PVAS

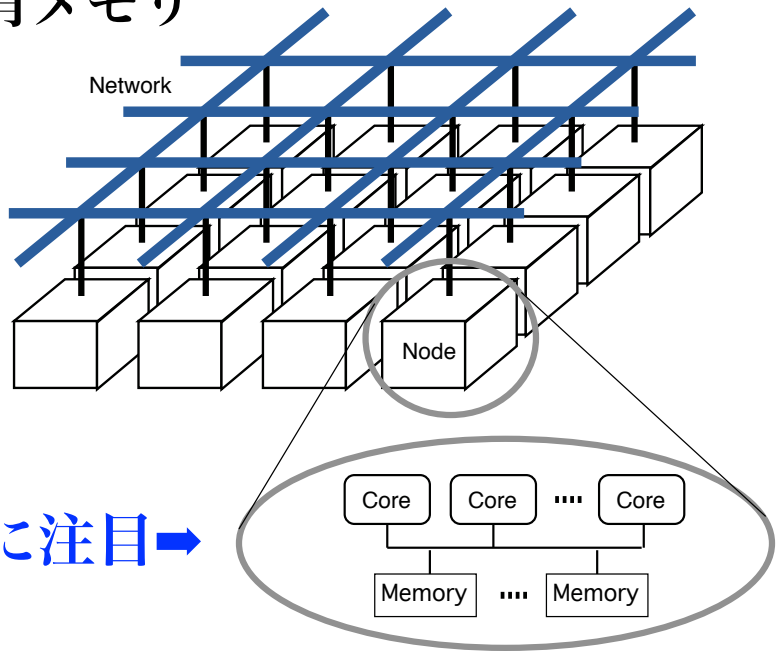
メニーコア向け並列実行モデル PVAS の紹介と Linux (x86_64/XeonPhi)でのPVASの使い方

AICSシステムソフトウェア研究室

堀　敦史

今時の並列計算機

- ノード間は分散メモリ
- ノード内は共有メモリ



ここに注目➡

ノード内並列

	Pros	Cons
プロセス並列	排他制御少ない	低速プロセス間通信
スレッド並列	高速スレッド間通信	排他制御多い

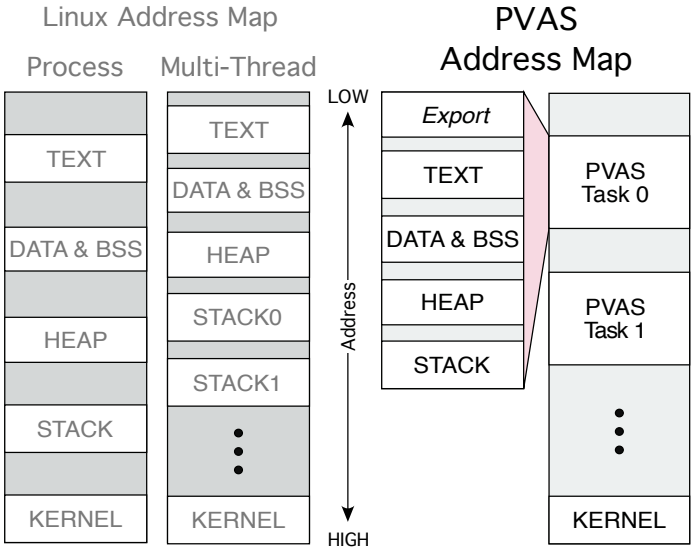
第3のノード内並列方式

	利点	欠点
プロセス並列	排他制御少ない	低速通信
スレッド並列	高速通信	排他制御多い
PVAS並列	高速通信 排他制御少ない	馴染みがない

PVAS: Partitioned Virtual Address Space

PVAS（ピーヴァス）とは？

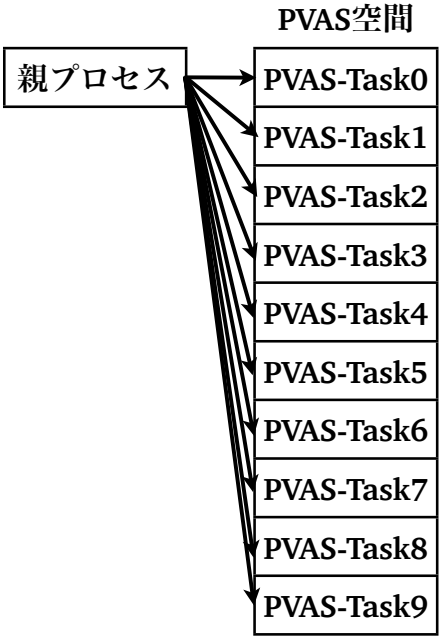
- 複数のプロセスを同じアドレス空間に配置
  - ➡ 変数は基本的にプロセス固有
  - ➡ 他のプロセスの情報に高速アクセス可能



PVASプログラミングに必要な最低限の知識

- PVAS空間
  - 複数のPVASタスクが共有するアドレス空間
  - PVDと呼ばれる識別子を持つ
- PVASタスク
  - プロセスとスレッドの中間的な性質
  - PVAS空間内でのみ動作
  - PVIDと呼ばれるユニークなIDを持つ
- Export領域
  - 他のPVASタスクとの情報交換するための領域
  - PVASタスクのひとつだけ持つことができる

PVAS-Hello



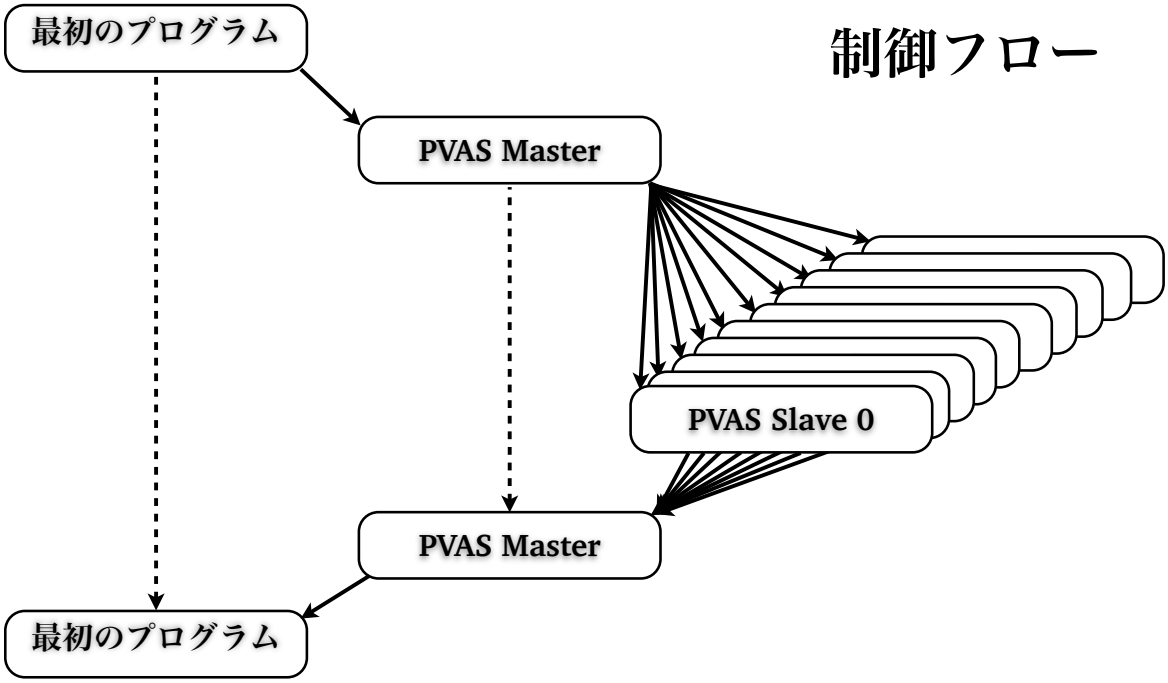

```
#include <pvas.h>

#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <errno.h>

#define N       (10)

int main( int argc, char **argv ) {
  extern char **environ;
  char *new_argv[3];
  pid_t pid;
  int pvd, pvid, i, err;

  if( argc < 2 ) {     /* not PVAS task */
    if( ( err = pvas_create( NULL, 0,
              &pvd ) ) != 0 ) {
      _exit( 9 );
    }
    new_argv[0] = argv[0];
    new_argv[1] = "dummy";
    new_argv[2] = NULL;

    for( i=0; i<N; i++ ) {
      pvid = i;
      err = pvas_spawn( pvd, &pvid,
          argv[0], new_argv,
          environ, &pid );
      if( err != 0 ) _exit( 9 );
    }
    do {
      /* wait for termination */
      wait( NULL );
    } while( errno != ECHILD );

    err = pvas_destroy( pvd );
    if( err != 0 ) _exit( 9 );
    /* done ! */

  } else {     /* PVAS task */
    if( pvas_get_pvid( &pvid ) != 0 ) {
      _exit( 9 );
    }
    printf( "Hello, this is PVAS task "
            "at %d (pid=%d)\n",
            pvid, getpid() );
  }
  return 0;
}
```


PVASタスク

- プロセスの親子関係は普通のプロセスと同じ
  - PVASタスクの終了を wait() で待てる
- pvas_spawn() で親プロセス（PVASタスク）のFDは継承される
  - printf() 等は普通に使える
- シグナルも普通のプロセスと同じ
- Coreバインドも可能（sched_setaffinity）

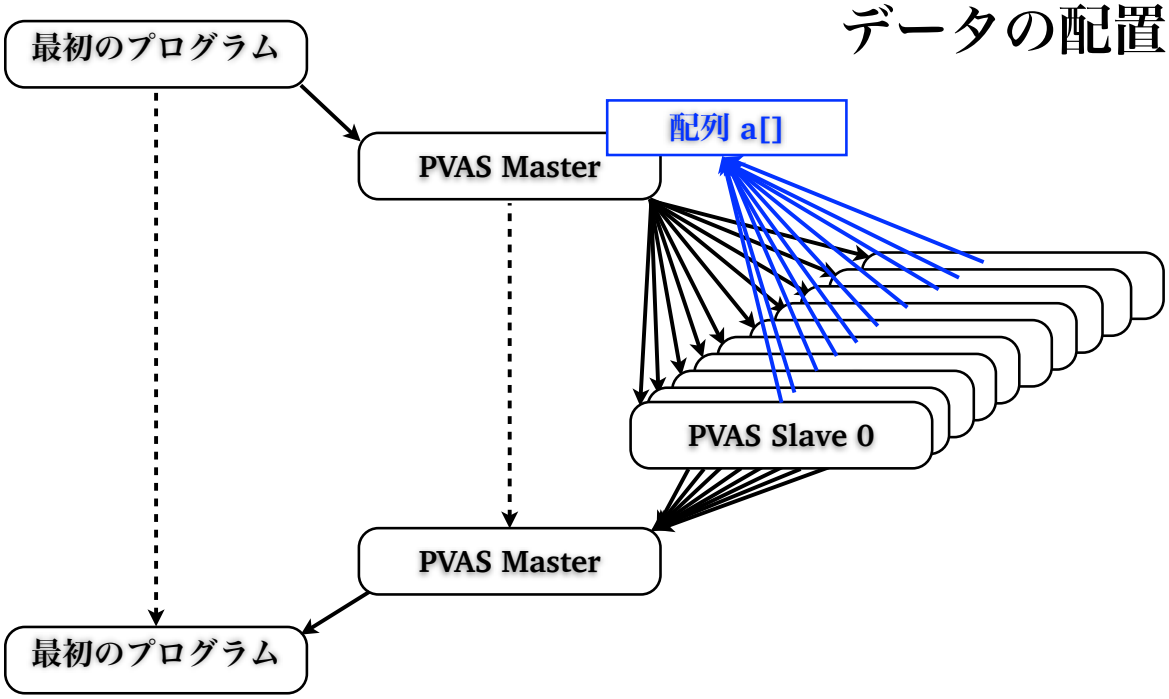
PVASを使った並列計算の例

制御フロー

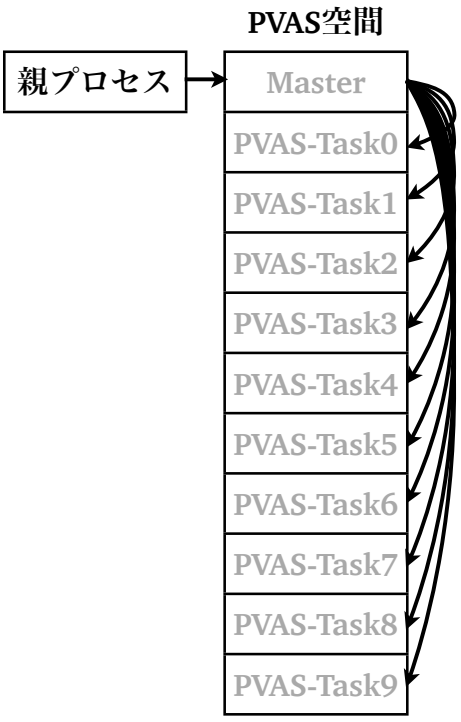


PVASを使った並列計算の例

データの配置



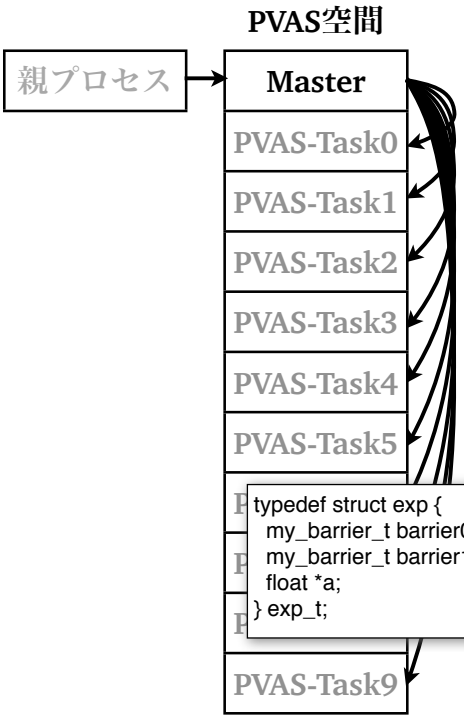
PVASを使った並列計算の例 (1/3)




```
#include <pvas.h>

#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>

#define N       (10)
#define M       (100000)

typedef struct exp {
  my_barrier_t barrier0;
  my_barrier_t barrier1;
  float *a;
} exp_t;
```

```
int main( int argc, char **argv ) {
  extern char **environ;
  pid_t  pid;
  int      pvd, pvid, err;

  pvid = -1;
  err = pvas_get_pvid( &pvid );
  if( err != 0 ) {     /* not PVAS task */
    if( ( err = pvas_create( NULL, 0,
                &pvd ) ) != 0 )
      _exit( 99 );
    pvid = 0;
    err = pvas_spawn( pvd, &pvid,
                argv[0], argv,
                environ, &pid );
    if( err != 0 ) _exit( 99 );
    do { /* wait for */
      wait( NULL );
    } while( errno != ECHILD );
    /* done ! */
```


PVASを使った並列計算の例 (2/3)

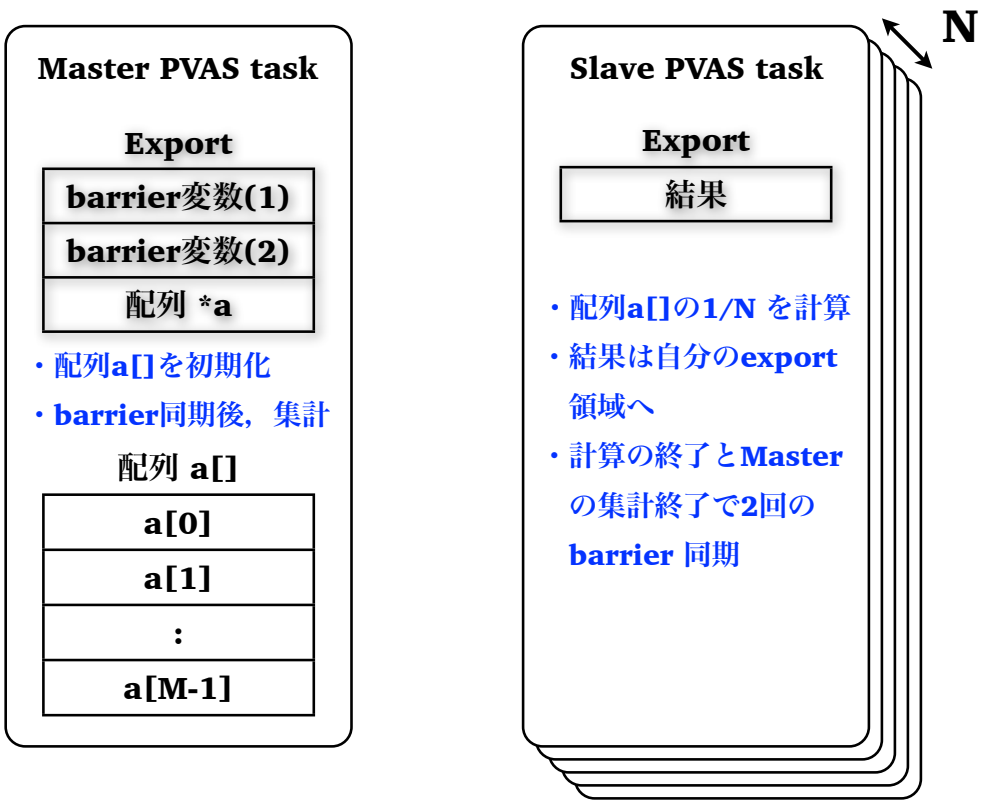



```
typedef struct exp {
  my_barrier_t barrier0;
  my_barrier_t barrier1;
  float *a;
} exp_t;
```

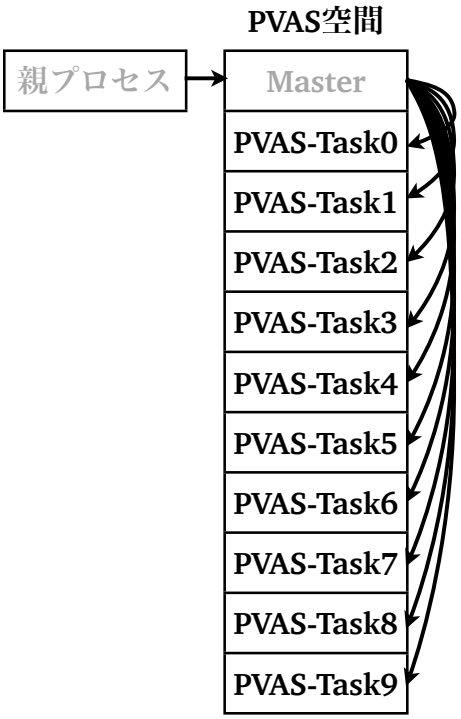
```
} else {              /* PVAS task */
  void  *export;
  exp_t *exp;
  float *a, sum;
  int   id, i;

  if( pvid == 0 ) { /* Master */

    a = (float*) malloc(
                  sizeof(float) * M );
    for( i=0; i<M; i++ )
        a[i] = ((float) i) / 1000.0;

    err = pvas_ealloc( sizeof(exp_t));
    if( err != 0 ) _exit( 99 );

    if( ( err = pvas_get_export_info(
                PVAS_MY_PVID,
                &export, NULL ) ) != 0 )
                _exit( 99 );
    exp = (exp_t*) export;
    exp->a = a;

    my_barrier_init( &exp->barrier0,
                        N+1 );
    my_barrier_init( &exp->barrier1,
                        N+1 );
```

```
    for( i=0; i<N; i++ ) {
      id = i + 1;
      err = pvas_spawn(
                PVAS_MY_SPACE,
                &id, argv[0], argv,
                environ, NULL);
      if( err != 0 ) _exit( 99 );
    }
    my_barrier_wait( &exp->barrier0 );
    sum = 0.0;
    for( i=0; i<N; i++ ) {
      /* get export region of myself */
      if( ( err = pvas_get_export_info(
            i+1, &export,
            NULL ) ) != 0 ) _exit( 99 );
      sum += *(float*)export;
    }
    my_barrier_wait( &exp->barrier1 );

    /* wait for termination */
    do {
        wait( &status ):
    } while( errno != ECHILD );
    /* all child tasks are done ! */

    printf( "sum = %f\n", sum );
```

Export 領域



PVASを使った並列計算の例 (3/3)




```
} else {      /* slave PVAS tasks */
  int i0, ie;

  /* get export region of master */
  if( ( err = pvas_get_export_info(
              0, &export,
              NULL ) ) != 0 ) {
    _exit( 99 );
  }
  exp = (exp_t*) export;
  a = exp->a;

  id = pvid - 1;
  i0 = ( M / N ) * id;
  ie = i0 + ( M / N );
  sum = 0.0;
  for( i=i0; i<ie; i++ ) sum += a[i];

  err = pvas_ealloc( sizeof( float );
  if( err != 0 ) _exit( 99 );

  /* get export region of myself */
  err = pvas_get_export_info(
        PVAS_MY_PVID,
        &export, NULL ) );
  if( err != 0 ) _exit( 99 );

  *(float*) export = sum;
  my_barrier_wait( &exp->barrier0 );
  /* here, the main task gathers */
  my_barrier_wait( &exp->barrier1 );
 }
}
return 0;
}
```

```
typedef struct exp {
  my_barrier_t barrier0;
  my_barrier_t barrier1;
  float *a;
} exp_t;
```

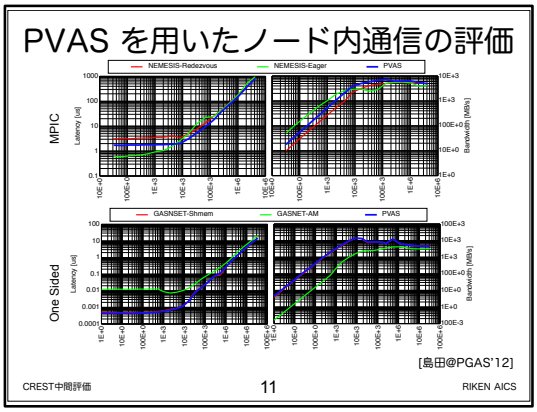
PVASプログラミングの注意事項

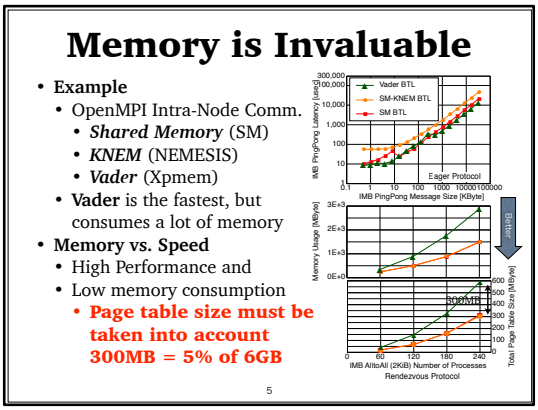
- PVASタスクから fork() はできない
  - 例えば system() は呼べない
  - 現在の実装上の都合
  - その代わりPVASタスクを “spawn” する
    - これは fork()+exec() のようなもの
- PVASタスクのコンパイル，リンクオプション
  - gcc -I/opt/pvas/include/ -L/opt/pvas/x86_64/lib -lpvas -lpthread pvas-hello.c -o pvas-hello -fpie -pie
  - -fpie オプションでコンパイル
  - -pie オプションでリンク
  - 静的リンクは使えない

PVASの応用

- 既に
  - MPIノード内通信の高速化（省メモリ化）
    - 連続領域の通信（1 Copy 通信）
    - 非連続領域の通信 MPI Datatype
  - XcalableMP の実装
- 今後
  - さらに省メモリな MPI 実装
- 誰か？
  - ツール（デバッガ，プロファイラなど）の開発

PVASの評価





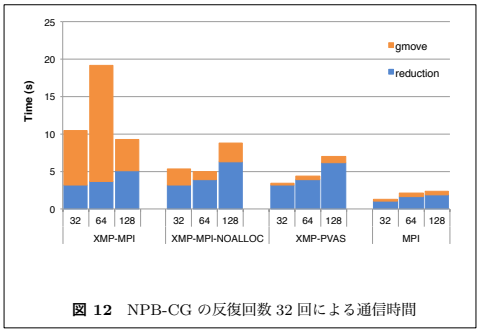


図 12 NPB-CG の反復回数 32 回による通信時間

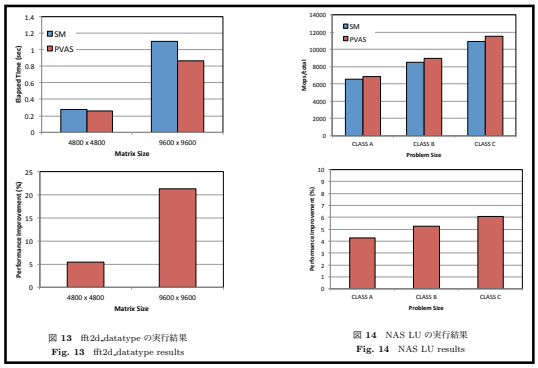


図 13 fft2d_datatype の実行結果
Fig. 13 fft2d_datatype results

図 14 NAS LU の実行結果
Fig. 14 NAS LU results

XMP (gmove) のPVAS実装，大川他（筑波大）
IPSJ-SIGHPC, 3月，2015

PVAS-RPMダウンロード

- x86_64 と Xeon Phi (KNC) で動作

http://www-sys-aics.riken.jp

ページの一番下

- VM operation
  - VM operation benchmark was evaluated by using hand-made micro benchmark.
  - In this benchmark, multiple tasks execute mmap, memset and munmap operations 1000 times over parallel degrees.
  - PVAS and multi-process implementation show good performance even if the number of tasks increases (Fig.4).

- Download pvas 1.1 rpm files (293MB)
- Download PVAS (678MB)
- Document
- Sample code

RPM インストール後

- Reboot（PVASパッチのLinux）

```
$ uname -a
Linux wallaby11.aics-sys.riken.jp 2.6.32-279.14.1.el6.pvas-1.1.x86_64 #1 SMP Wed
Feb 3 17:22:32 JST 2016 x86_64 x86_64 x86_64 GNU/Linux
```

- /opt/pvas 以下にインストールされる
- コンパイル&実行

```
$ gcc myprog.c -fpie -pie -o myprog -I/opt/pvas/include -L/opt/pvas/x86_64/lib -lpvas
-lpthread
$ export LD_LIBRARY_PATH=/opt/pvas/x86_64/lib:$LD_LIBRARY_PATH
$ ./myprog
```

- 是非使ってみたいという人は，ahori@riken.jp まで．PVASマシンのアカウント作ります．