メニーコアにおける新しいタスクモデル

PCクラスタワークショップ in 大阪2015

2月20日（金）

理化学研究所 計算科学研究機構

堀　敦史

プロセスとスレッド

- プロセス
  - それぞれ独立したメモリ空間
  - ノード内のプロセスは互いに保護されている
  - プロセスは（基本的に）独立した資源をもつことが出来る
- スレッド
  - 共有されたメモリ空間
  - スレッド間に保護はない（同じメモリ空間）
  - 資源は共有される（I/Oなども共有）

たとえ話 - 共同生活

- マンション
  - プライバシーは厳密に保たれる
    - 疎な人間関係
  - バス・トイレなどが戸別にある
    - いつでも好きな時に使える
- シェアハウス
  - プライバシーは（あまり）保護されていない
    - 密な人間関係
  - バス・トイレなどは共有
    - 一度に１人しか使えない

並列プログラム

- これまでのタスクモデルを使った並列プログラムの実現
  - マルチプロセス（MPI）- マンション
    - プロセス間通信が重い
    - 排他制御が必要な場面は多くない
  - マルチスレッド（OpenMP）- シェアハウス
    - スレッド間通信は軽い
    - 排他制御が必要な場面が多い

効率的な共同生活

- マンション／マルチスプロセス
  - プロセス間通信のオーバヘッドを下げたい
  - 壁に一時的に穴を開ける
    - マンション：時間とお金がかかる
    - マルチプロセス：穴を開けるオーバヘッド
- シェアハウス／マルチスレッド
  - 排他制御の頻度を下げたい
    - シェアハウス：専用バストイレを設ける！
    - マルチスレッド：専用メモリ空間を設ける

なぜプライバシーが必要か？

- 昔：コンピュータは高価であり、個人が占有するのにはもったいなかった
  - ユーザが独立したプログラムを走らせる
  - プライバシーが必要
- 今：個人がコンピュータを占有できるほど安くなった
  - それでもプライバシーは必要
  - 並列プログラムの場合でもそれを構成する個々のプロセスにプライバシーは必要か？
  - そもそもOpenMPにプライバシーは存在しない

なぜ資源の共有が問題になるのか？

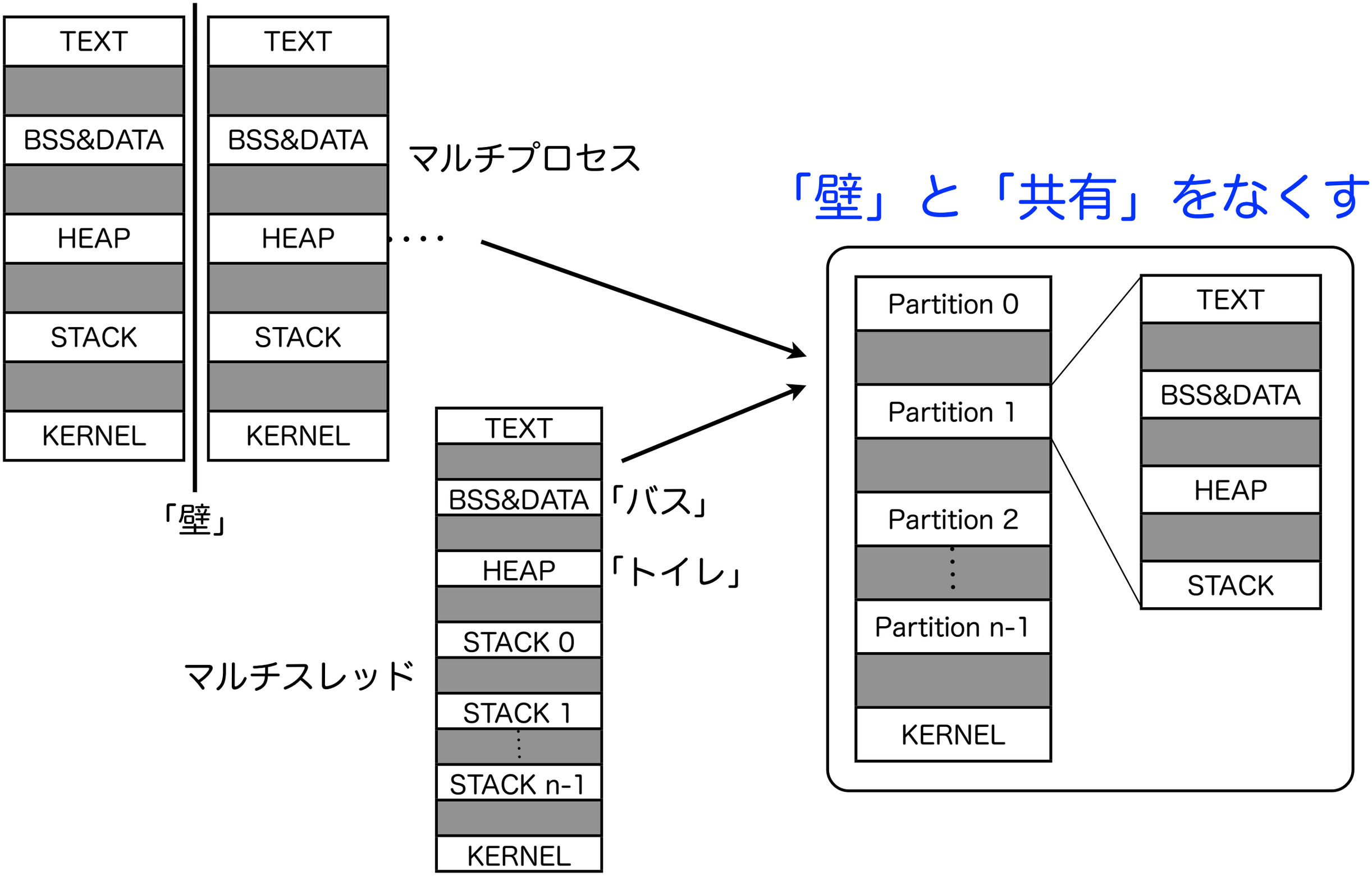
- スレッド固有資源の必要性
  - そもそも共有する必要がない
  - 排他制御を排除することで高速実行が可能
- マルチスレッド
  - 昔：待ち時間の有効活用
    - 排他制御の問題小
  - 今：複数CPUの有効活用
    - 排他制御のオーバヘッドが大きい

```
void foo(double a[][N], double b[]) {
   int i,j;
   double s;
#pragma omp parallel for private(i, s)
   for(i=0;i<N;i++){
      s=0;
      for(j=0;j<N;j++) s+=a[i][j];
         b[i]=s;
     }
}
```

もっと並列効率を良くしたい

- マルチプロセスから見た改善案
  - プロセスの「壁」を取り払う
    - 壁がなければ、穴を開ける時間とお金が不要
  - だけど、資源は共有しない
    - 個別にバス・トイレ（現状維持）
- マルチスレッドからみた改善案
  - 最初から全ての変数を “private” にし、共有したい時だけ共有できれば良い
    - 自分の部屋だけ private とするのではなく、最初から全ての部屋にバス・トイレ完備

プロセスとスレッドの中間

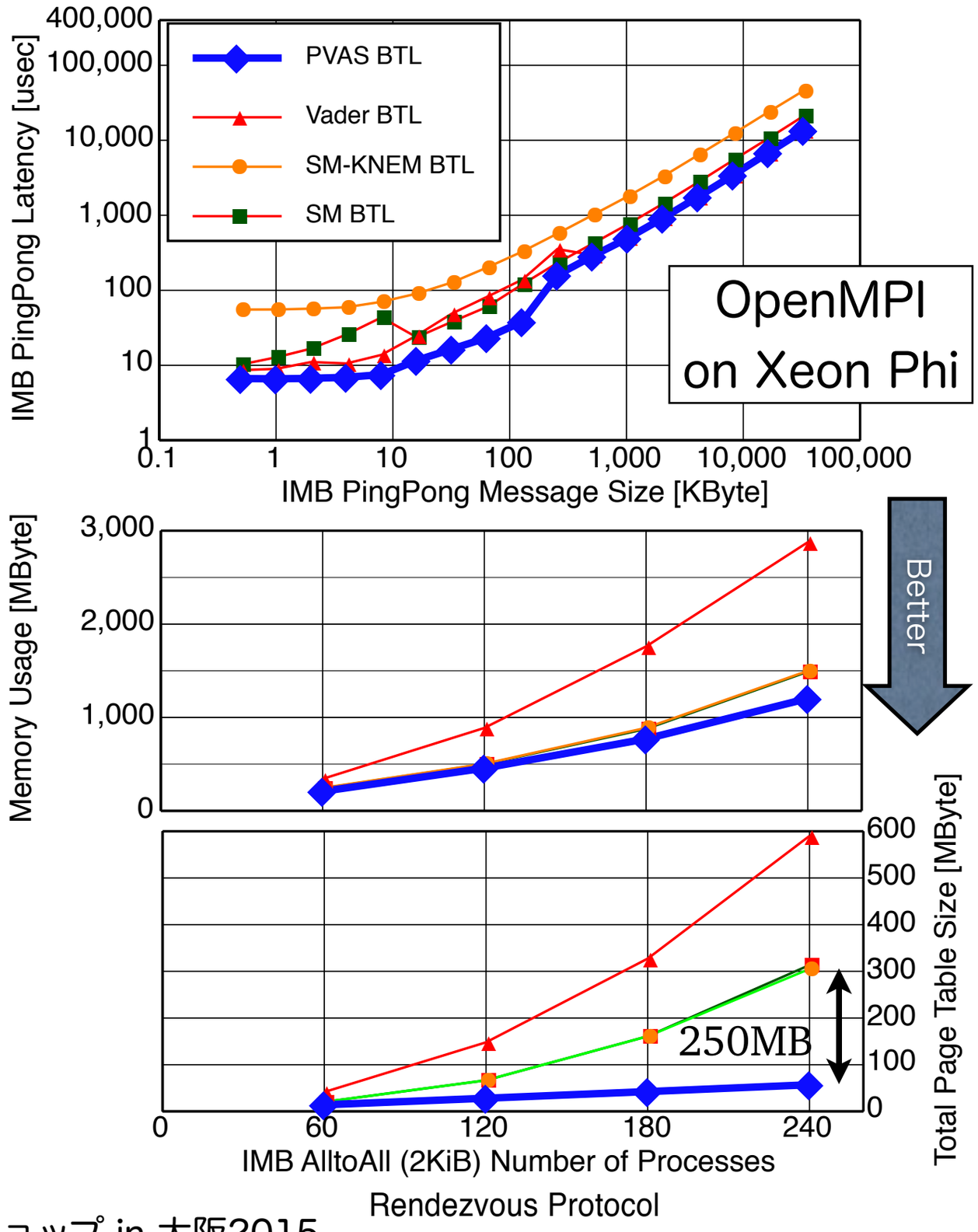


PVAS [ピーヴァス]

- Partitioned Virtual Address Space
  - プロセスとスレッドの中間的なもの
    - 実行の単位を「PVASタスク」と呼ぶ
  - 変数は全てPVASタスク固有
  - しかしながら、同じアドレス空間なので、共有したければ共有できる
    - 普通に LD/ST 命令でアクセスできるため高速
    - 共有部分のみ、必要ならば排他制御

PVASを用いたMPIの実装

- PVASを用いることで
  ノード内通信の効率化
  - 高速、かつ
  - 省メモリ
- なぜ？
  - 1copyで通信可能
  - ページテーブルが小さくて済む
- CPUコア数が大きい程有利
  - メニーコア



PVAS-ULP

- MPIにおける通信遅延の隠蔽
  - ノンブロッキング通信
    - 人手による実行順序の入れ替え
      - プログラミングの手間がかかる
      - プログラムが見難くなる
      - 静的なので環境毎にチューニング必要
- User-Level Process
  - ユーザレベルでプロセス切替
    - 高速なプロセス切替（PVASだから実現可能）
  - 通信待ちが発生したら、他のULPに制御を移す
    - 特別なプログラミング不要
    - プログラムは見易い
    - ランタイムが制御するので、動的

米アルゴンヌ研究所との共同研究

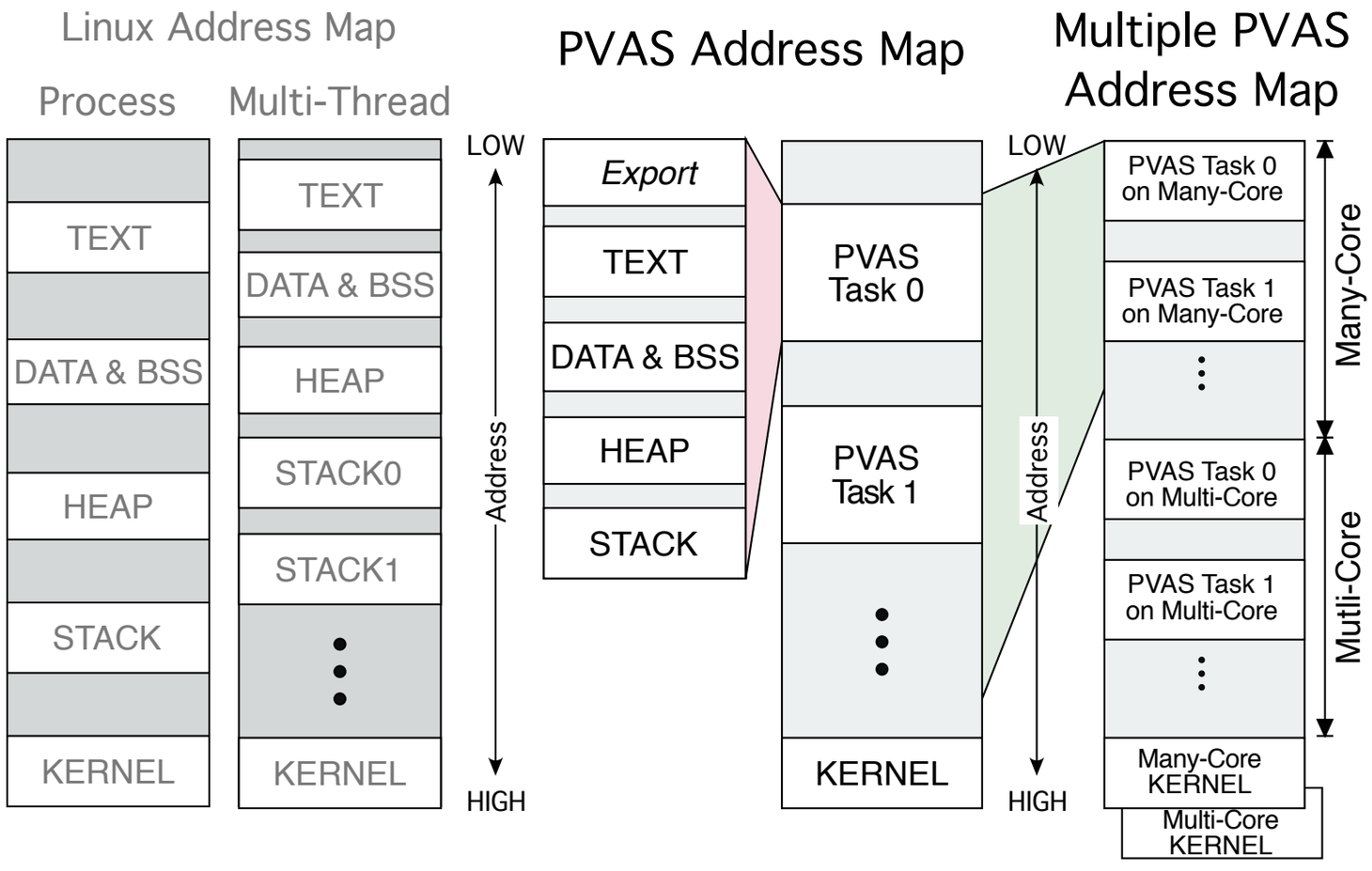
その他の共同研究

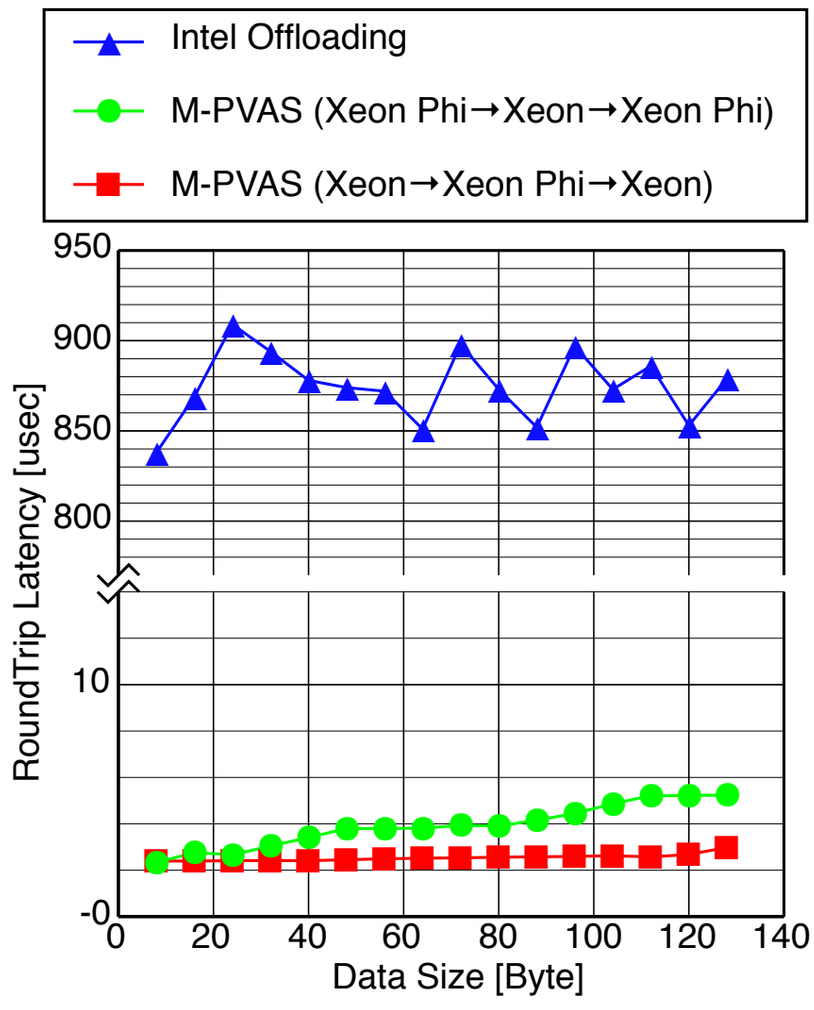
- 筑波大（佐藤先生との共同研究）
  - PVAS を用いて XcalableMP を実装する
  - 情報処理学会HPC研究会（3月）で発表予定

- 米テネシー大学
  - OpenMPI の PVAS を用いた最適化
    - 集団通信の高速化
    - 省メモリ実装

Multiple-PVAS

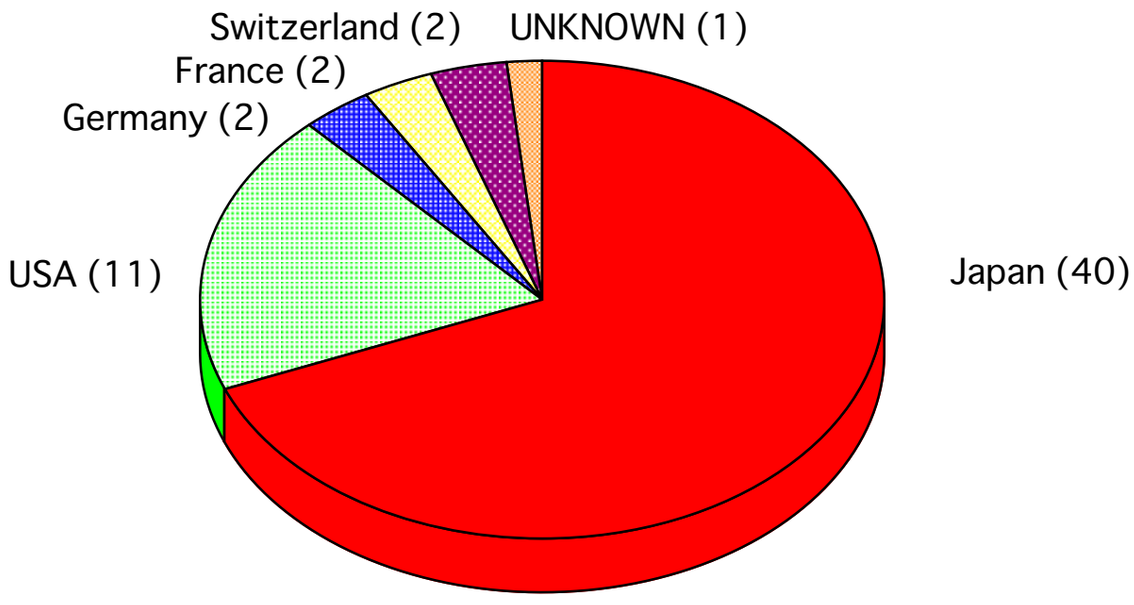
- メニーコアとマルチコアが混在する計算ノード
  - PVAS のアイデアを拡張





まとめ

- 新しい並列実行のためのタスクモデルの提案
  - PVAS
    - 従来のプロセスとスレッドの中間
    - 特にメニーコアで有効
- ソフトウェアの公開（2013年11月～）



http://aics-sys.riken.jp/releasedsoftware/pvas.html