文献[2,4]参照．

５．４　シグナルの処理

PVM プログラムでは，何らかの原因によりプログラムが中断された場合にタスクが残ったままになるのを避けるため，シグナルの処理を行った方が良い．詳細はプログラム例と文献[4]参照．

PVM 上で動作しているタスクを確認するには PVM のプロンプトで次のようにする．

```
pvm> ps
```

もし，PVM 上にタスクが残ったままになった場合は，次のようにして終了させる．

```
pvm> kill <タスク ID>
```

６　注意事項

1. 最初に，実験で使用するマシンの電源が入っていることを確認せよ．
2. マシンの負荷（シングルユーザかマルチユーザかなど）によって，実行時間が異なる．各データに対して 10 回以上実行し，代表的な値を有効数字 3 桁程度で記録せよ．10 回の平均を実行時間とすること．
3. 静的，動的，配列版それぞれで実行するときは，別々のディレクトリにソースファイルと Makefile.aimk と second.c を入れること．

質問事項があれば，TA の上野君、橋爪君、岩井君(ri001065,ri007061,ri001061@ed.ritsumei.ac.jp)と山崎(yamazaki@se.ritsumei.ac.jp)までメールを送ること．可能な範囲で，TA が対応します．

4. 付録のサンプルプログラムが山崎研ホームページ(http://www.hpc.se.ritsumei.ac.jp/)にあります．必要なファイルをダウンロードして下さい。コピーペーストは不可。コピーペーストすると、Makefile のタブがスペースになり、コンパイルエラーとなります。また，PVM 関連の卒論として，樋口君（2001 年 3 月），壁谷君・神崎君・西谷さん（2000 年 3 月）の卒論が同じホームページにあります．

参考文献

[1] Geist A. 他: ``PVM: Parallel Virtual Machine'', MIT Press, 1994.

[2] Geist A. 他（村田 英明 訳）: ``PVM3 ユーザーズガイド&リファレンス　マニュアル 日本語版'', 1995.
(PS ファイル： http://www.hpc.se.ritsumei.ac.jp/pvm/jpvm-020528.ps)
(PDF ファイル： http://www.hpc.se.ritsumei.ac.jp/pvm/jpvm-020528.pdf)

[3] B.Wilkinson and M.Allen: Parallel Programming, Prentice-Hall, 1999.
飯塚、緑川　訳：並列プログラミング入門、丸善、2000.

[4] B.Wilkinson and M.Allen: Parallel Programming, 2nd edition, Prentice-Hall, 2004.

[5] RAINBOW Guide 制作プロジェクト: ``RAINBOW Guide -UNIX 操作入門編-'', 立命館大学 総合情報センター教育研究システム課 分室

[6] PVM ホームページ: http://www.csm.ornl.gov/pvm/

付録

(1)sum_master_s.c 静的マスタープログラム

```
#include <stdio.h>
#include <signal.h>
#include "pvm3.h"

#define SLAVE      4    /* 起動するスレーブ数 */
#define N      10000    /* 1 からこの数までの和を求める */

int Nslave;             /* 起動したスレーブ数 */
int Tids[SLAVE];        /* 各スレーブのタスク ID */

/* シグナルの処理 */
void exit_handler(void)
{
    int sn;
    /* 全スレーブを終了させる */
    for(sn = 0; sn < Nslave; sn++)
        pvm_kill(Tids[sn]);

    pvm_exit();   /* PVM から離脱 */
    exit(1);      /* プログラム異常終了 */
}

int main()
{
    int i;
    int start, step, end;
    int qt, rm;
    int lsum, sum = 0;

    /* スレーブ・プログラムを起動 */
    Nslave = pvm_spawn("sum_slave_s", (char**)0, 0, "", SLAVE, Tids);
    if(Nslave <= 0){
        fputs("PVM: Can't start slaves¥n", stderr);   /* エラーの表示 */
        return 1;                                     /* プログラム異常終了 */
    }

    /* シグナルの処理 */
    signal(SIGINT, (void*)exit_handler);
    signal(SIGTERM, (void*)exit_handler);
```

```
    /* 処理を各スレーブに割り当てる */
    qt = N / Nslave;   /* 商 */
    rm = N % Nslave;   /* 余り */
    start = 1;
    for(i = 0; i < Nslave; i++){
        step = qt;
        if(i < rm)
            step++;   /* 端数の処理 */

        end = start + step - 1;          /* end をセット */

        pvm_initsend(PvmDataDefault);   /* 送信初期化 */
        pvm_pkint(&start, 1, 1);         /* start をパック */
        pvm_pkint(&end, 1, 1);           /* end をパック */
        pvm_send(Tids[i], 11);           /* スレーブに送信 */

        start += step;                   /* 次回の start をセット */
    }

    /* 全スレーブから結果を回収する */
    for(i = 0; i < Nslave; i++){
        pvm_recv(Tids[i], 21);       /* スレーブから受信 */
        pvm_upkint(&start, 1, 1);    /* start をアンパック */
        pvm_upkint(&end, 1, 1);      /* end をアンパック */
        pvm_upkint(&lsum, 1, 1);     /* lsum をアンパック */

        /* 部分和を表示 */
        printf("%5d - %5d: %8d   (TID:%x)¥n", start, end, lsum, Tids[i]);

        sum += lsum;   /* 部分和を合計 */
    }
    printf("        Total: %8d¥n", sum);   /* 計算結果を表示 */
    pvm_exit();   /* PVM から離脱 */
    return 0;     /* プログラム正常終了 */
}
```

(2)sum_slave_s.c 静的スレーブプログラム

```
#include <signal.h>
#include "pvm3.h"
```

```
/* シグナルの処理 */
void sigterm_handler(void)
{
    pvm_exit();   /* PVM から離脱 */
    exit(1);      /* プログラム異常終了 */
}

int main()
{
    int ptid;
    int i;
    int start, end;
    int lsum;

    /* シグナルの処理 */
    signal(SIGTERM, (void*)sigterm_handler);

    ptid = pvm_parent();   /* マスターのタスク ID を取得 */

    pvm_recv(ptid, 11);          /* マスターから受信 */
    pvm_upkint(&start, 1, 1);    /* start をアンパック */
    pvm_upkint(&end, 1, 1);      /* end をアンパック */

    /* start から end までの和を計算 */
    lsum = 0;
    for(i = start; i <= end; i++)
        lsum += i;

    pvm_initsend(PvmDataDefault);   /* 送信初期化 */
    pvm_pkint(&start, 1, 1);        /* start をパック */
    pvm_pkint(&end, 1, 1);          /* end をパック */
    pvm_pkint(&lsum, 1, 1);         /* lsum をパック */
    pvm_send(ptid, 21);             /* マスターに送信 */

    pvm_exit();   /* PVM から離脱 */
    return 0;     /* プログラム正常終了 */
}
```

(3)sum_master_d.c 動的マスタープログラム

```
#include <stdio.h>
#include <signal.h>
```

```
#include "pvm3.h"

#define SLAVE      4   /* 起動するスレーブ数 */
#define TDIV      20   /* 処理の分割数 */
#define N      10000   /* 1からこの数までの和を求める */

int Nslave;            /* 起動したスレーブ数 */
int Tids[SLAVE];       /* 各スレーブのタスク ID */

/* シグナルの処理 */
void exit_handler(void)
{
    int sn;

    /* 全スレーブを終了させる */
    for(sn = 0; sn < Nslave; sn++)
        pvm_kill(Tids[sn]);

    pvm_exit();   /* PVMから離脱 */
    exit(1);      /* プログラム異常終了 */
}

int main()
{
    int i;
    int tid, bufid;
    int start, step, end;
    int start2, end2;
    int qt, rm;
    int lsum, sum = 0;
    int minus1 = -1;

    /* スレーブ・プログラムを起動 */
    Nslave = pvm_spawn("sum_slave_d", (char**)0, 0, "", SLAVE, Tids);
    if(Nslave <= 0){
        fputs("PVM: Can't start slaves¥n", stderr);   /* エラーの表示 */
        return 1;                                     /* プログラム異常終了 */
    }
    /* シグナルの処理 */
    signal(SIGINT, (void*)exit_handler);
    signal(SIGTERM, (void*)exit_handler);
```

```
/* 各スレーブに最初の仕事を割り当てる */
qt = N / TDIV;   /* 商 */
rm = N % TDIV;   /* 余り */
start = 1;
for(i = 0; i < Nslave; i++){
    step = qt;
    if(i < rm)
        step++;   /* 端数の処理 */

    end = start + step - 1;          /* end をセット */

    pvm_initsend(PvmDataDefault);    /* 送信初期化 */
    pvm_pkint(&start, 1, 1);         /* start をパック */
    pvm_pkint(&end, 1, 1);           /* end をパック */
    pvm_send(Tids[i], 11);           /* スレーブに送信 */

    start += step;                   /* 次回の start をセット */
}

/* 仕事が終って結果を返してきたスレーブに次の仕事を割り当てる */
for(i = Nslave; i < TDIV + Nslave; i++){
    bufid = pvm_recv(-1, 21);     /* スレーブから受信 */
    pvm_upkint(&start2, 1, 1);    /* start2 をアンパック */
    pvm_upkint(&end2, 1, 1);      /* end2 をアンパック */
    pvm_upkint(&lsum, 1, 1);      /* lsum をアンパック */

    /* 結果を返したタスクの ID を調べる */
    pvm_bufinfo(bufid, (int*)0, (int*)0, &tid);

    /* 部分和を表示 */
    printf("%5d - %5d: %8d   (TID:%x)¥n", start2, end2, lsum, tid);

    sum += lsum;   /* 部分和を合計 */

    step = qt;
    if(i < rm)
        step++;   /* 端数の処理 */

    end = start + step - 1;              /* end をセット */
```

```
        if(start <= N){   /* 未処理の仕事がある？ */
            pvm_initsend(PvmDataDefault);   /* 送信初期化 */
            pvm_pkint(&start, 1, 1);        /* start をパック */
            pvm_pkint(&end, 1, 1);          /* end をパック */
            pvm_send(tid, 11);              /* スレーブに送信 */
        }

        start += step;                      /* 次回の start をセット */
    }

    /* 全スレーブに終了の合図を送る */
    pvm_initsend(PvmDataDefault);   /* 送信初期化 */
    pvm_pkint(&minus1, 1, 1);       /* minus1(-1)をパック */
    pvm_mcast(Tids, Nslave, 11);    /* 全スレーブにマルチキャスト */

    printf("        Total: %8d¥n", sum);   /* 計算結果を表示 */

    pvm_exit();   /* PVM から離脱 */
    return 0;     /* プログラム正常終了 */
}
```

(4)sum_slave_d.c 動的スレーブプログラム

```
#include <signal.h>
#include "pvm3.h"

/* シグナルの処理 */
void sigterm_handler(void)
{
    pvm_exit();   /* PVM から離脱 */
    exit(1);      /* プログラム異常終了 */
}

int main()
{
    int ptid;
    int i;
    int start, end;
    int lsum;
    /* シグナルの処理 */
    signal(SIGTERM, (void*)sigterm_handler);

    ptid = pvm_parent();   /* マスターのタスク ID を取得 */
```

```
    for(;;){
        pvm_recv(ptid, 11);          /* マスターから受信 */
        pvm_upkint(&start, 1, 1);    /* start をアンパック */
        if(start < 0)                /* すべての仕事が終了？ */
            break;
        pvm_upkint(&end, 1, 1);      /* end をアンパック */

        /* start から end までの和を計算 */
        lsum = 0;
        for(i = start; i <= end; i++)
            lsum += i;

        pvm_initsend(PvmDataDefault);   /* 送信初期化 */
        pvm_pkint(&start, 1, 1);        /* start をパック */
        pvm_pkint(&end, 1, 1);          /* end をパック */
        pvm_pkint(&lsum, 1, 1);         /* lsum をパック */
        pvm_send(ptid, 21);             /* マスターに送信 */
    }

    pvm_exit();   /* PVM から離脱 */
    return 0;     /* プログラム正常終了 */
}
```

(5)sum_master_s_array.c 配列版マスタープログラム

```
#include <stdio.h>
#include <signal.h>
#include "pvm3.h"

#define SLAVE      4   /* 起動するスレーブ数 */
#define N       1000   /* 配列の数 */

int Nslave;            /* 起動したスレーブ数 */
int Tids[SLAVE];       /* 各スレーブのタスク ID */
/* シグナルの処理 */
void exit_handler(void)
{
    int sn;

    /* 全スレーブを終了させる */
    for(sn = 0; sn < Nslave; sn++)
```

```
        pvm_kill(Tids[sn]);

    pvm_exit();   /* PVM から離脱 */
    exit(1);      /* プログラム異常終了 */
}

int main()
{
    int i;
    int start, step, end;
    int qt, rm;
    int lsum, sum = 0;
    int array[N];

    /* スレーブ・プログラムを起動 */
    Nslave = pvm_spawn("sum_slave_s_array", (char**)0, 0, "", SLAVE, Tids);
    if(Nslave <= 0){
        fputs("PVM: Can't start slaves¥n", stderr);   /* エラーの表示 */
        return 1;                                     /* プログラム異常終了 */
    }

    /* シグナルの処理 */
    signal(SIGINT, (void*)exit_handler);
    signal(SIGTERM, (void*)exit_handler);

    /* 配列の初期設定 */
    for(i = 0; i < N; i++)
       array[i] = i+1;

    /* 処理を各スレーブに割り当てる */
    qt = N / Nslave;   /* 商 */
    rm = N % Nslave;   /* 余り */
    start = 0;
    for(i = 0; i < Nslave; i++){
        step = qt;
        if(i < rm)
            step++;   /* 端数の処理 */

        end = start + step - 1;              /* end をセット */

        pvm_initsend(PvmDataDefault);        /* 送信初期化 */
```

```
        pvm_pkint(&start, 1, 1);              /* start をパック */
        pvm_pkint(&end, 1, 1);                /* end をパック */
        pvm_pkint(&array[start], step, 1);    /* 配列をパック */
        pvm_send(Tids[i], 11);                /* スレーブに送信 */

        start += step;                        /* 次回の start をセット */
    }

    /* 全スレーブから結果を回収する */
    for(i = 0; i < Nslave; i++){
        pvm_recv(Tids[i], 21);        /* スレーブから受信 */
        pvm_upkint(&start, 1, 1);     /* start をアンパック */
        pvm_upkint(&end, 1, 1);       /* end をアンパック */
        pvm_upkint(&lsum, 1, 1);      /* lsum をアンパック */

        /* 部分和を表示 */
        printf("%5d - %5d: %8d   (TID:%x)¥n", start, end, lsum, Tids[i]);

        sum += lsum;    /* 部分和を合計 */
    }

    printf("        Total: %8d¥n", sum);    /* 計算結果を表示 */

    pvm_exit();    /* PVM から離脱 */
    return 0;      /* プログラム正常終了 */
}
```

(6)sum_slave_s_array.c 配列版スレーブプログラム

```
#include <signal.h>
#include "pvm3.h"

/* シグナルの処理 */
void sigterm_handler(void)
{
    pvm_exit();    /* PVM から離脱 */
    exit(1);       /* プログラム異常終了 */
}

int main()
{
    int ptid;
    int i;
```

```
    int start, end;
    int lsum;
    int n;
    int *array;

    /* シグナルの処理 */
    signal(SIGTERM, (void*)sigterm_handler);

    ptid = pvm_parent();   /* マスターのタスク ID を取得 */

    pvm_recv(ptid, 11);          /* マスターから受信 */
    pvm_upkint(&start, 1, 1);    /* start をアンパック */
    pvm_upkint(&end, 1, 1);      /* end をアンパック */

    n = end - start + 1;
    array = (int *)malloc(sizeof(int)*n);

    pvm_upkint(array, n, 1);   /* 配列をアンパック */

    /* start から end までの配列の和を計算 */
    lsum = 0;
    for(i = 0; i < n; i++)
        lsum += *array++;

    pvm_initsend(PvmDataDefault);   /* 送信初期化 */
    pvm_pkint(&start, 1, 1);        /* start をパック */
    pvm_pkint(&end, 1, 1);          /* end をパック */
    pvm_pkint(&lsum, 1, 1);         /* lsum をパック */
    pvm_send(ptid, 21);             /* マスターに送信 */

    pvm_exit();   /* PVM から離脱 */
    return 0;     /* プログラム正常終了 */
}
```