Distributed Aggregate with Migration

マイグレーションを支援する分散集合オブジェクト

30388 高橋 慧
近山･田浦研究室

powered by Phoenix Library

グリッド環境での計算

- インターネット上の計算機で並列処理
- 動的なプロセッサ数の変化に対応させたい
  - ►使用可能なプロセッサが頻繁に変化 (故障など)
- プロセッサ数増減に対応するプログラムは
  記述が難しい または 遅い
- 本研究の提案：「分散集合オブジェクト」
  - ►動的プロセッサ数の増減に対応
  - ►記述が容易 (オブジェクト指向)
  - ►高速 (無駄な通信が発生しない)

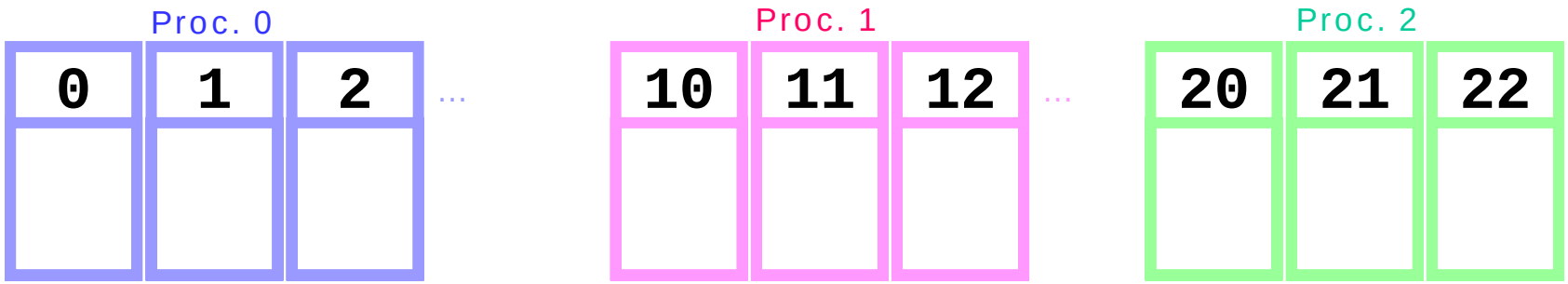
発表の流れ

- はじめに
- 背景
- 提案
- 実装
- 実験
- まとめ

ここに現在位置が出ます

分散集合 (Distributed Aggregate)

- ▶インデックスで位置を識別できるデータ
- ▶断片が複数のプロセッサに存在
- ▶分散配列･分散ハッシュ表など
- ▶プロセッサ数が固定だと、インデックスから
  プロセッサ番号を簡単に計算可能



例えば…要素が各プロセッサ10個ずつの場合、12番のデータを持つのは
12 / 10 = 1 より1番のプロセッサと簡単にわかる

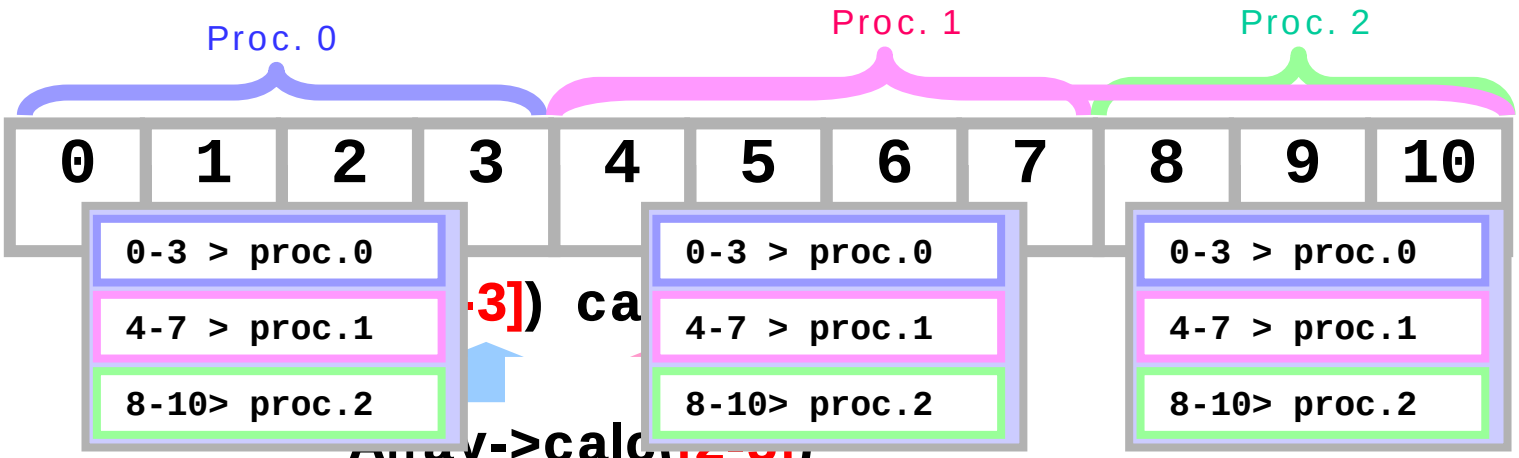
分散集合の記述

- プロセッサ数が変化すると…
  - ▶どの要素がどのプロセッサにあるか不明
  - ▶要素とプロセッサの対応表が必要
- 対応表を一プロセッサが集中管理
  - ▶記述は簡単
  - ▶対応表を持つプロセッサにメッセージが集中
- 対応表を全プロセッサが分散保持
  - ▶性能は良い
  - ▶記述が大変

```
 0, 1,.. 9 > proc.0
10,11,..19 > proc.1
20,21,..29 > proc.2
```

分散集合オブジェクトモデル

- 分散集合をオブジェクト指向で記述
- メソッド呼び出し:
  「集合全体」に対しインデックス範囲を指定
- マイグレーションによりプロセッサ増減に対応
- (実装)要素とプロセッサの対応を分散保持
  ⇒メッセージが一つのプロセッサに集中しない



メソッド呼び出し

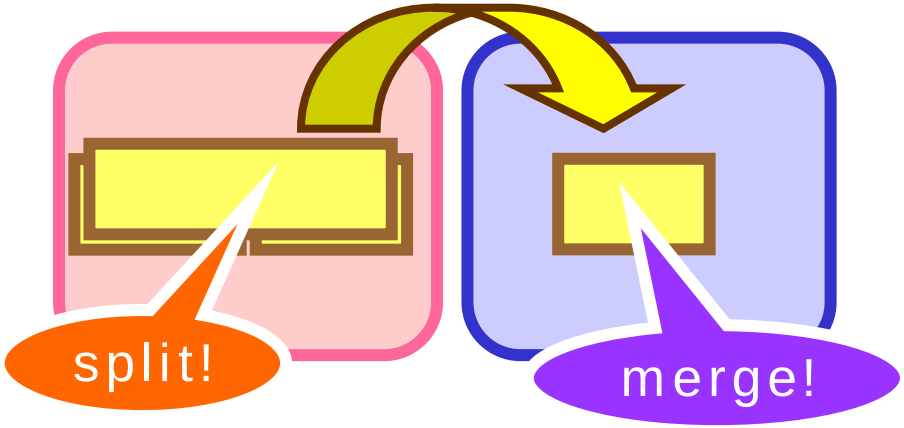
分散配列の要素をインクリメント

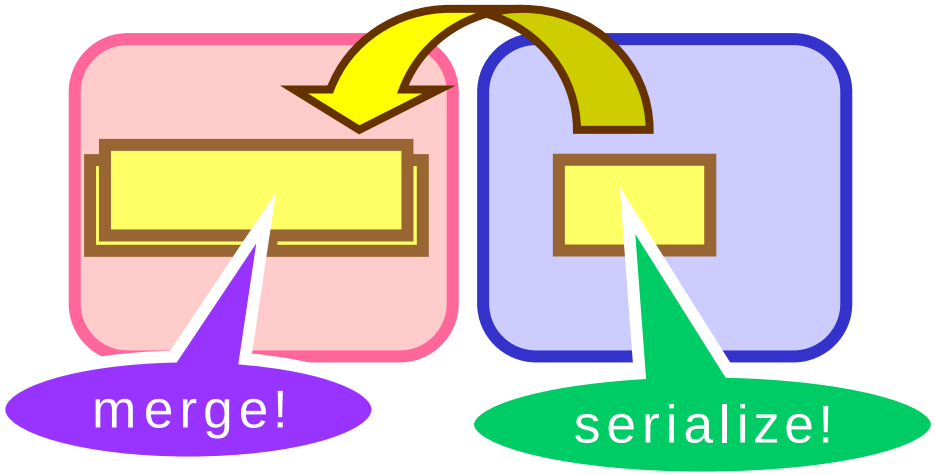
►配分によらず要素[100-200)をインクリメントできる

```
void DistributedArray::increment(IndexSet *is){
  foreach (index <- is){
    data[index]++;
  }
}

int main(){
…
  WholeArray->increment ([100-200));
```

マイグレーション

- プロセッサ増加
  - ▶new
  - ▶分割 (split)
  - ▶合併 (merge)
- プロセッサ減少
  - ▶シリアライズ
  - ▶合併 (merge)
  - ▶delete





マイグレーション

計算に参加し、断片を受け取る擬似コード

- ►オブジェクト定義で、split()とmerge()を記述
- ►プログラマはobject_idを指定してjoin()を呼び出し

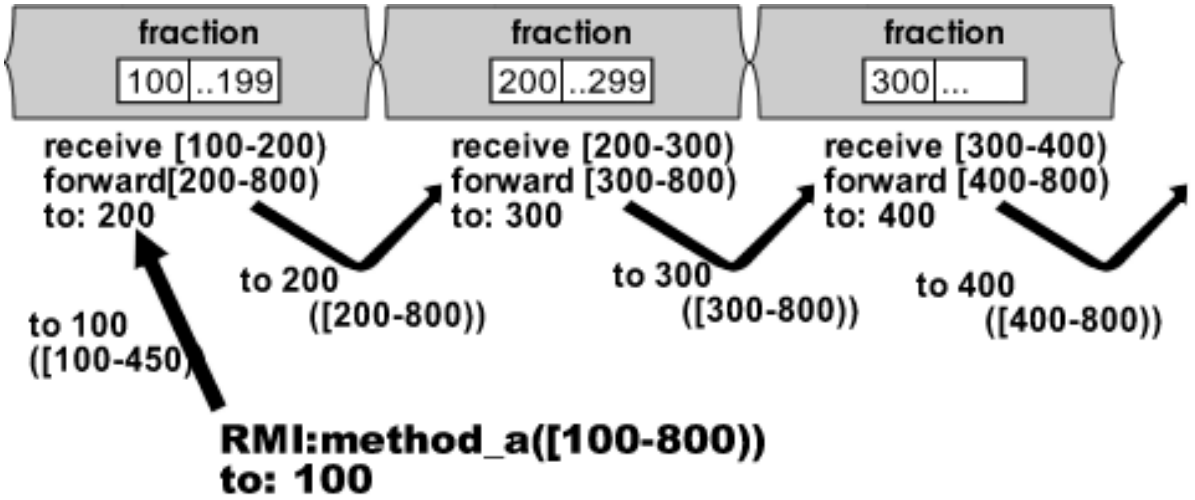
```
DistributedArray::split(void)
{(データとインデックス集合を分割)
  return (分割したデータ);}
DistributedArray::serialize()
{ return (保持するデータ);}
DistributedArray::merge(データ)
{(データをデシリアライズして併合)}

int main(){
  join(object_id);
```

システムの実装

- C++で、Phoenixライブラリを用いて実装
  - ►IndexSet (一次元/二次元の二種類を提供)
  - ►シリアライズ用ライブラリ
  - ►RMI / マイグレーション
  - ►約3000行

ルーティングとリレー

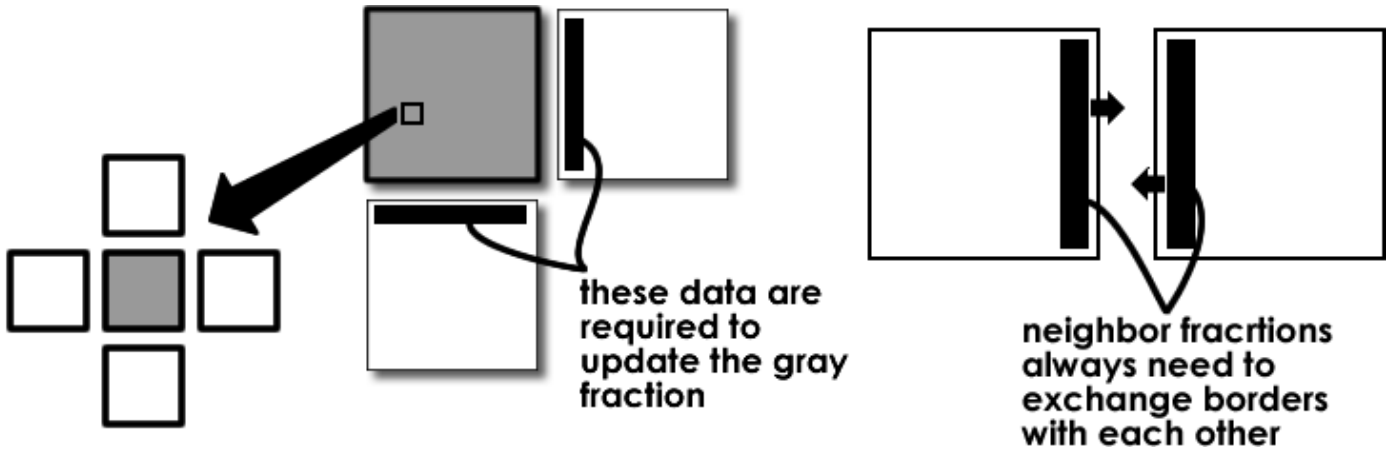
- Phoenixライブラリの概要
  - ▶各プロセッサがたくさんの番号を持つ (0-100など)
  - ▶番号を指定してメッセージを送信　(send to 5など)
  - ▶整数とプロセッサの対応を分散保持
- インデックス一つをPhoenixの番号に対応
  - ▶集合演算でRMI要求を中継



アプリケーションの記述

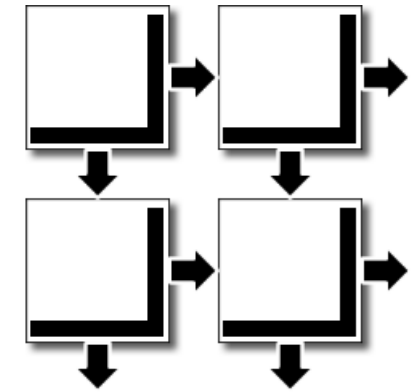
偏微分方程式の数値解法

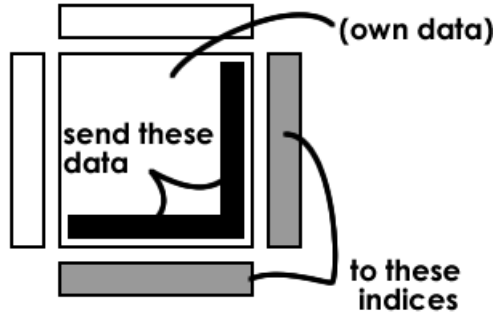
- ►二次元配列を分散保持
- ►以下をループ
  - 端のデータを交換
  - 値を更新
  - if(残差 < 閾値) 終了 else ループ



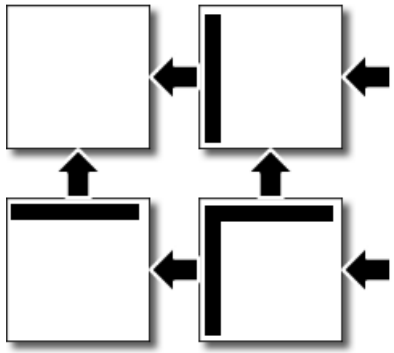
端のデータの交換

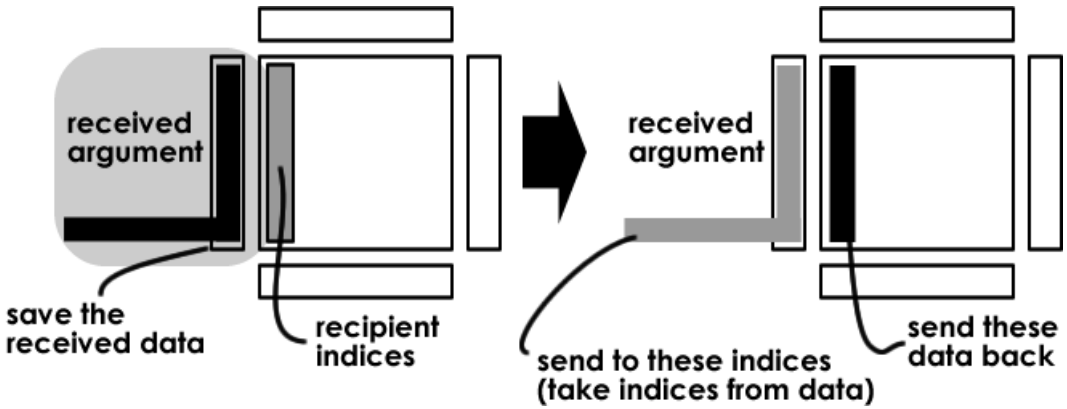
- send_border()
  - ►宛先は右・下隣のIndex集合
  - ►データは右端・下端のデータ





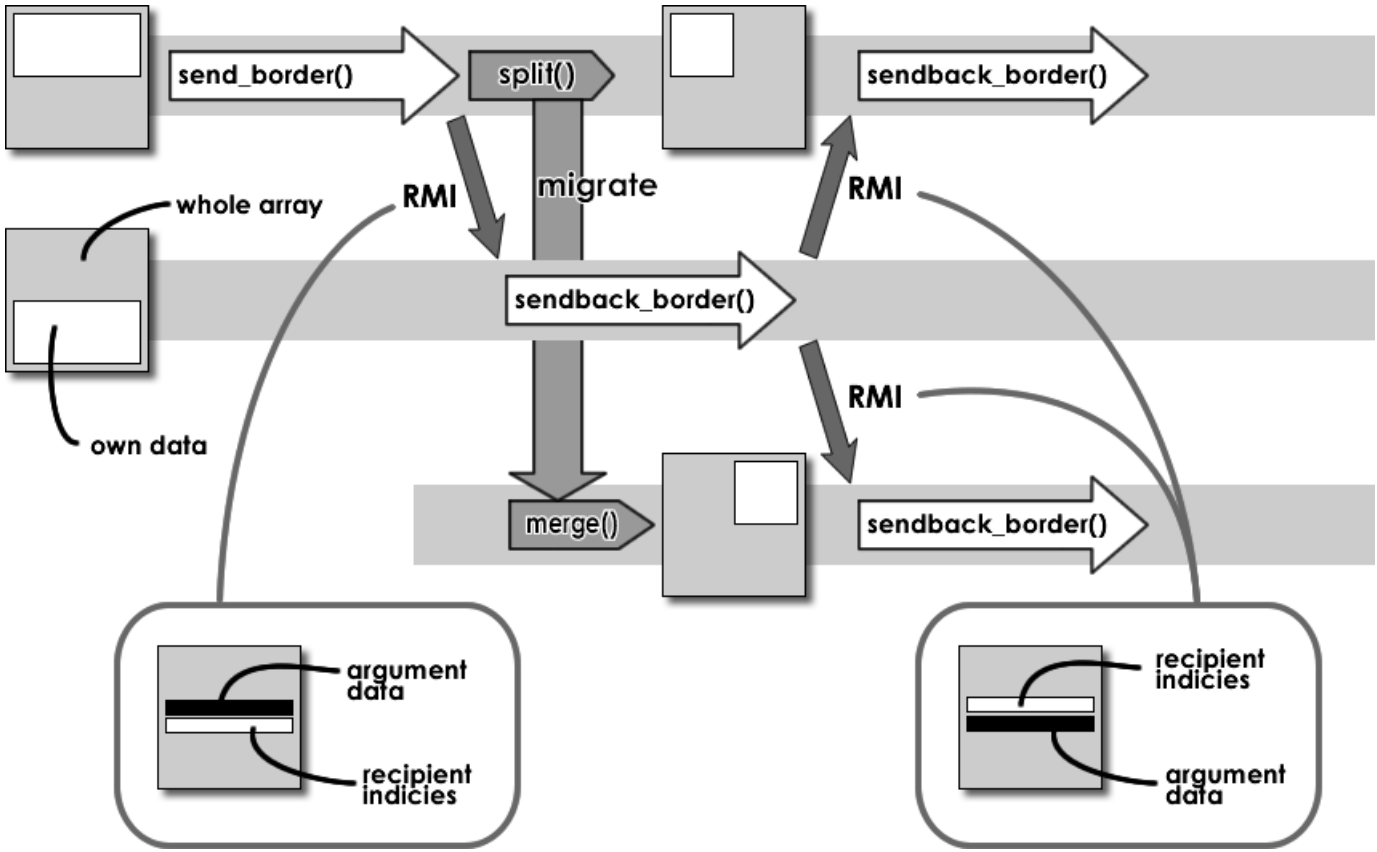
- sendback_border()
  - ►宛先は「送られてきたデータのIndex集合」
  - ►データは「宛先のIndex集合」





マイグレーションの様子

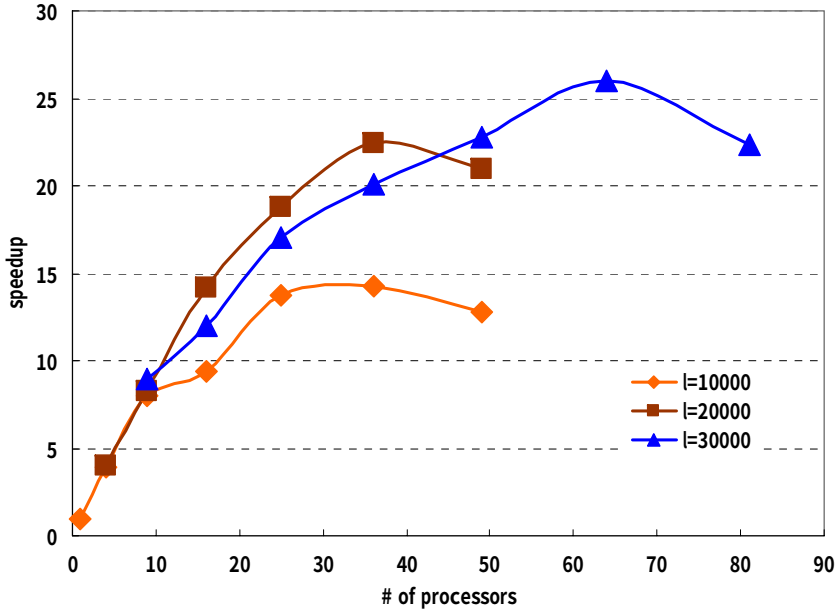
端の交換中も集合の分割が可能



実験

実行結果

- プロセッサ数の増減に対応
  - ▶64台での連続joinに成功
- Mflops値
  - ▶一回のループ時間から計算
  - ▶一辺10000, 20000, 30000で実行
- 台数増加により頭打ち
  - ▶残差総計のアルゴリズムが原因と推測



まとめ

- 分散集合オブジェクトの提案・実装を行った
  - ▶Index範囲を指定したメソッド呼び出し
  - ▶断片のマイグレーションに対応
  - ▶メソッド呼び出し時にメッセージが集中しない
- 容易な記述でプロセッサ増減に対応
  - ▶偏微分方程式の解法
- 今後の課題
  - ▶記述性の改善 (返り値のサポートなど)

Fin.