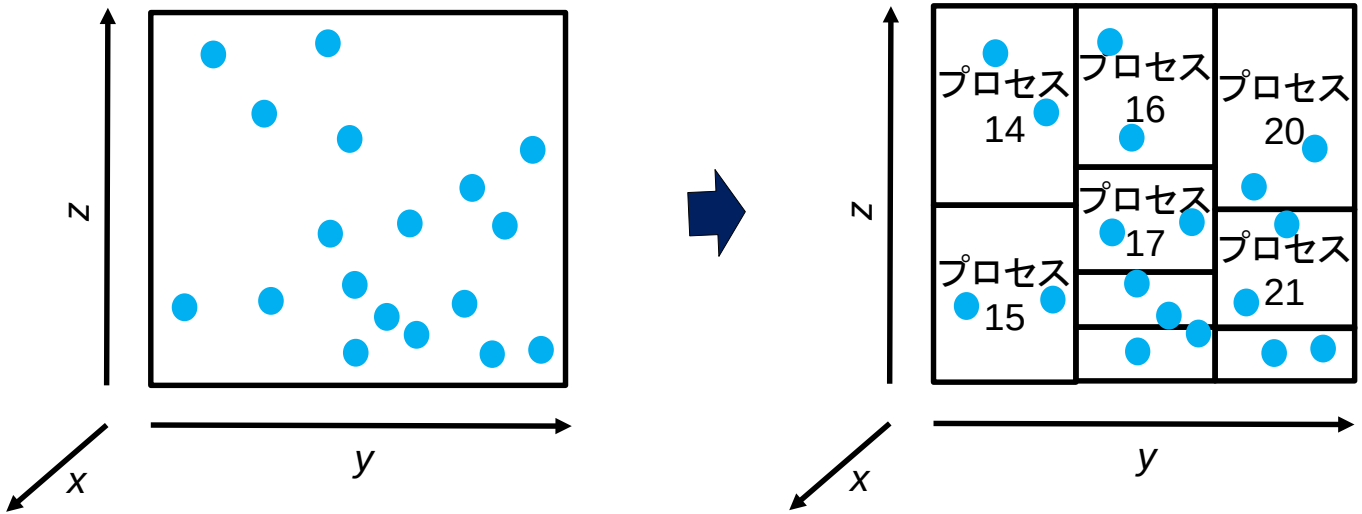
粒子系シミュレーションにおける高効率な粒子データ通信

薄田 竜太郎
(九州大学 情報基盤研究開発センター)

領域分割

■ 粒子系シミュレーションのプロセス並列方式

各プロセスの計算時間をできるだけ等しくする



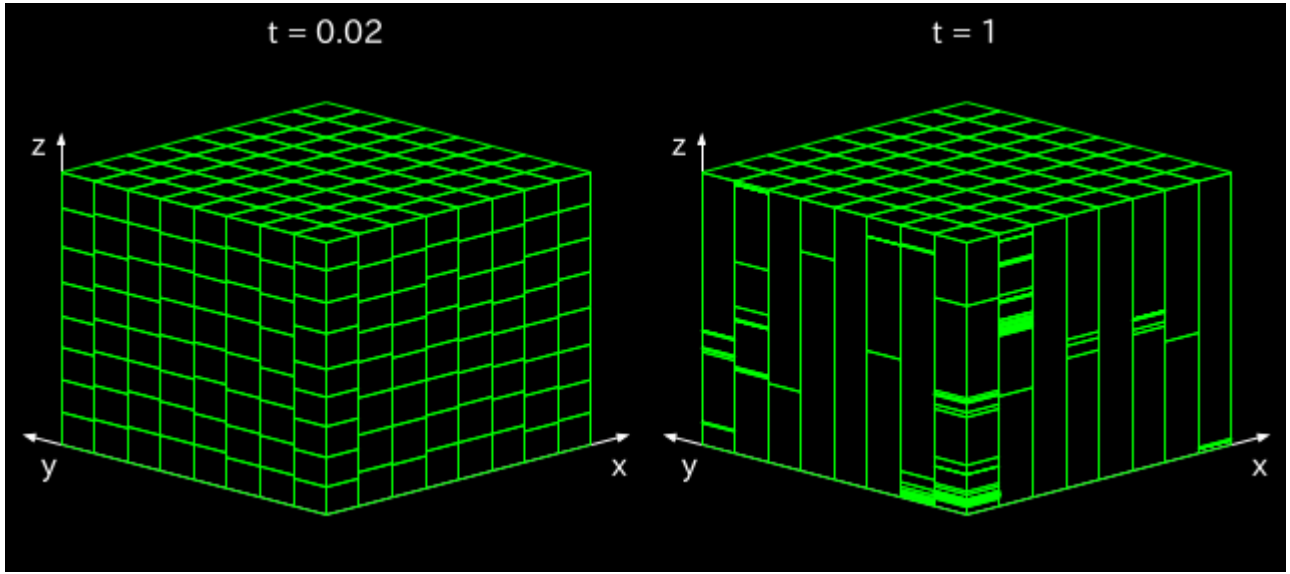
領域分割の例

■ シミュレーション開始時

一様に近い粒子分布

■ シミュレーション終了時

クラスタのある粒子分布



ほぼ同じ体積に分割

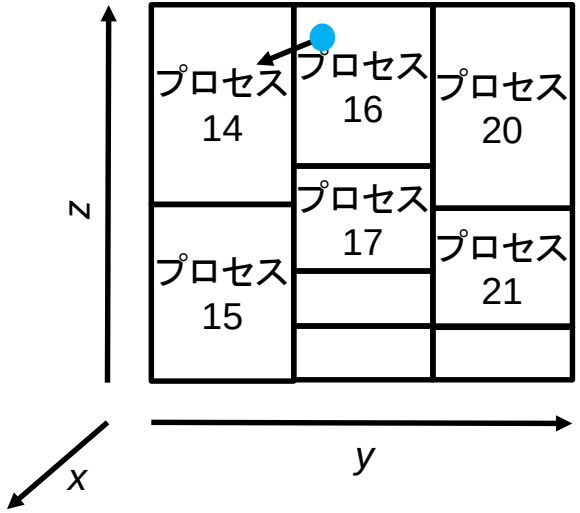
異なる体積に分割

512プロセス 重力*N*体シミュレーション

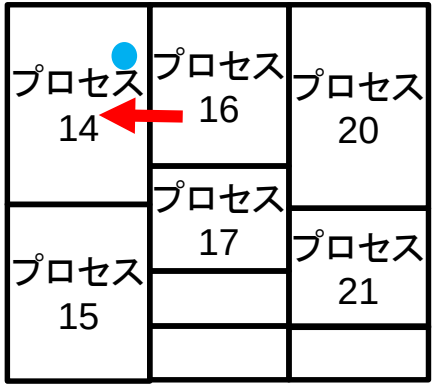
粒子データ通信 1

■ 粒子の移動

シミュレーション時刻 = t_1



シミュレーション時刻 = t_2



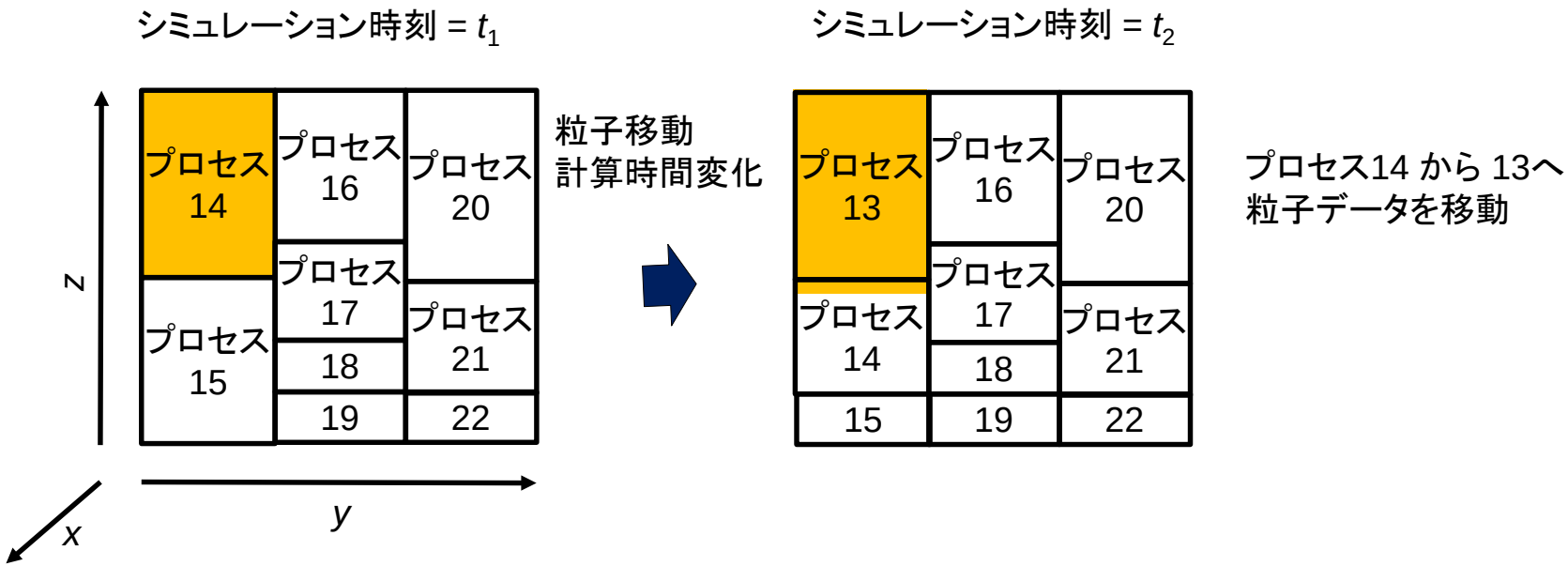
プロセス16から14へ
粒子データを移動

プロセス16は14へ送信するとわかる
プロセス14は16から受信するかわからない

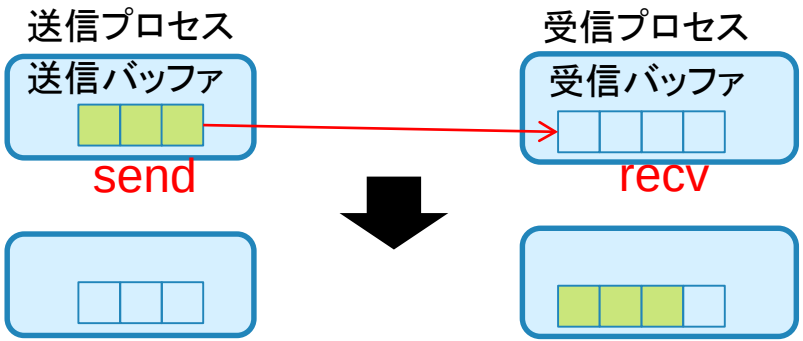
受信プロセスはいくつのプロセスが送信してくるかわからない

粒子データ通信 2

■ 領域分割



send-recv型 通信



受信プロセスはいくつのプロセスが送信してくるか
知っている必要がある

MPIはsend-recv型 通信を基盤としている

粒子データ通信では受信プロセスはいくつのプロセスが送信してくるかわからない

粒子データ通信を簡単かつ効率的に扱うことは困難

粒子データ通信のアルゴリズム

類似した通信パターンは離散事象シミュレーション、グラフ計算、レベルセット法などにも見られる

複数の通信を組み合わせる

■ プロセス数に対するスケーラビリティ

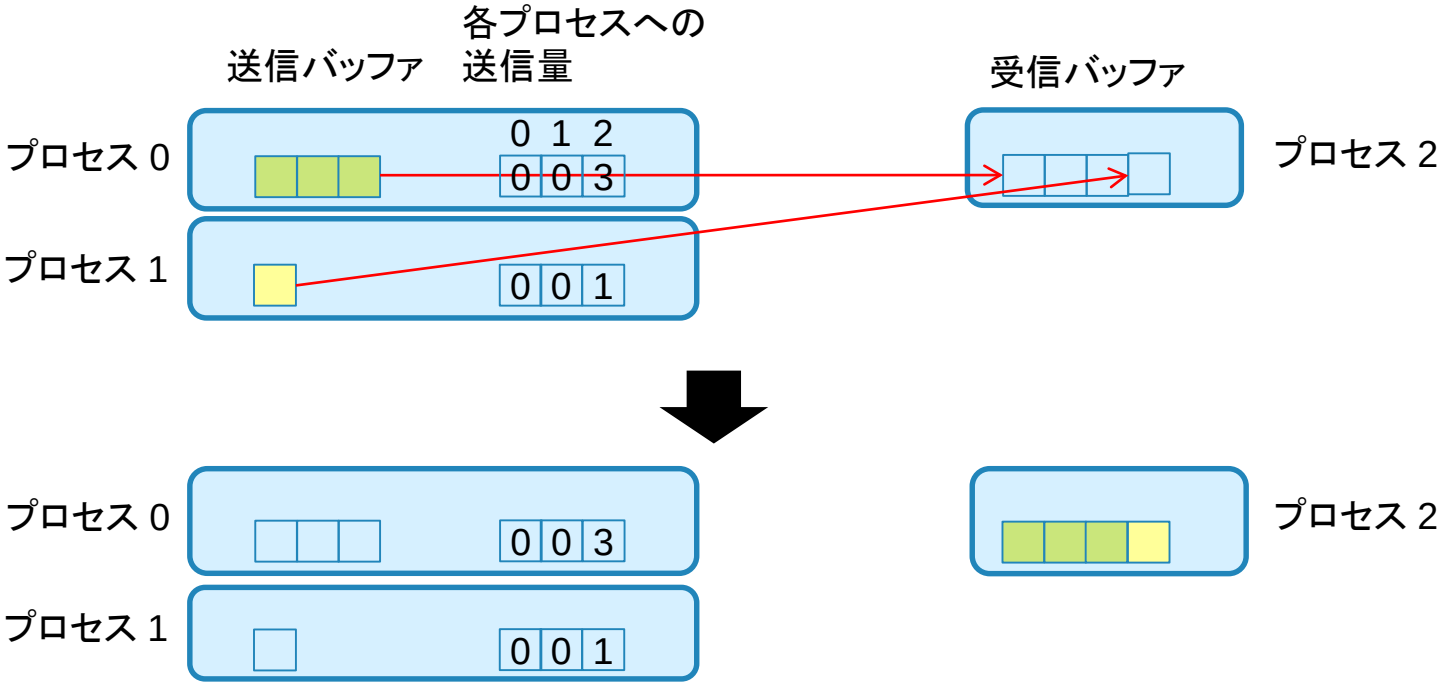
アルゴリズム	Heofler et al. (2010) マイクロベンチマーク+ 性能モデル MPI + 独自ライブラリ	Gerstenberger et al. (2014) マイクロベンチマーク + 性能モデル MPI + 独自ライブラリ
all-to-all×2		
reduce-scatter + send-recv		
片側通信 + send-recv		◎
send-recv + non-blocking barrier	◎	◎

実際の通信時間による性能比較

- 6種類の実装

実装	通信ライブラリ
all-to-all × 2	MPI
send-recv + reduce	
send-recv + non-blocking barrier	
reduce-scatter + send-recv	
片側通信	
片側通信	ACP

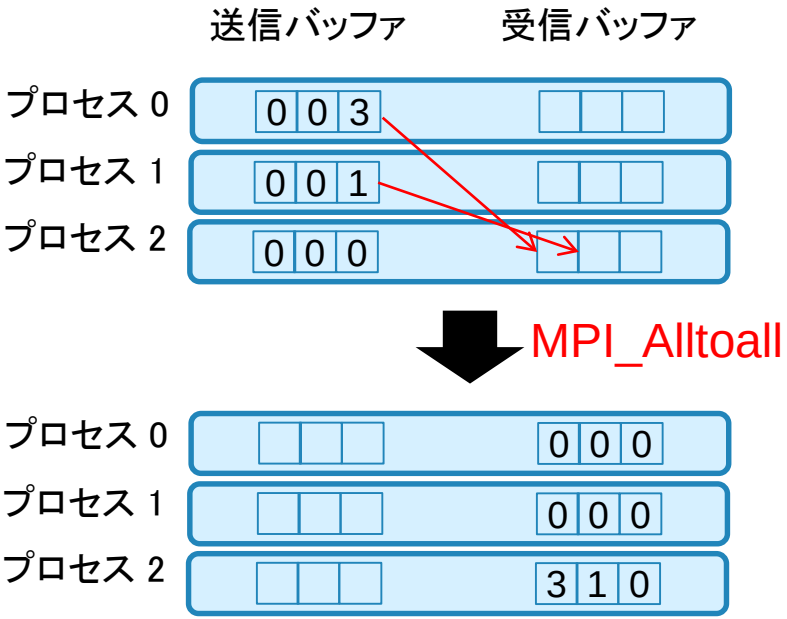
通信の例



プロセス0, 1は受信プロセスと送信量を知っている

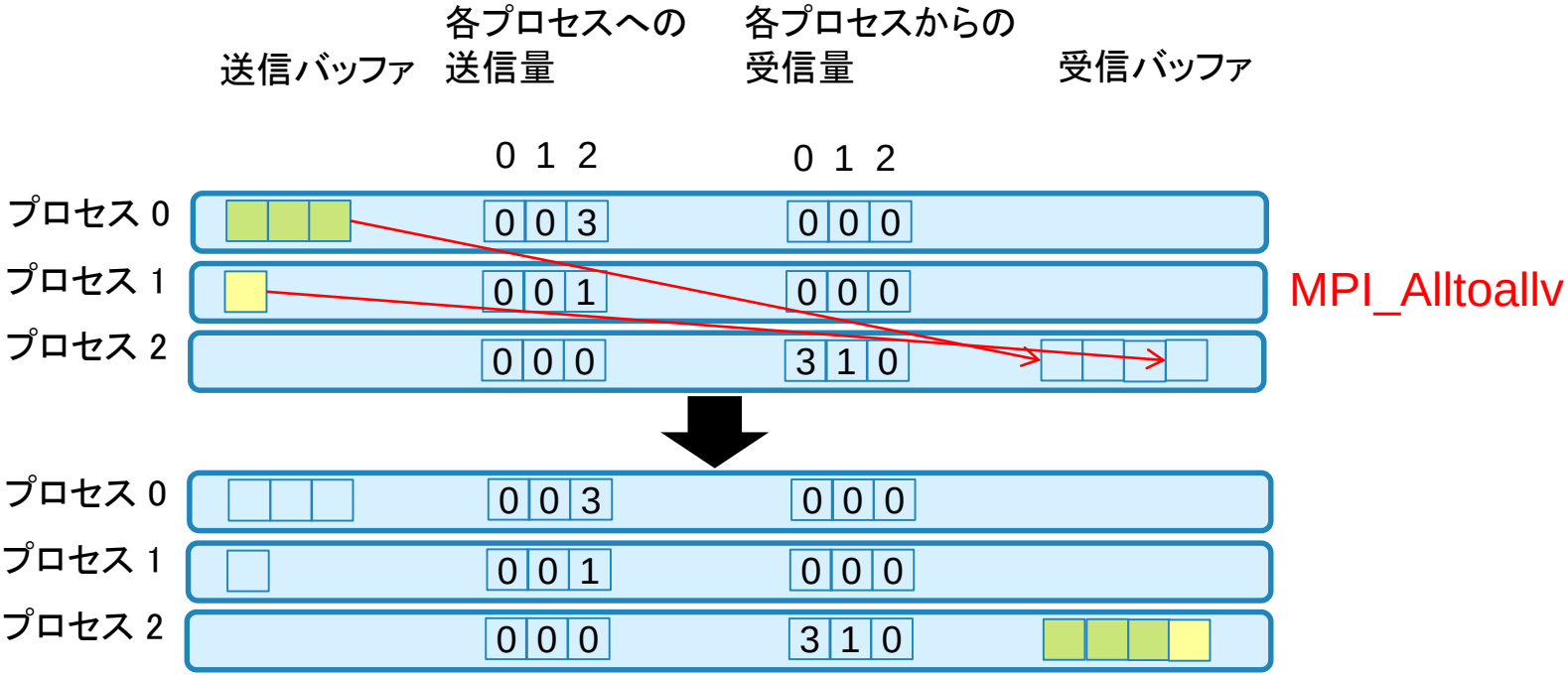
プロセス 2は送信プロセスと受信量を知らない

MPI_Alltoall



プロセス間の通信量がすべて同じ

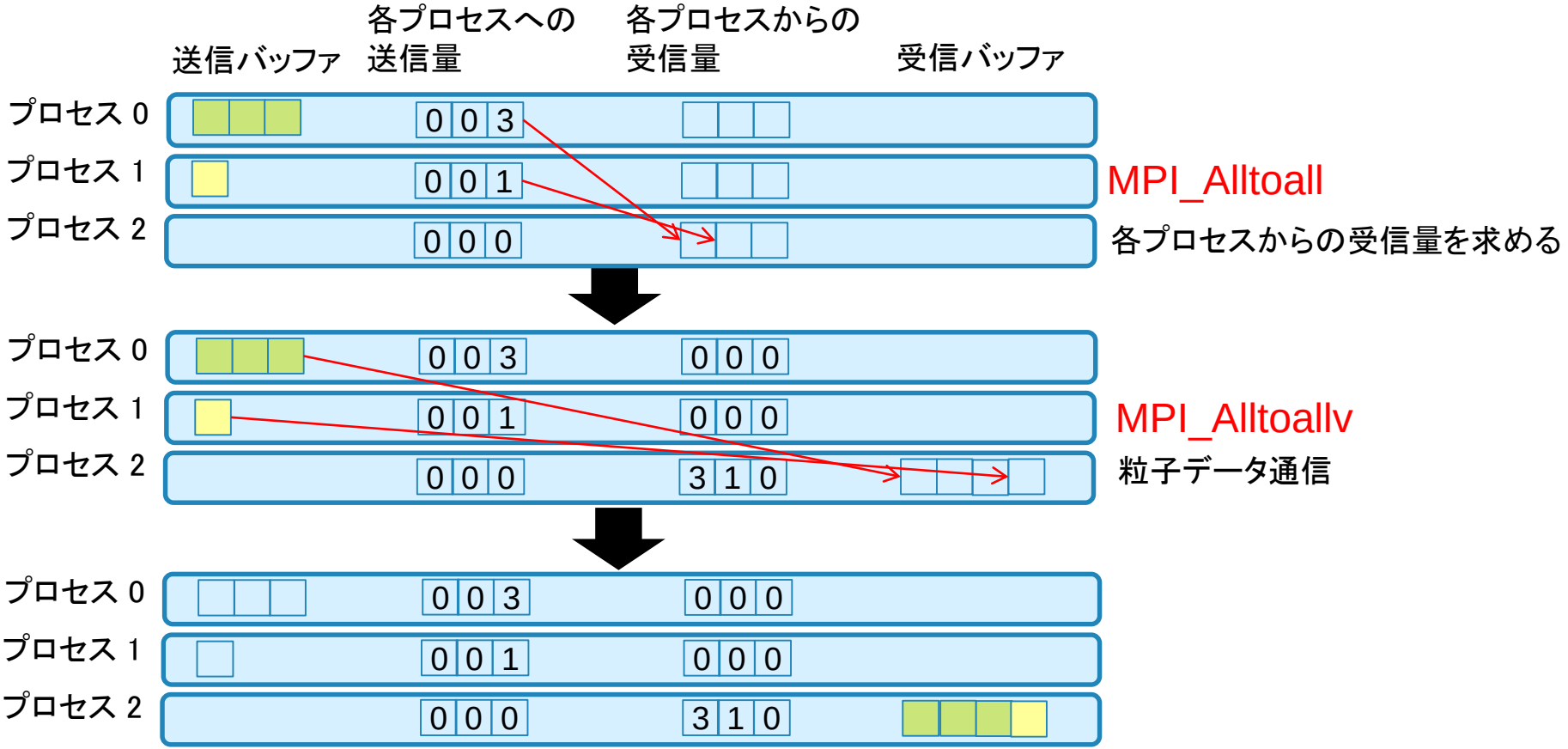
MPI_Alltoallv



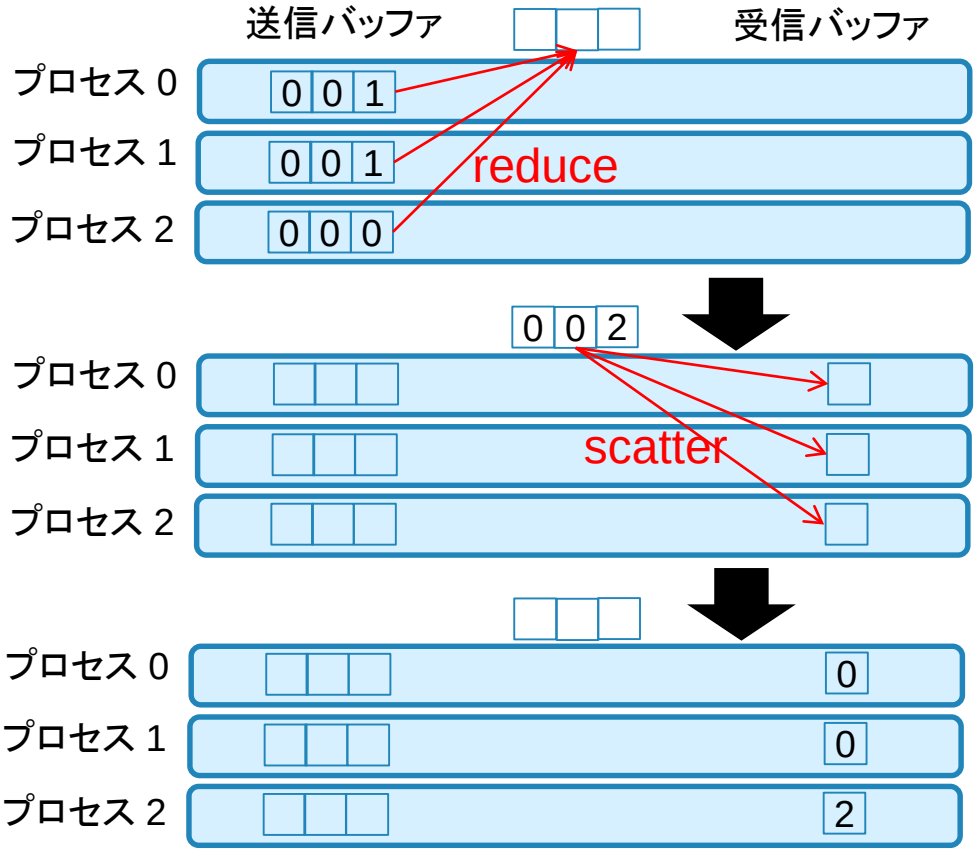
送信プロセス, 受信プロセスの組み合わせによって通信量が異なってもよい

通信しない場合は通信量0

all-to-all×2による実装

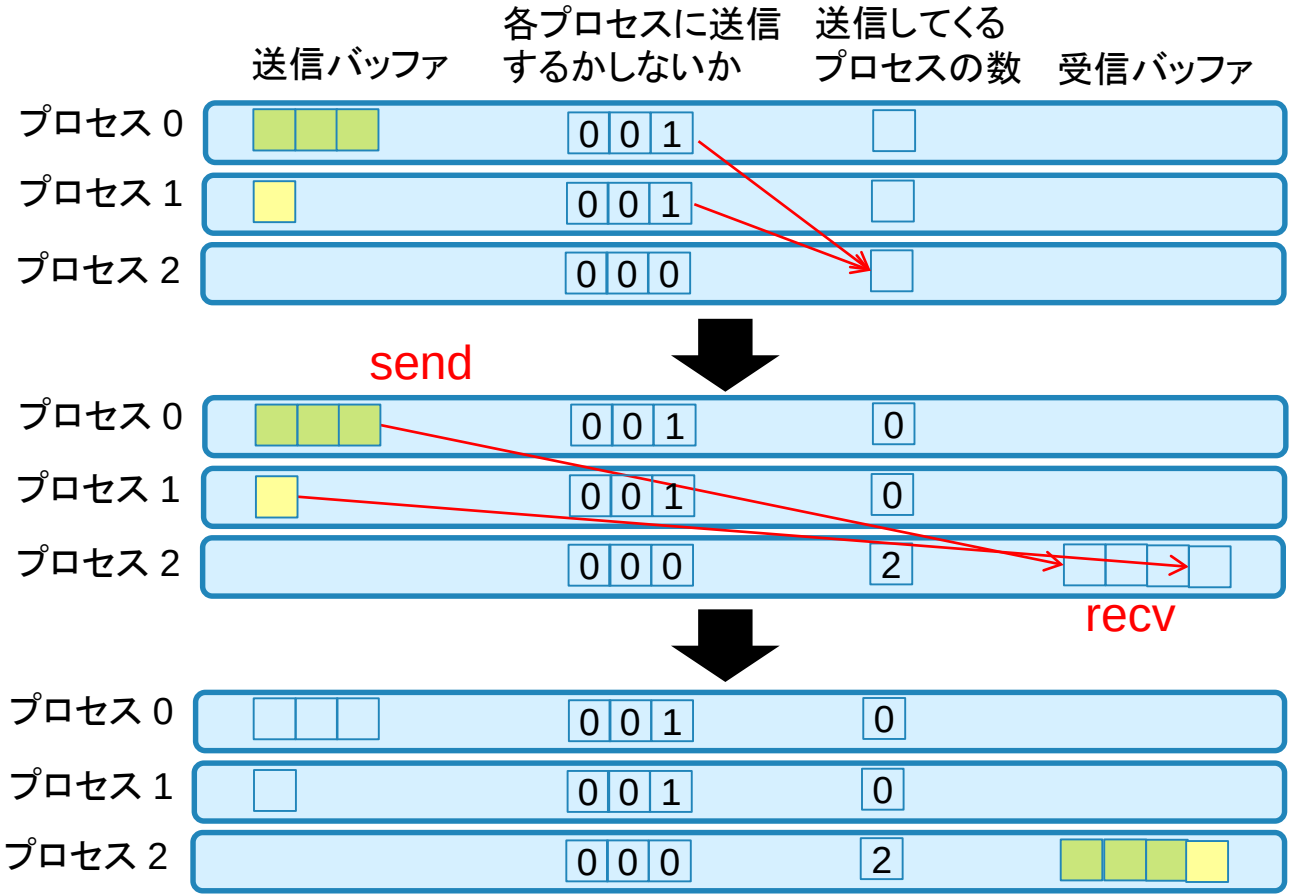


reduce-scatter通信

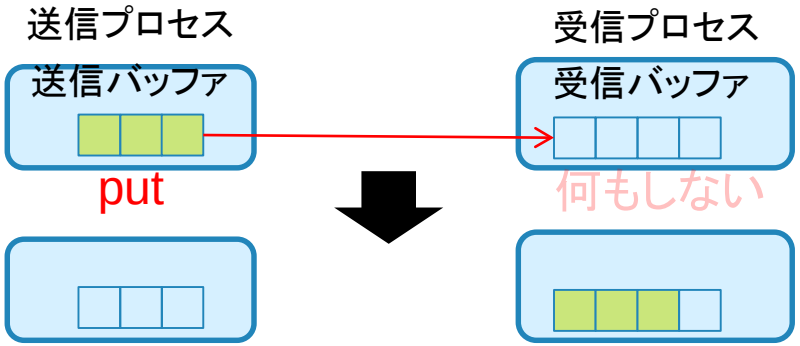


MPI_Reduce_scatter

reduce-scatter + send-recvによる実装



片側(one-sided)通信



受信プロセスはいくつのプロセスが送信してくるか、
知らなくてよい
受信プロセスにおける書き込み位置は
送信プロセスが指定

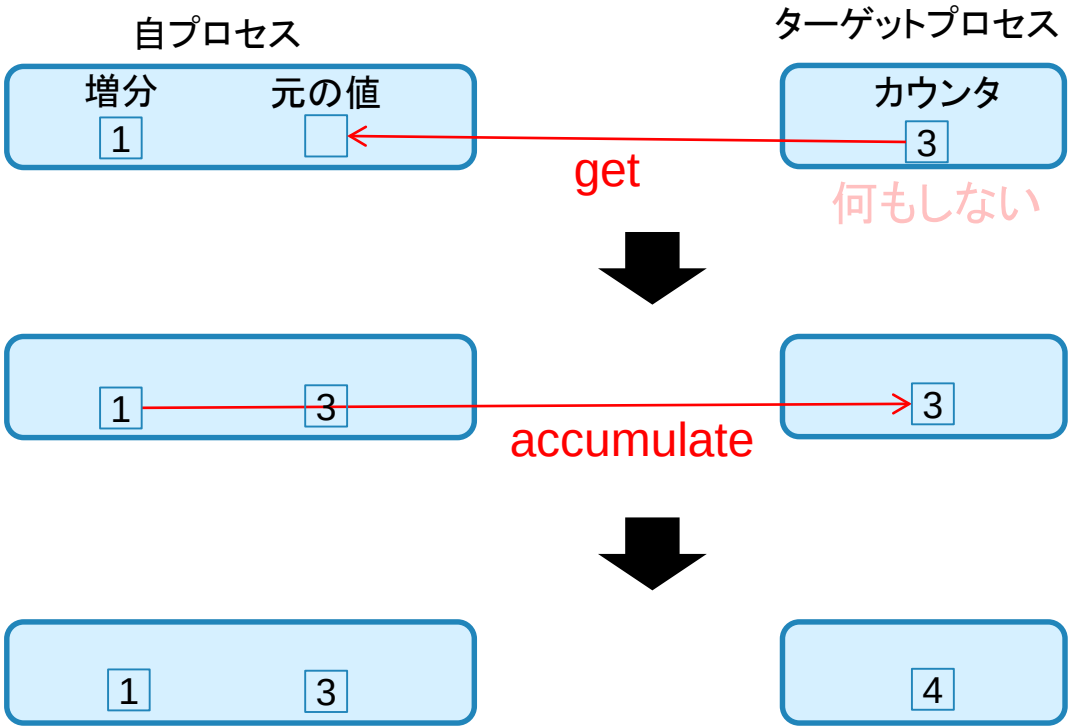
MPIも片側通信をサポートしているが、あまり利用されていない

粒子データ通信では受信プロセスはいくつのプロセスが送信してくるかわからない

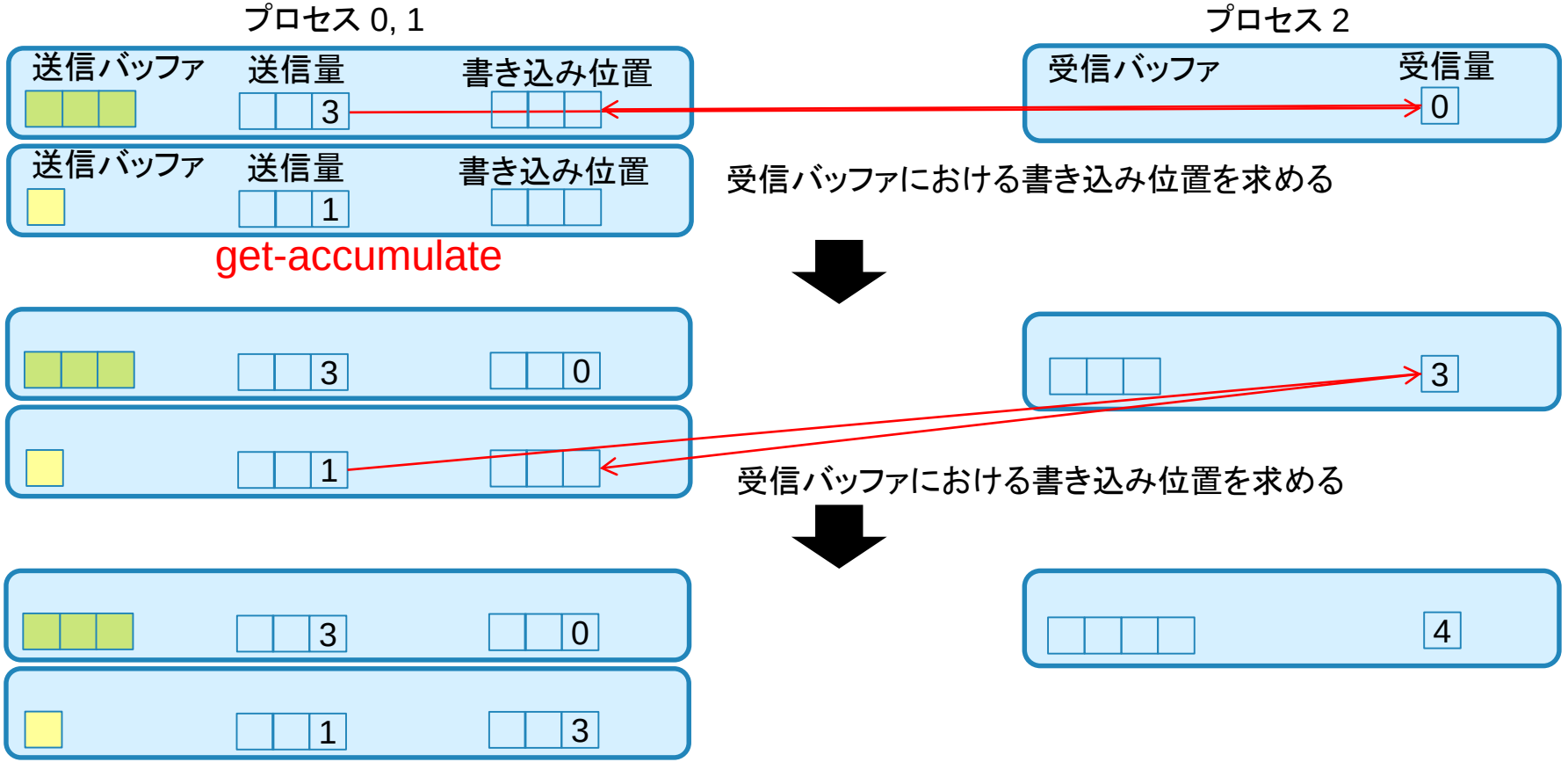
片側通信は粒子データ通信に適している？

get-accumulate通信

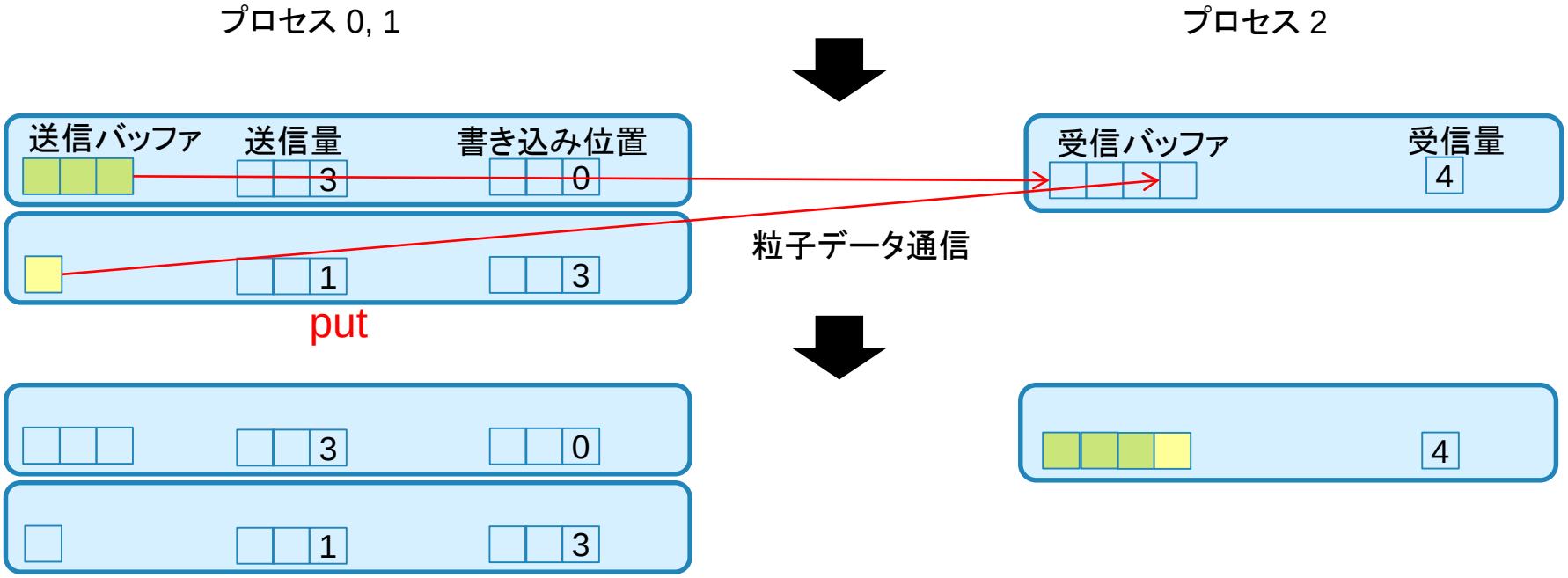
- 片側通信の1つ



片側通信による実装 1


プロセス 0, 1
プロセス 2
送信バッファ
送信量
書き込み位置
受信バッファ
受信量
3
0
1
受信バッファにおける書き込み位置を求める
get-accumulate
受信バッファにおける書き込み位置を求める
4

片側通信による実装 2


プロセス 0, 1
プロセス 2
送信バッファ
送信量
書き込み位置
3
0
受信バッファ
受信量
4
1
3
粒子データ通信
put
3
0
4
1
3

ACP通信ライブラリ

- プロセス並列アプリケーションのための通信ライブラリ
- 片側通信を基盤としている

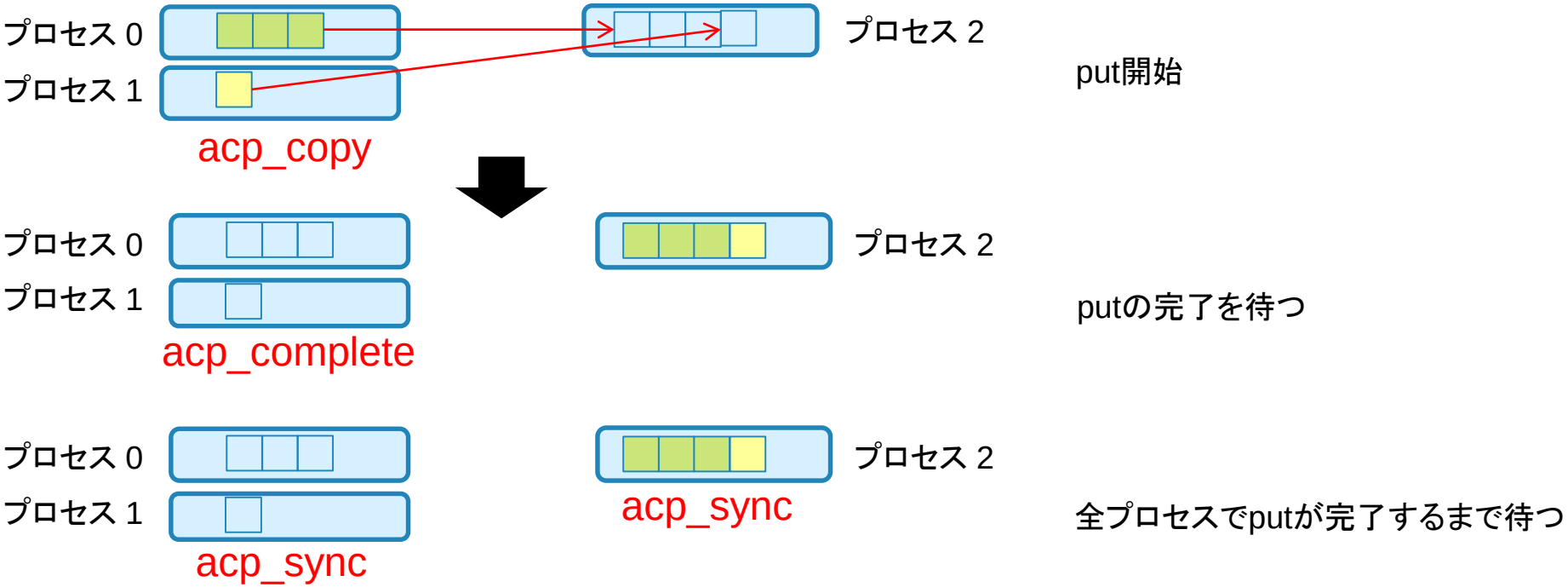
  MPIはsend-recv型通信を基盤としている

- ACP基本層はMPIよりプリミティブ

  通信関数が呼ばれると直ちに通信を開始する

  MPIの片側通信は通信関数が呼ばれても直ちに通信を開始するとは限らない

ACPの片側通信


プロセス 0
プロセス 1
プロセス 2
acp_copy
put開始
プロセス 0
プロセス 1
プロセス 2
acp_complete
putの完了を待つ
プロセス 0
プロセス 1
プロセス 2
acp_sync
acp_sync
全プロセスでputが完了するまで待つ

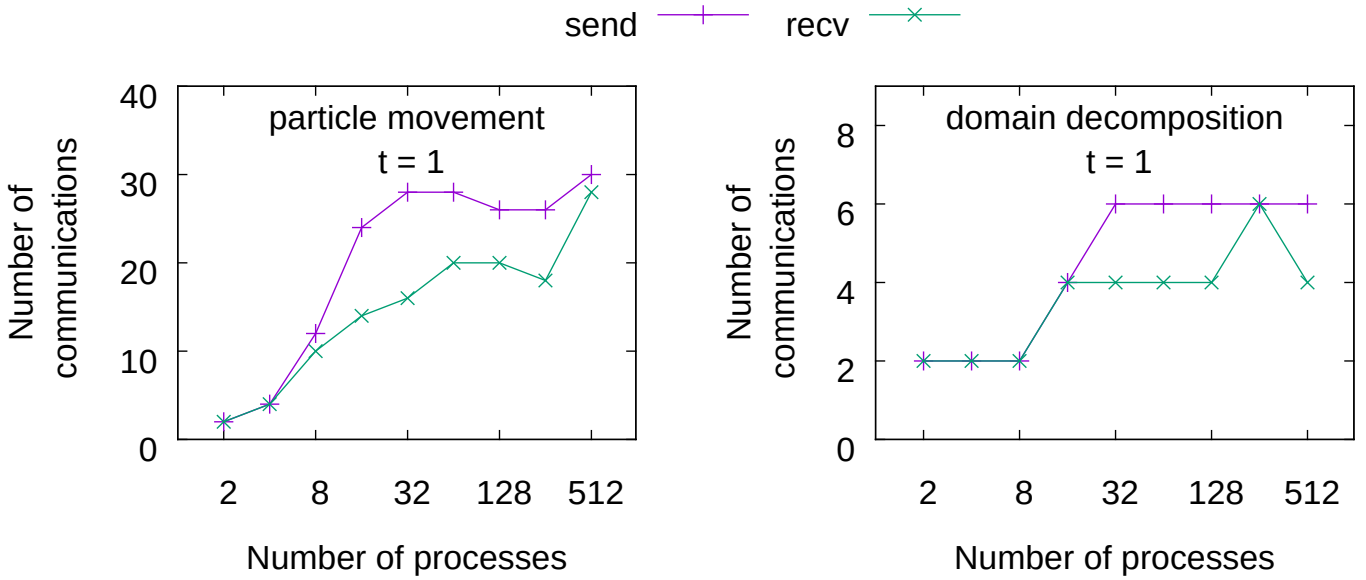
MPIとACPの通信性能 InfiniBand

- send-recv, putの性能

ライブラリ	通信	最小レイテンシ (μs)	2MBにおける速度 (GB/s)
Intel MPI 5.1.3	send-recv	2.5	3.6
	put+同期	5.0	4.1
	put	0.8	4.6
ACP	put+同期	7.9	4.7
	put	0.4	4.4

富士通PRIMERGY CX400
InfiniBand FDR (6.78 GB / s)
ノード間通信

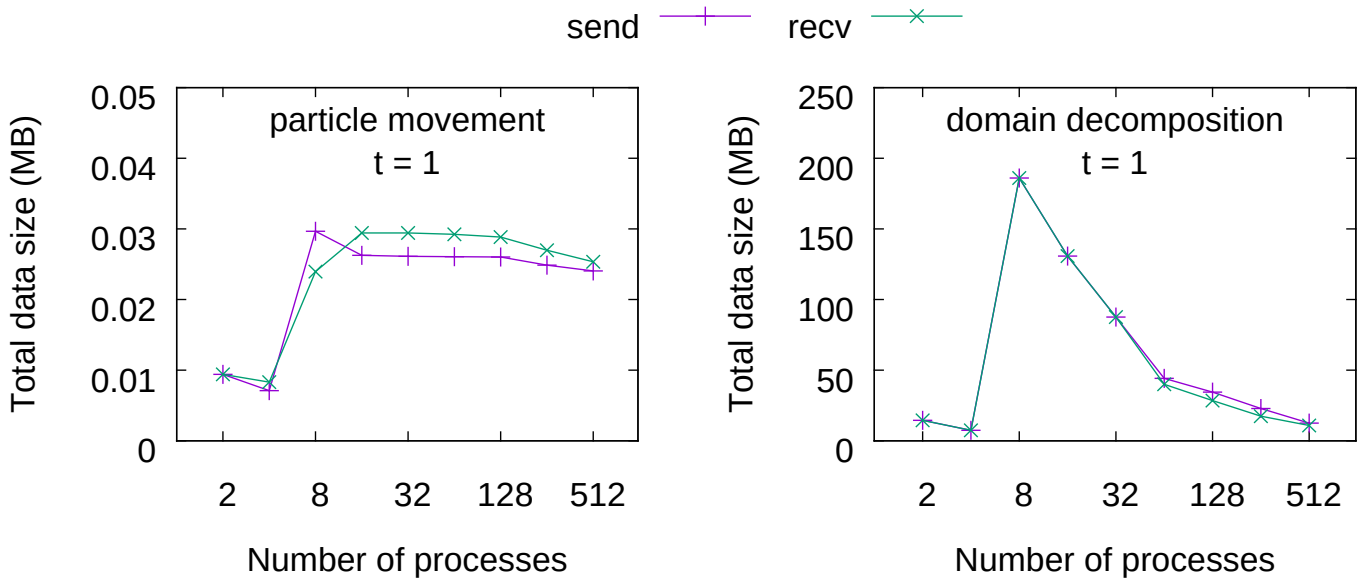
粒子データ通信の通信回数 1



270^3 粒子

送信回数, 受信回数が最大のプロセス

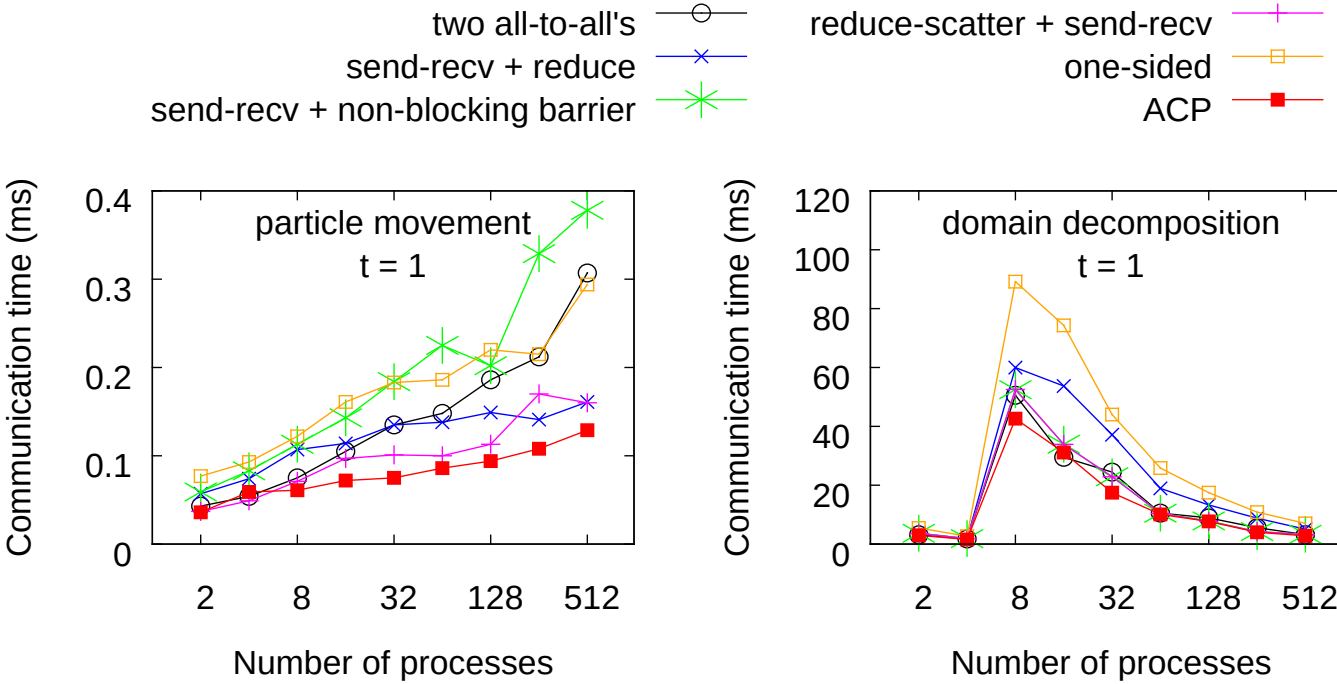
粒子データ通信の通信量 1



270^3 粒子

送信量, 受信量が最大のプロセス

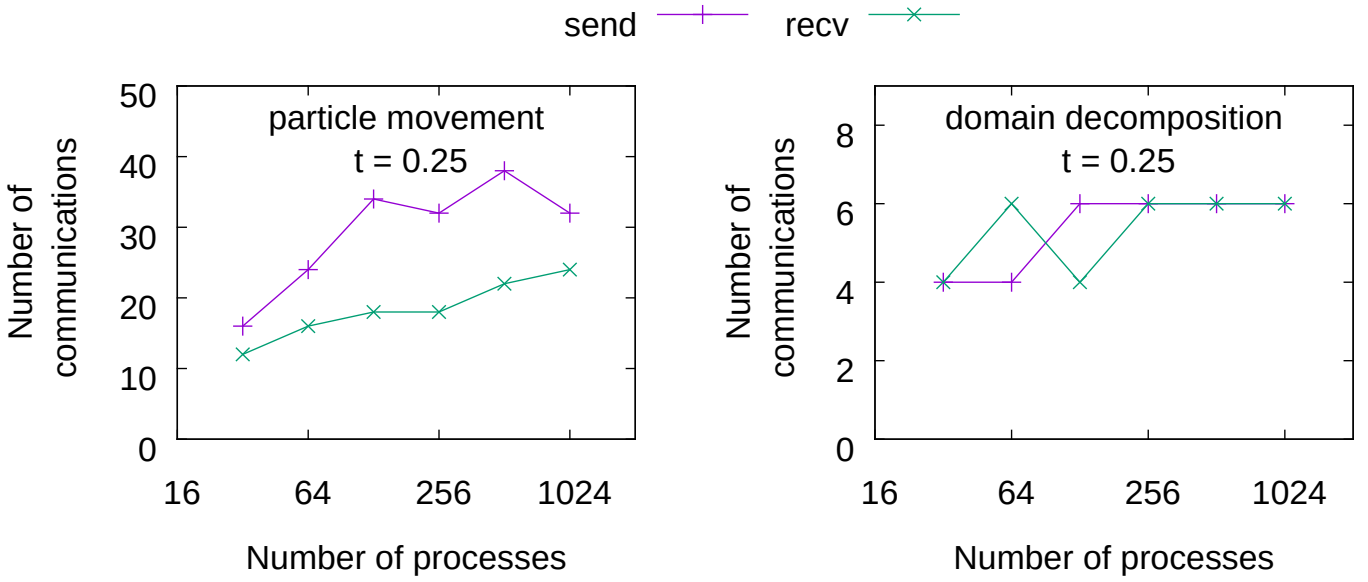
粒子データ通信の性能比較 InfiniBand 1



ACPまたはreduce-scatter+send-recvが最速

270^3 粒子
Intel MPI 5.1.3
1プロセス / ノード
OpenMPIはやや遅い

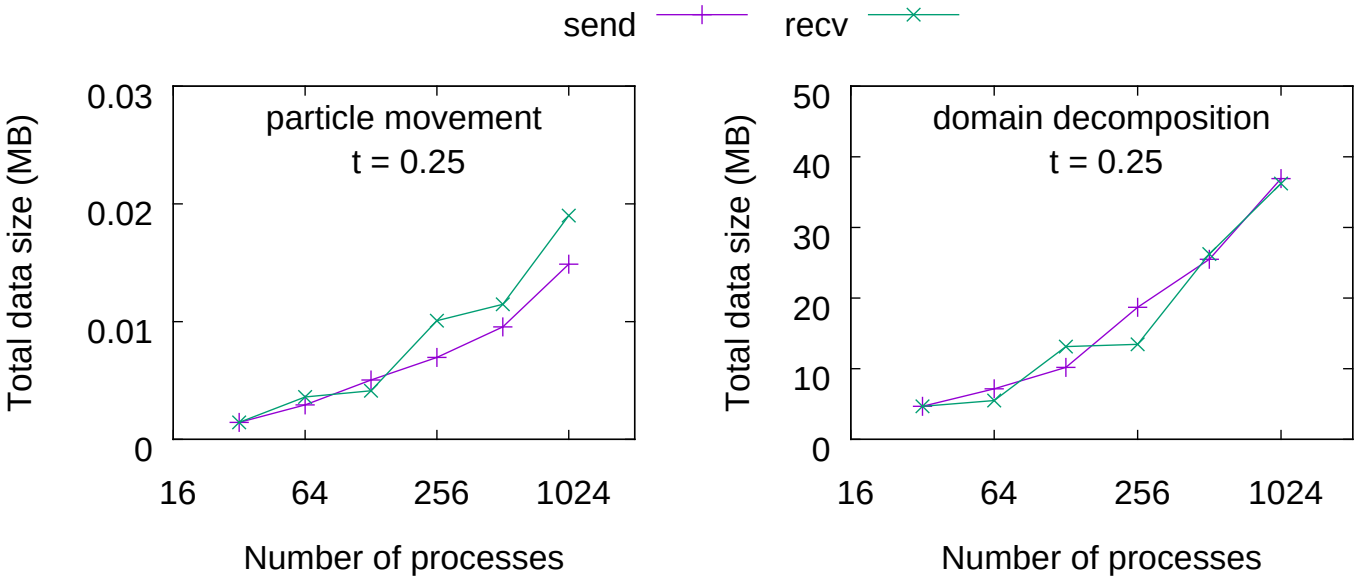
粒子データ通信の通信回数 2



100^3 – 682^3粒子

送信回数, 受信回数が最大のプロセス

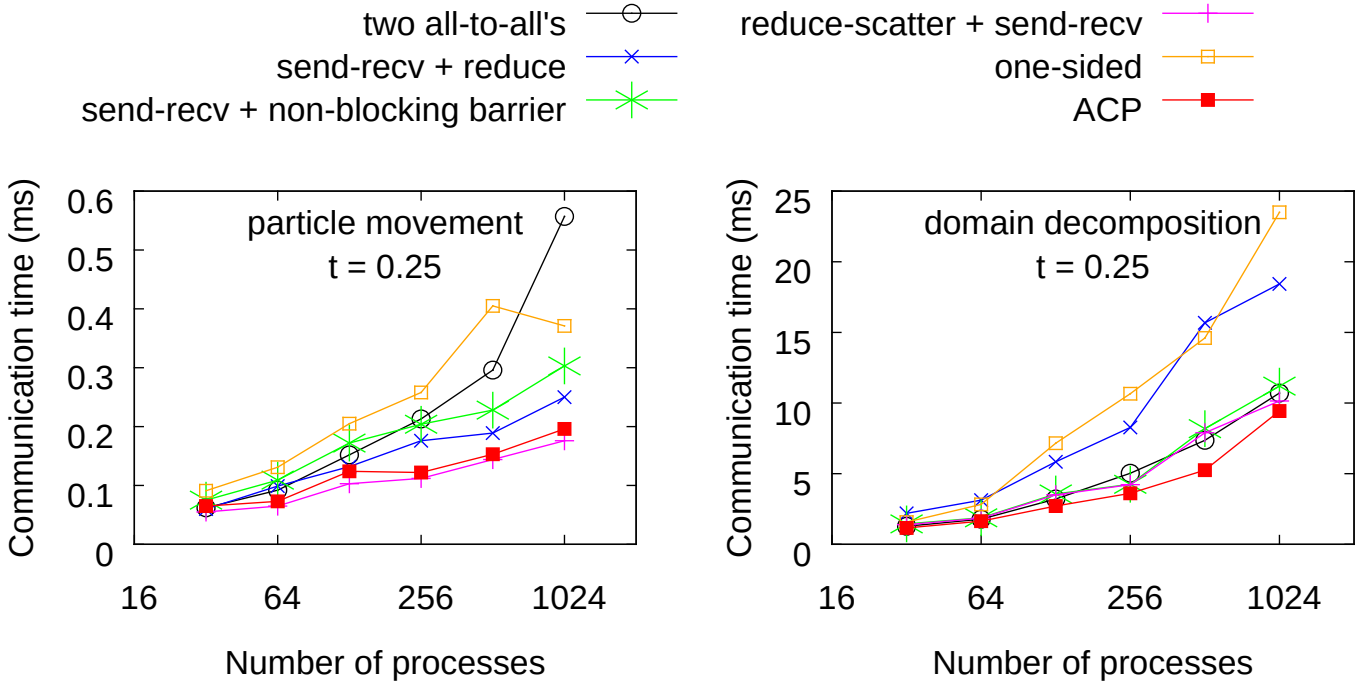
粒子データ通信の通信量 2



100^3 – 682^3粒子

送信量, 受信量が最大のプロセス

粒子データ通信の性能比較 InfiniBand 2



ACPまたはreduce-scatter+send-recvが最速

100^3 – 682^3粒子
Intel MPI 5.1.3
1プロセス / ノード
OpenMPIはやや遅い

MPIとACPの通信性能 Tofuインターコネクト

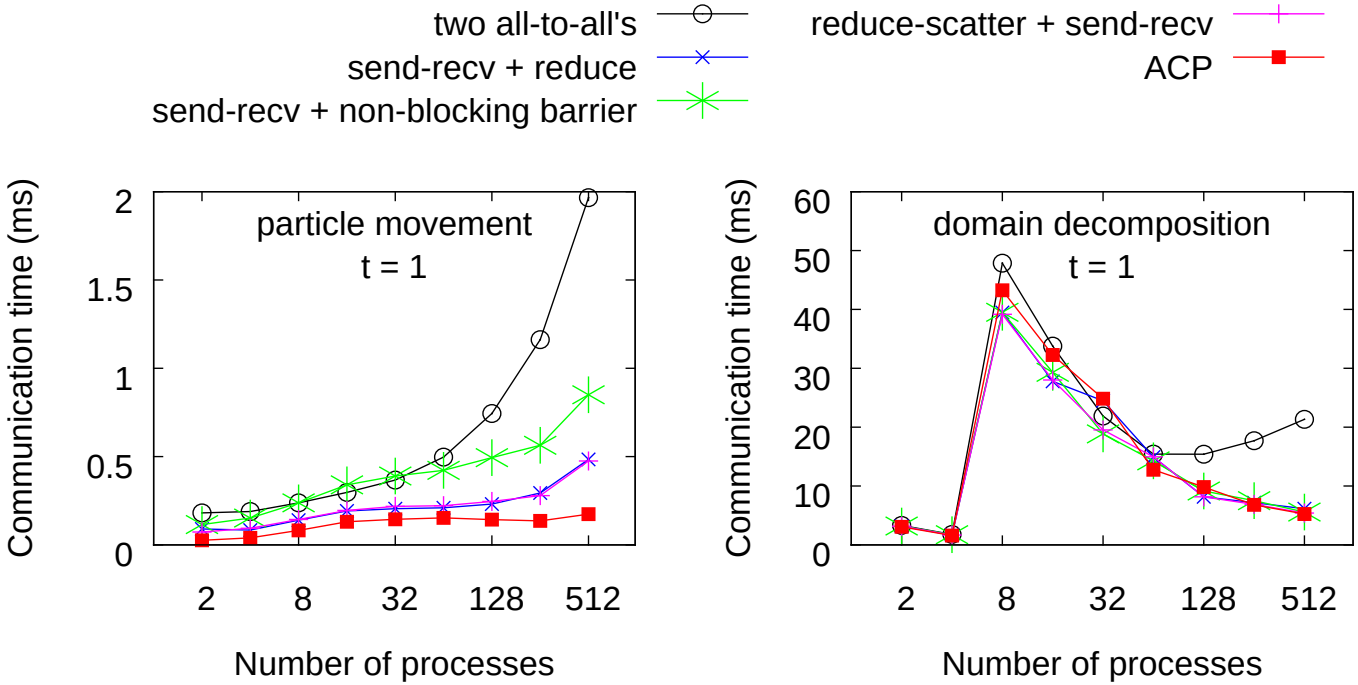
- send-recv, putの性能

ライブラリ	通信	最小レイテンシ (μs)	2MBにおける速度 (GB/s)
富士通 MPI 1.2.1	send-recv	1.6	4.6
ACP	put+同期	5.3	4.7
	put	2.2	4.7

富士通PRIMEHPC FX10
Tofu (5 GB / s)
ノード間通信

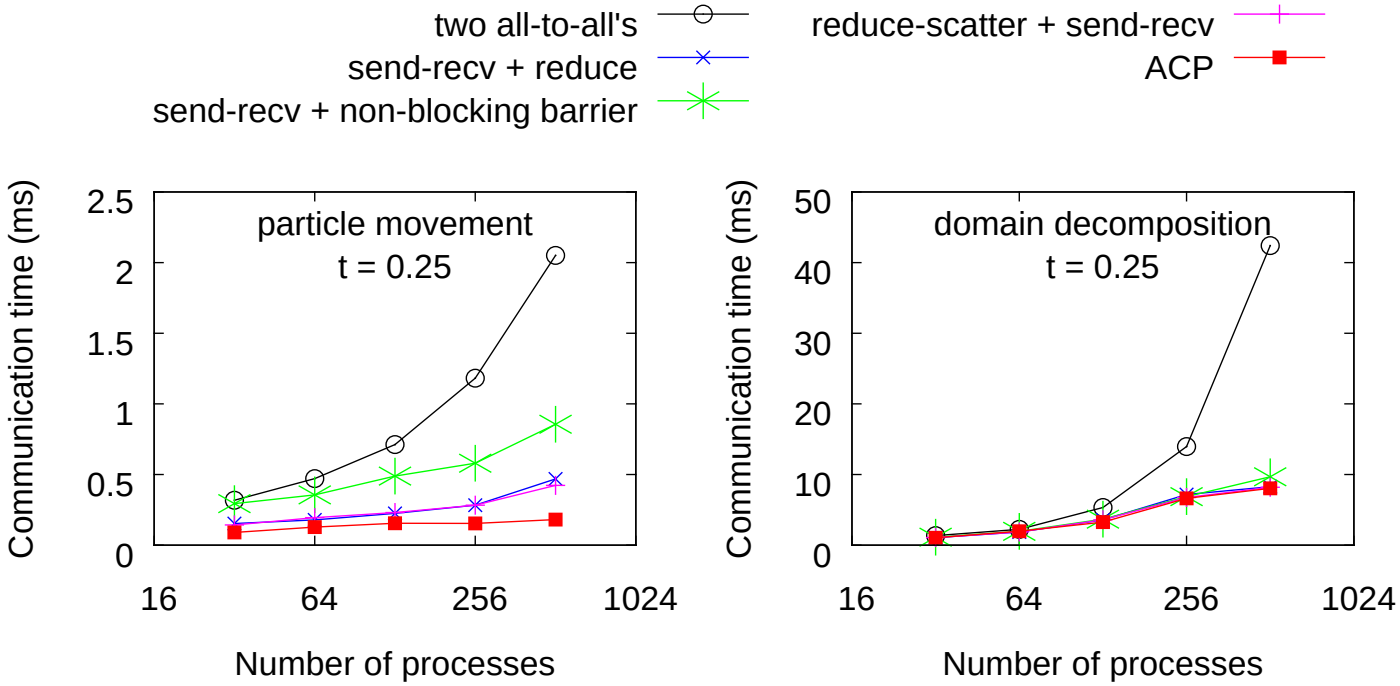
富士通 MPI 1.2.1はget-accumulateをサポートしていないため、片側通信による実装はない

粒子データ通信の性能比較 Tofu 1



270^3 粒子
富士通MPI 1.2.1
1プロセス / ノード

粒子データ通信の性能比較 Tofu 2



100^3 – 464^3粒子
富士通MPI 1.2.1
1プロセス / ノード

reduce-scatter+send-recvが速い理由

MPIは送信関数が呼ばれても直ちに通信を開始するとは限らない
通信がいつ開始するかは受信プロセスの動作にも依存する

実装	通信ライブラリ	プロセス数=Pの時の最小コスト	粒子データが到着しているか受信前にチェック
all-to-all×2	MPI	$O(P)$	×
send-recv + reduce		$O(\log P)$	○
send-recv + non-blocking barrier		$O(\log P)$	○
reduce-scatter + send-recv		$O(P)$	× ?
片側通信		$O(\log P)$	
片側通信	ACP	$O(\log P)$	

まとめ

- 粒子系シミュレーションにおける粒子データ通信は
  send-recv型通信で簡単かつ効率的に扱うことが困難
- 複数の通信を組み合わせたアルゴリズムが考案されている
- さまざまな状況の粒子データ通信に対して、
  ACPまたはreduce-scatter + send-recvによる実装が最速または最速に近い
- ACPによる実装はMPIの片側通信による実装よりも速い